

# Introduction to C

by Olve Maudal



a one day course

(Last update: 2013 July 4, 09:04)

# Course Outline

## **Introduction**

Exercise 0 (Hello, world!)

## **A Brief Tour**

Exercise 1 (Deep Thought)

## **Control Flow**

Exercise 2 (Prime Numbers)

## **Test-Driven Development**

Exercise 3 (Phone Book)

## **Summary**

**Pizza and Raspberry Pi hacking**

# Exercise

hello.c

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

Type in this code. Compile and execute the program.

Discuss each line of this program and try to explain what it does.  
Do a "dry run", follow the flow of execution and discuss what happens.

# A Brief Tour

tools and vocabulary



# A<sup>✓not so</sup> Brief Tour

tools and vocabulary



hello.c

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

hello.c

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

```
$ cc -o hello hello.c
```



hello.c

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

```
$ cc -o hello hello.c
$ ./hello
```



hello.c

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

```
$ cc -o hello hello.c
$ ./hello
The answer is 42
```

hello.c

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

```
$ cc -o hello hello.c
$ ./hello
The answer is 42
$ echo $?
```

hello.c

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

```
$ cc -o hello hello.c
$ ./hello
The answer is 42
$ echo $?
0
```

hello.c

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

```
$ cc -o hello hello.c
$ ./hello
The answer is 42
$ echo $?
0
$
```

**Was this the result you expected?**

Was this the result you expected?

Of course it was!

Was this the result you expected?

Of course it was!

But let's do a dry run and step through the code



Was this the result you expected?

Of course it was!

But let's do a dry run and step through the code

(we will look at the compilation step later.)



Was this the result you expected?

Of course it was!

But let's do a dry run and step through the code

(we will look at the compilation step later.)

The following is an example of what ***might*** happen when executing this code:

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

After some basic initialization the run-time environment will call the main function.

```
#include <stdio.h>
```

```
static int calc(int a, int b, int c)
{
    return a * b / c;
}
```

```
int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }
```

→ 

```
int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```



then preparation for the call to `calc()` starts by evaluating all the arguments to be passed to the function.

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

guess which argument is evaluated first?

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```





```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```



unlike other popular programming languages, the order of evaluation is mostly unspecified in C. In this case the compiler or runtime environment might choose to evaluate `life()` first



```
#include <stdio.h>
```

```
static int calc(int a, int b, int c)
{
    return a * b / c;
}
```

```
int universe = 7;
```



```
static int life(void) { return 6; }
```

```
int everything(void) { return 1; }
```

```
int main(void)
```

```
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, 6, everything());
    printf("The answer is %d\n", a);
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, 6, everything());
    printf("The answer is %d\n", a);
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, 6, everything());
    printf("The answer is %d\n", a);
}
```



universe is replaced with 7.

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc( 7 , 6 , everything());
    printf("The answer is %d\n", a);
}
```





```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    , everything());
    printf("The answer is %d\n", a);
}
```





```
#include <stdio.h>
```

```
static int calc(int a, int b, int c)
{
    return a * b / c;
}
```

```
int universe = 7;
```

```
static int life(void) { return 6; }
```

→ 

```
int everything(void) { return 1; }
```

```
int main(void)
```

```
{
    int a = calc(    7    ,    6    , everything());
    printf("The answer is %d\n", a);
}
```

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    , everything());
    printf("The answer is %d\n", a);
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    , everything());
    printf("The answer is %d\n", a);
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}
```



now we are ready to call the `calc()` function. This can be done by pushing arguments on an execution stack, reserve space for the return value and perhaps some housekeeping values.

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc( 7 , 6 , 1 );
    printf("The answer is %d\n", a);
}
```

now we are ready to call the `calc()` function. This can be done by pushing arguments on an execution stack, reserve space for the return value and perhaps some housekeeping values.



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc( 7 , 6 , 1 );
    printf("The answer is %d\n", a);
}
```

now we are ready to call the `calc()` function. This can be done by pushing arguments on an execution stack, reserve space for the return value and perhaps some housekeeping values.

guess which argument is pushed first?



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}
```

the way arguments are passed to a function is defined by the calling convention (ABI) used by the compiler and runtime environment. Here is just an example...

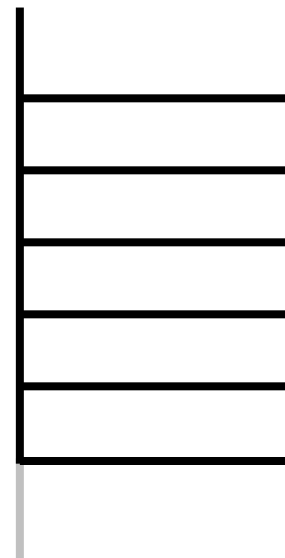
```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}
```

the way arguments are passed to a function is defined by the calling convention (ABI) used by the compiler and runtime environment. Here is just an example...



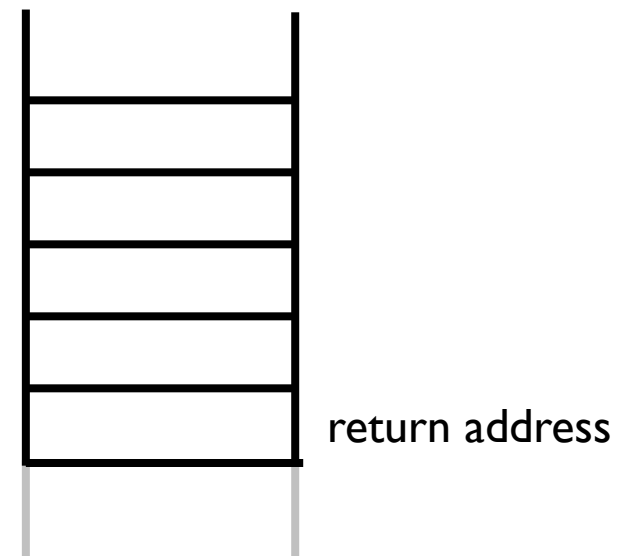
```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}
```

the way arguments are passed to a function is defined by the calling convention (ABI) used by the compiler and runtime environment. Here is just an example...



```

#include <stdio.h>

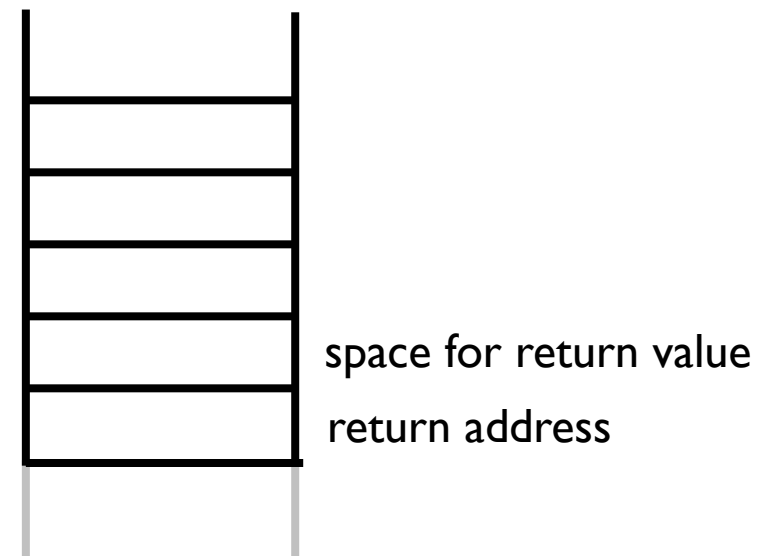
static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}

```

the way arguments are passed to a function is defined by the calling convention (ABI) used by the compiler and runtime environment. Here is just an example...



```

#include <stdio.h>

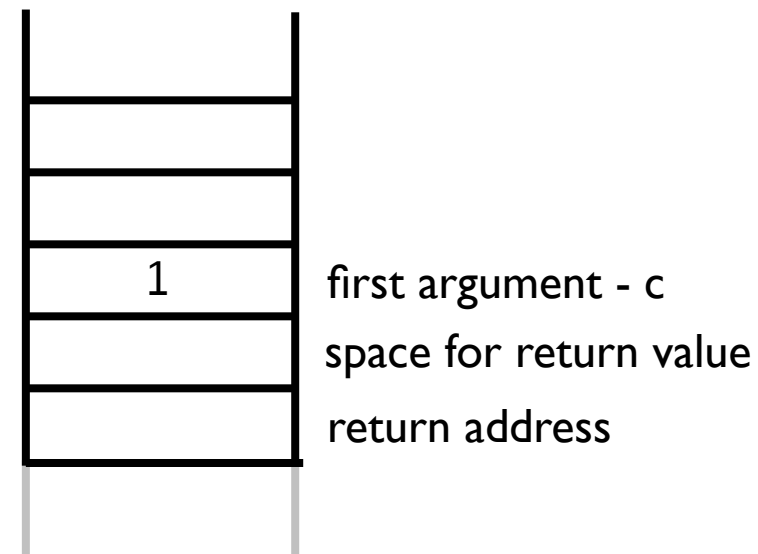
static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}

```

the way arguments are passed to a function is defined by the calling convention (ABI) used by the compiler and runtime environment. Here is just an example...



```

#include <stdio.h>

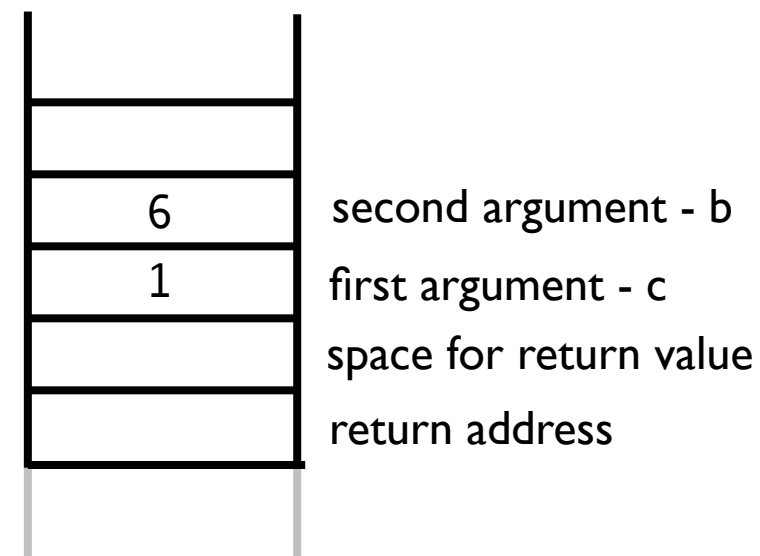
static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}

```

the way arguments are passed to a function is defined by the calling convention (ABI) used by the compiler and runtime environment. Here is just an example...



```

#include <stdio.h>

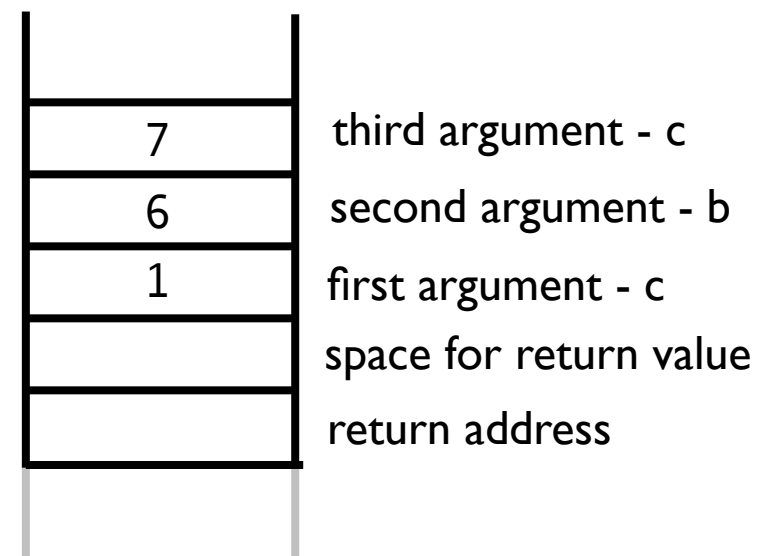
static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}

```

the way arguments are passed to a function is defined by the calling convention (ABI) used by the compiler and runtime environment. Here is just an example...





```

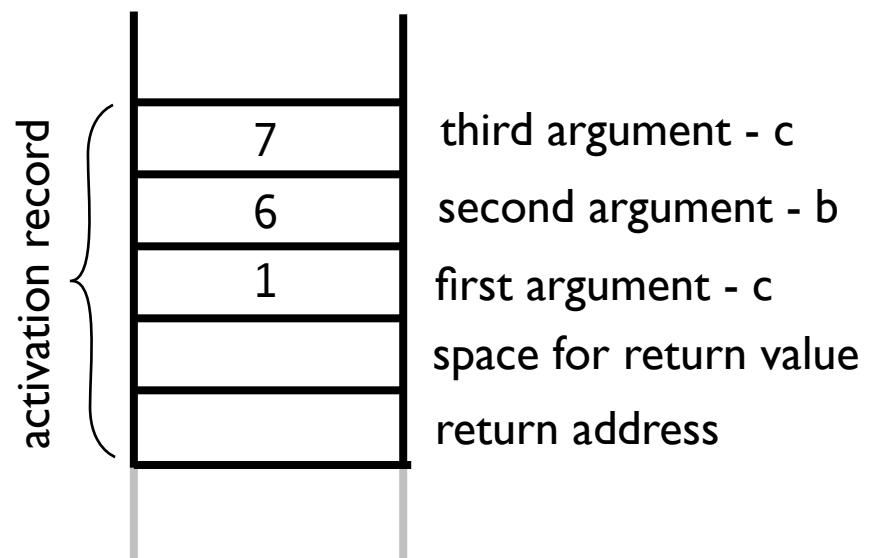
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}

```





```

#include <stdio.h>

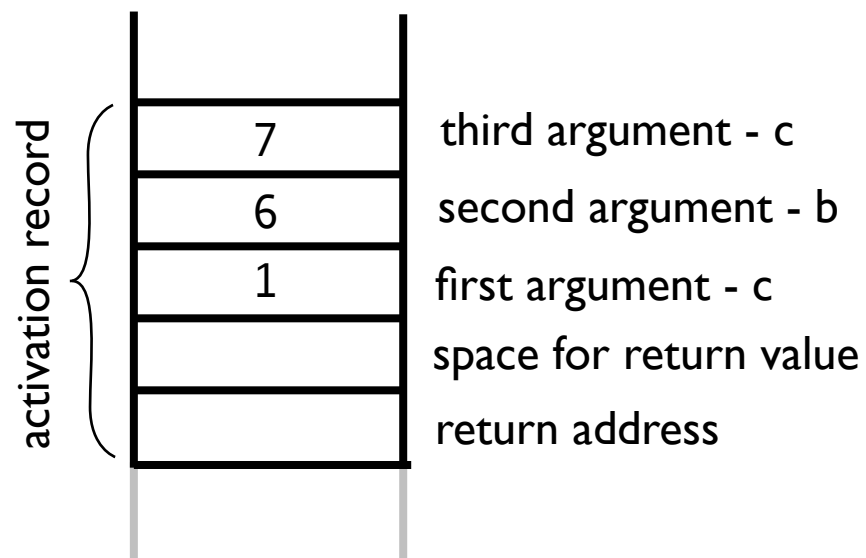
static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}

```

and when the “activation record” is populated, the program can jump into the function.





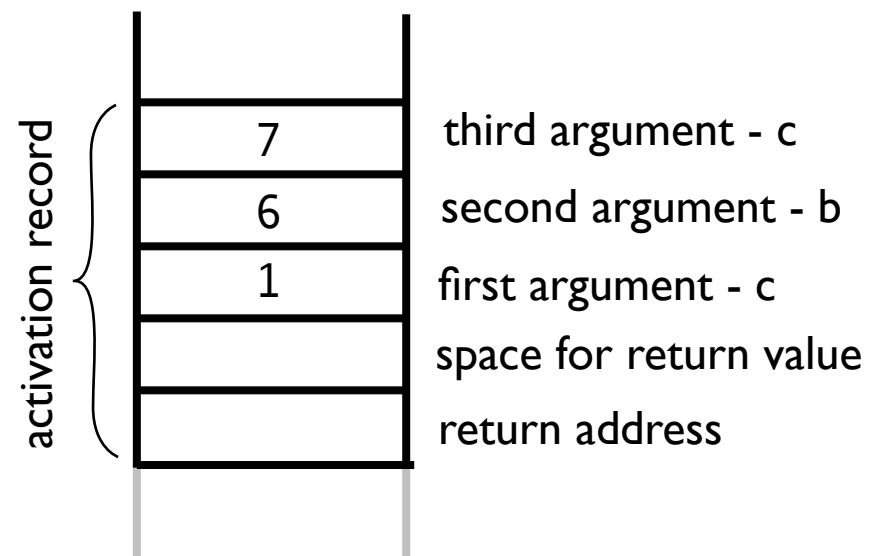
```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}
```

and when the “activation record” is populated, the program can jump into the function.



```
#include <stdio.h>
```

```
static int calc(int a, int b, int c)
```

```
{
```

```
    return a * b / c;
```

```
}
```

```
int universe = 7;
```

```
static int life(void) { return 6; }
```

```
int everything(void) { return 1; }
```

```
int main(void)
```

```
{
```

```
    int a = calc(    7    ,    6    ,    1    );
```

```
    printf("The answer is %d\n", a);
```

```
}
```

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}
```



and then evaluate the expression.

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 7 * 6 / 1;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}
```

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 7 * 6 / 1;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(    7    ,    6    ,    1    );
    printf("The answer is %d\n", a);
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42 / 1;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(7, 6, 1);
    printf("The answer is %d\n", a);
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(7, 6, 1);
    printf("The answer is %d\n", a);
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(7, 6, 1);
    printf("The answer is %d\n", a);
}
```

```

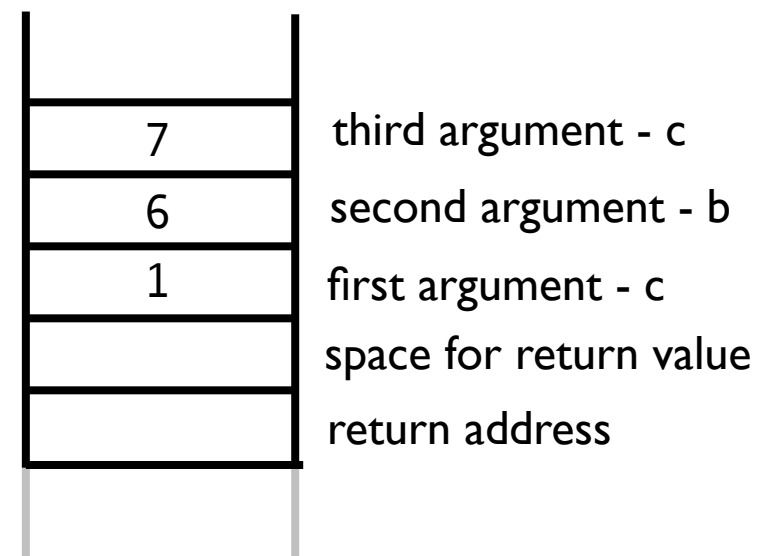
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(7, 6, 1);
    printf("The answer is %d\n", a);
}

```



```

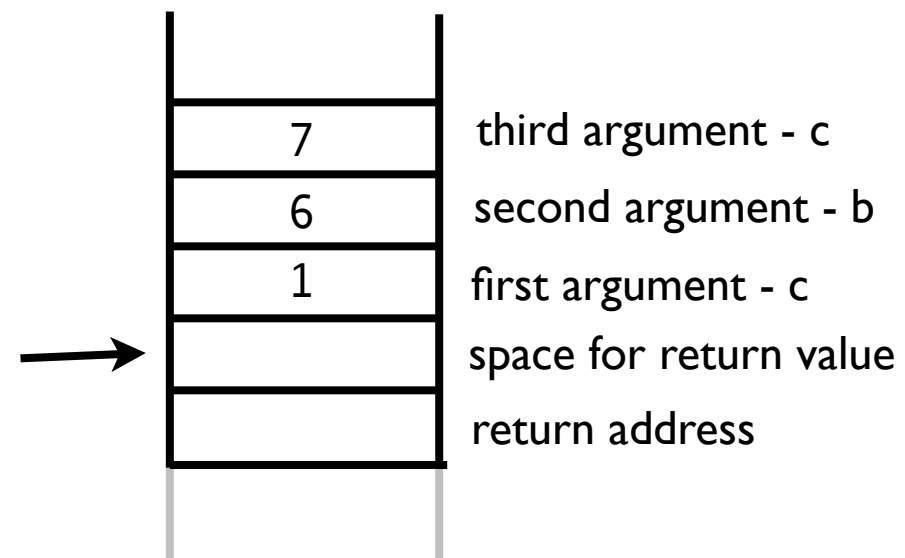
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(7, 6, 1);
    printf("The answer is %d\n", a);
}

```



```

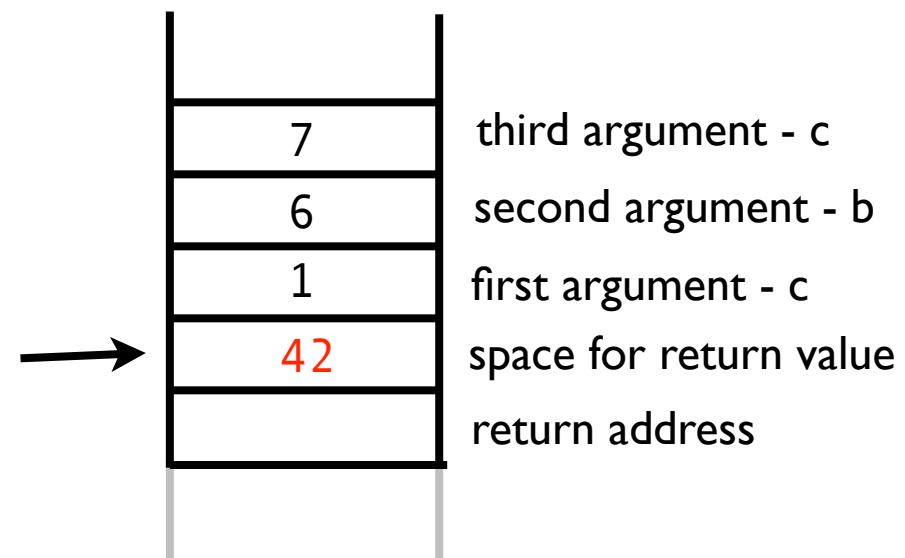
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(7, 6, 1);
    printf("The answer is %d\n", a);
}

```



```

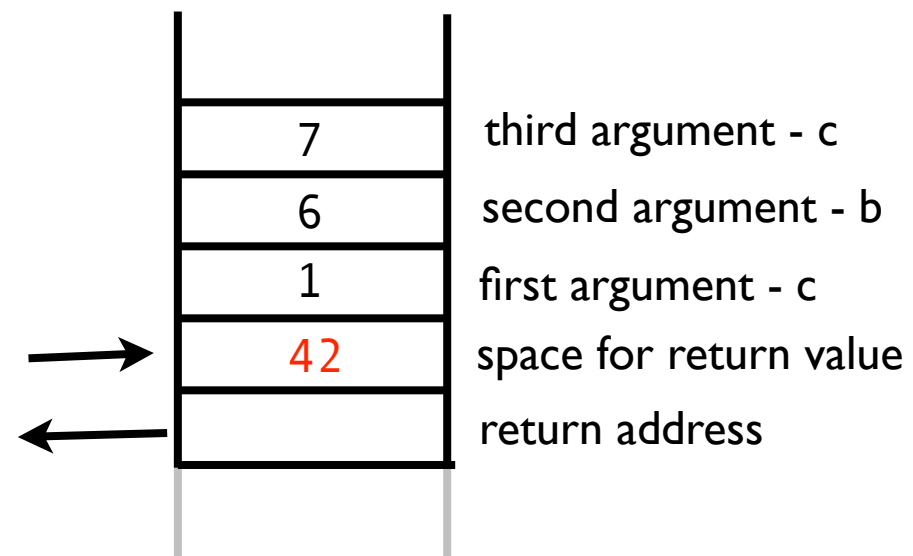
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(7, 6, 1);
    printf("The answer is %d\n", a);
}

```



```

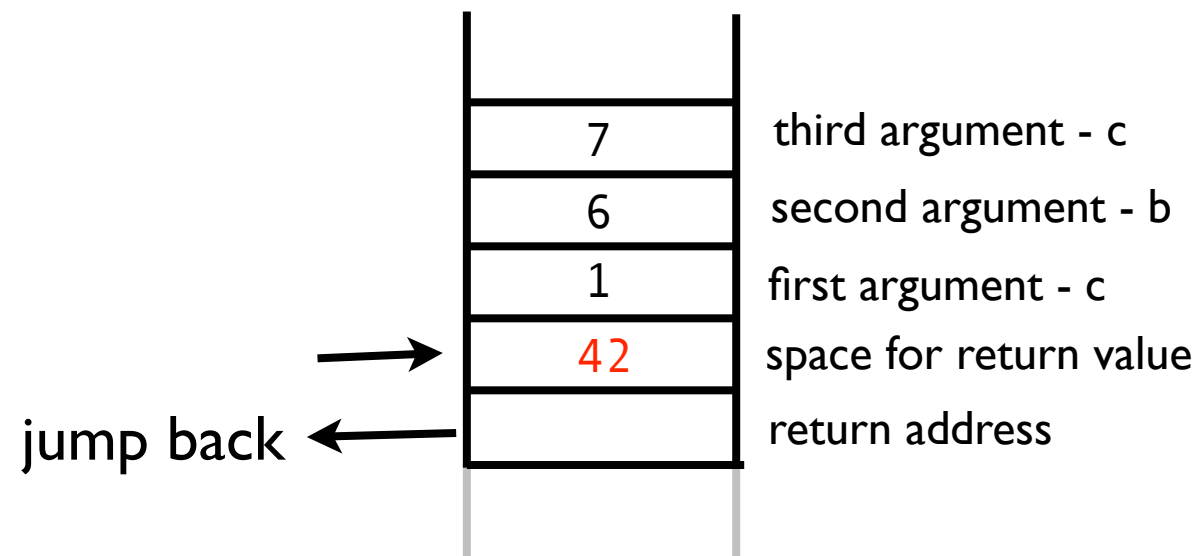
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42 ;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc( 7 , 6 , 1 );
    printf("The answer is %d\n", a);
}

```



```

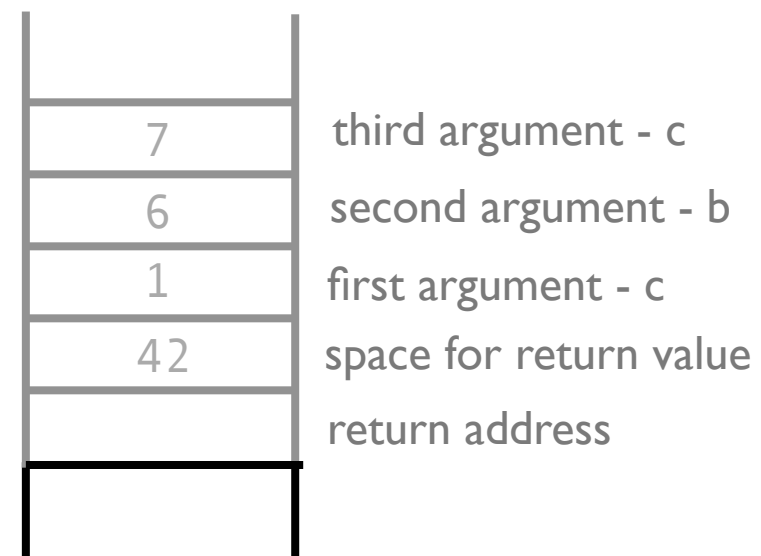
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    → int a = calc( 7 , 6 , 1 );
    printf("The answer is %d\n", a);
}

```





```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = 42;
    printf("The answer is %d\n", a);
}
```





```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42 ;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = 42;
    printf("The answer is %d\n", a);
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42 ;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = 42;
    printf("The answer is %d\n", 42);
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42 ;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = 42;
    printf("The answer is %d\n", 42);
}
```

and then 42 and the pointer to the character string is pushed on the execution stack before the library function `printf()` is called

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = 42;
    printf("The answer is %d\n", 42);
}
```

The `printf()` function writes out  
to the standard output stream

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = 42;
    printf("The answer is %d\n", 42);
}
```

The answer is 42

The printf() function writes out  
to the standard output stream

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = 42;
    printf("The answer is %d\n", 42);
}
```



The answer is 42

The printf() function writes out  
to the standard output stream

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return 42;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = 42;
    printf("The answer is %d\n", 42);
}
```

```
The answer is 42
$ echo $?
0
```

and a default value to indicate success, in this case 0, is returned back to the run-time environment



Was this exactly what you expected?



Was this exactly what you expected?

Good!

Was this exactly what you expected?

Good!

This was just an example of what ***might*** happen.

Was this exactly what you expected?

Good!

This was just an example of what ***might*** happen.

Because...

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

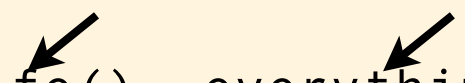
int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

maybe the compiler is clever and see that `life()` and `everything()` always returns 6 and 1





```
#include <stdio.h>
```

```
static int calc(int a, int b, int c)
{
    return a * b / c;
}
```

```
int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }
```

```
int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```



maybe the compiler is clever and see that life() and everything() always returns 6 and 1





```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

maybe the compiler is clever and see that `life()` and `everything()` always returns 6 and 1

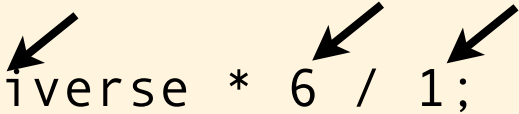
and then by inlining `calc()` perhaps the compiler choose to optimize the code into...

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = universe * 6 / 1;
    printf("The answer is %d\n", a);
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = universe * 6 / 1;
    printf("The answer is %d\n", a);
}
```

and since nobody else uses the functions perhaps the compiler decides to not create code for `life()` and `calc()`

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = universe * 6 / 1;
    printf("The answer is %d\n", a);
}
```

and since nobody else uses the functions perhaps the compiler decides to not create code for `life()` and `calc()`

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = universe * 6 / 1;
    printf("The answer is %d\n", a);
}
```

and since nobody else uses the functions perhaps the compiler decides to not create code for `life()` and `calc()`

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = universe * 6 / 1;
    printf("The answer is %d\n", a);
}
```

and since nobody else uses the functions perhaps the compiler decides to not create code for `life()` and `calc()`

and since variable `a` is used only here, then it might skip creating object `a` and just evaluate the expression as part of the `printf()` expression.

```
#include <stdio.h>
```

```
static int calc(int a, int b, int c)
```

```
{
```

```
    return a * b / c;
```

```
}
```

```
int universe = 7;
```

```
static int life(void) { return 6; }
```

```
int everything(void) { return 1; }
```

```
int main(void)
```

```
{
```

```
    int a = universe * 6 / 1;
```

```
    printf("The answer is %d\n", universe * 6 / 1);
```

```
}
```



```
#include <stdio.h>
```

```
static int calc(int a, int b, int c)
```

```
{
```

```
    return a * b / c;
```

```
}
```

```
int universe = 7;
```

```
static int life(void) { return 6; }
```

```
int everything(void) { return 1; }
```

```
int main(void)
```

```
{
```

```
int a = universe * 6 / 1;
```

```
printf("The answer is %d\n", universe * 6 / 1);
```

```
}
```



```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = universe * 6 / 1;
    printf("The answer is %d\n", universe * 6 / 1);
}
```

But it will still print...

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = universe * 6 / 1;
    printf("The answer is %d\n", universe * 6 / 1);
}
```

But it will still print...

```
The answer is 42
$ echo $?
0
```

```
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = universe * 6 / 1;
    printf("The answer is %d\n", universe * 6 / 1);
}
```

But it will still print...

```
The answer is 42
$ echo $?
0
```

The key take away from this session is that things like execution stack and calling conventions are not dictated by the standard. The evaluation order of expressions and arguments is mostly unspecified. And the optimizer might rearrange the execution of the code significantly. In C, nearly everything can happen, and will happen, internally as long as the external behavior is satisfied.

We just had a glimpse of what ***might*** happen when code is executed.

Scary stuff? Not really, but with only a shallow understanding of the language it is easy to make big mistakes.

The goal of this course is to give you a deep understanding of C, by starting from scratch and relearn the language in a systematic way.

First, let's start to establish a vocabulary. To do that, we need an even more contrived example program...

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

```
$ cc foo.c
$ ./a.out
Hello everyone
$ echo $?
0
```

Why do we  
need this?

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

```
$ cc foo.c
$ ./a.out
Hello everyone
$ echo $?
0
```



Why do we  
need this?

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

and what does  
this mean?

```
$ cc foo.c
$ ./a.out
Hello everyone
$ echo $?
0
```



Why do we  
need this?

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

and what does  
this mean?

how is a for loop  
*really* working?

```
$ cc foo.c
$ ./a.out
Hello everyone
$ echo $?
0
```

Why do we need this?

```
int printf(const char * restrict, ...);  
void exit(int);  
#define EXIT_FAILURE 1
```

and what does this mean?

```
static void say_hello(const char * who)  
{  
    printf("Hello %s\n", who);  
}
```

how many arguments does this function accept?

```
// compute the answer  
int the_answer()  
{  
    register int a;  
    int b = 6; /* mystic base value */  
    a = 0;  
    for (int i=0; i<7; i++)  
        a += b;  
    return a;  
}
```

how is a for loop really working?

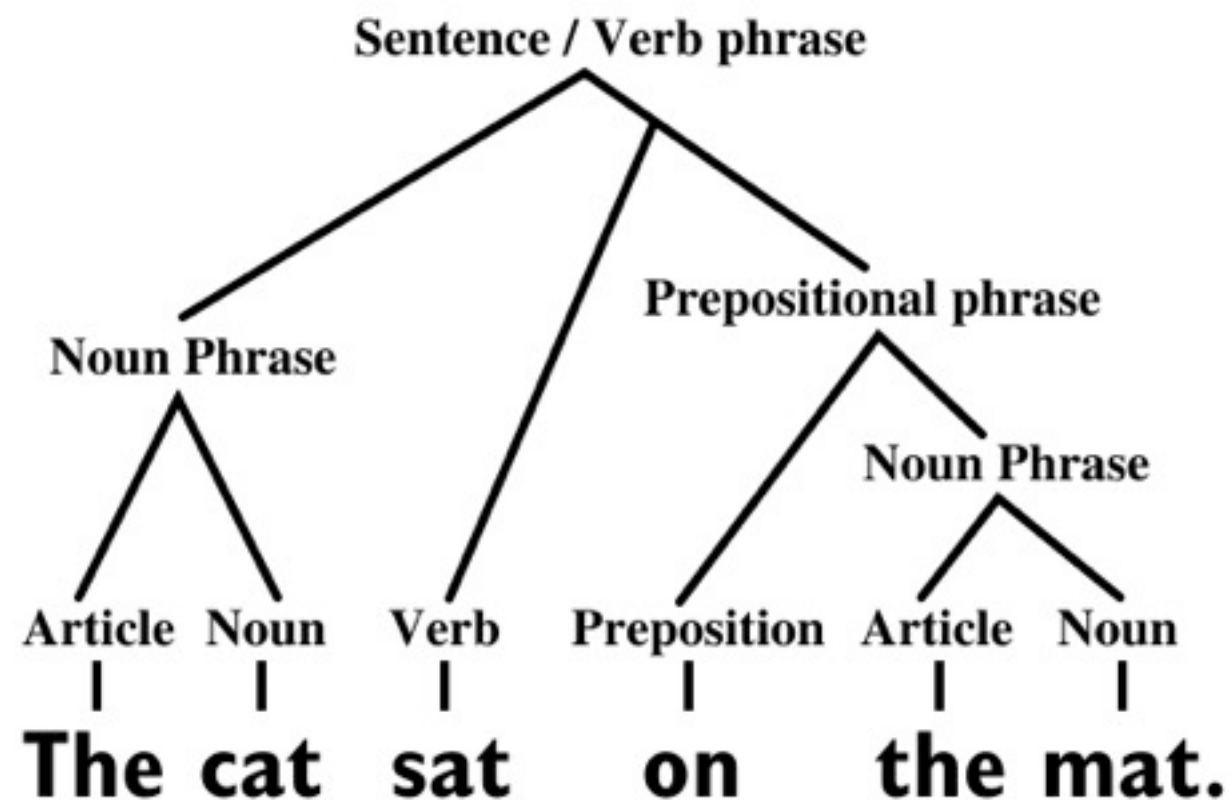
```
const int life_universe_everything = 42;
```

```
int main(void)  
{  
    say_hello("everyone");  
    int a = the_answer();  
    if (a != life_universe_everything)  
        exit(EXIT_FAILURE);  
}
```

```
$ cc foo.c  
$ ./a.out  
Hello everyone  
$ echo $?  
0
```

To get a deep understanding of any language you need to be able to ‘break’ it down and analyse it, at least you need to recognize the words used when experts are discussing among themselves

Basic constituent structure analysis of a sentence:



```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

keyword

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

keyword

keyword

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    keyword → for (int i=0; i<7; i++)
                a += b;
    keyword → return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    keyword → int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```



```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```



function prototype  
(declaration) 

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

function prototype  
(declaration) 

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

function prototype  
(declaration)

function definition

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

linkage  
specification



```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

linkage  
specification



```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

linkage  
specification

identifier with  
internal linkage

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```



linkage  
specification

identifier with  
internal linkage

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```



linkage  
specification

identifier with  
internal linkage

identifier with  
external linkage

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

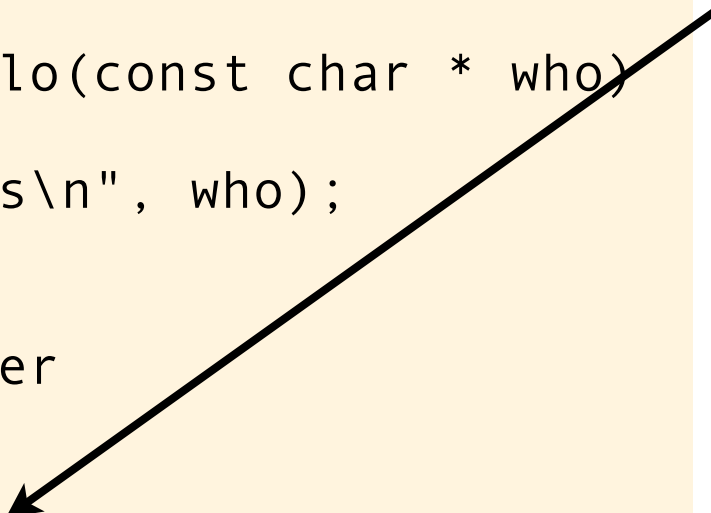
static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

declaration without  
initialization



```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

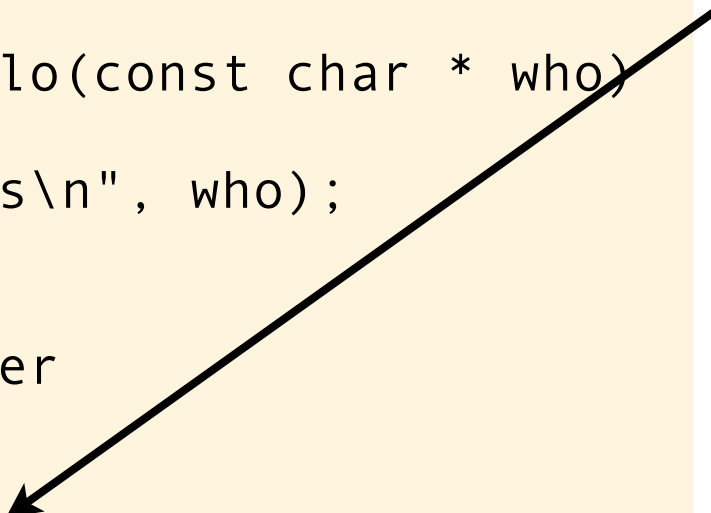
static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

declaration without  
initialization



```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

declaration without  
initialization

declaration with  
initialization

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

declaration without  
initialization

declaration with  
initialization

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

declaration without  
initialization

declaration with  
initialization

expression



```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

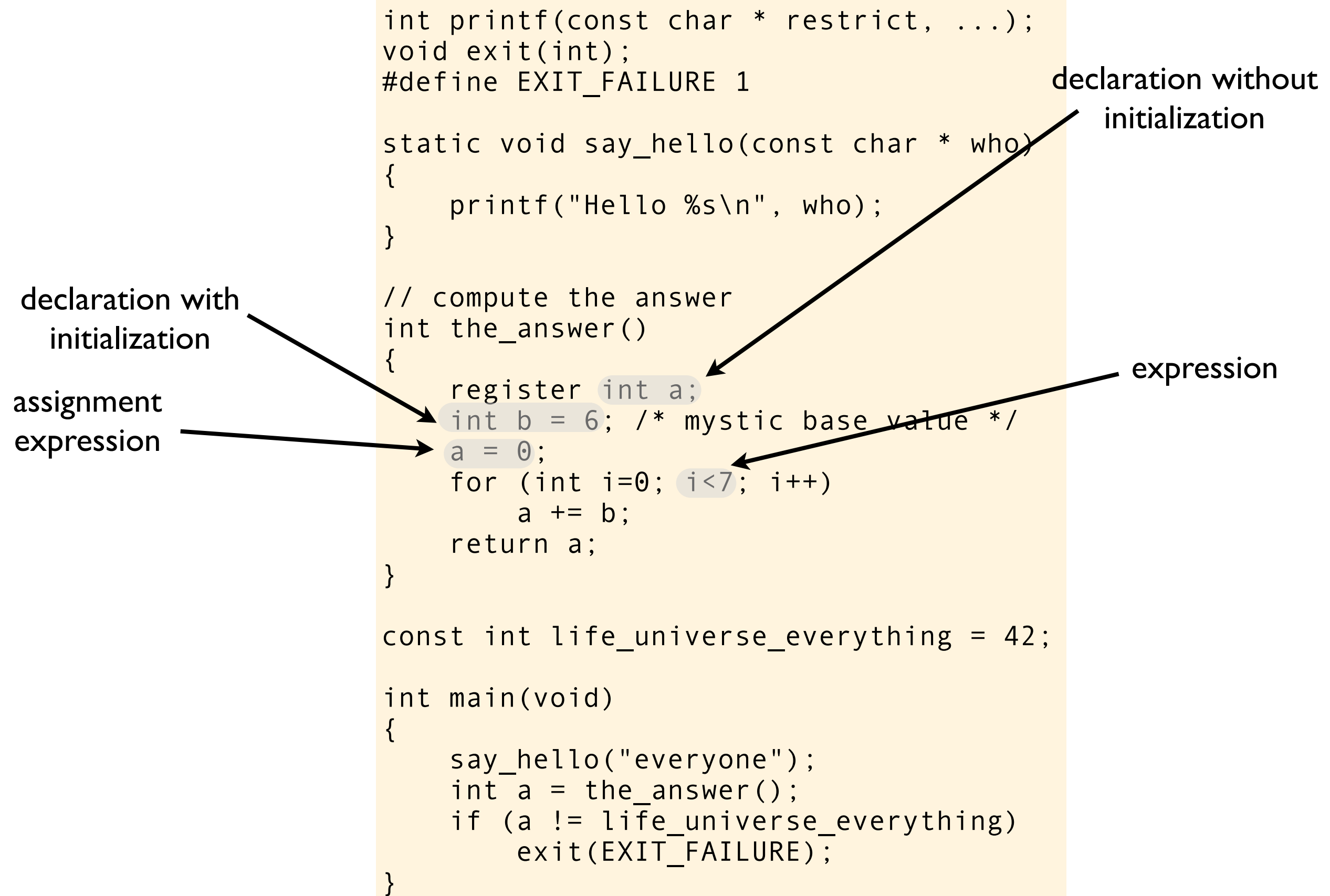
int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

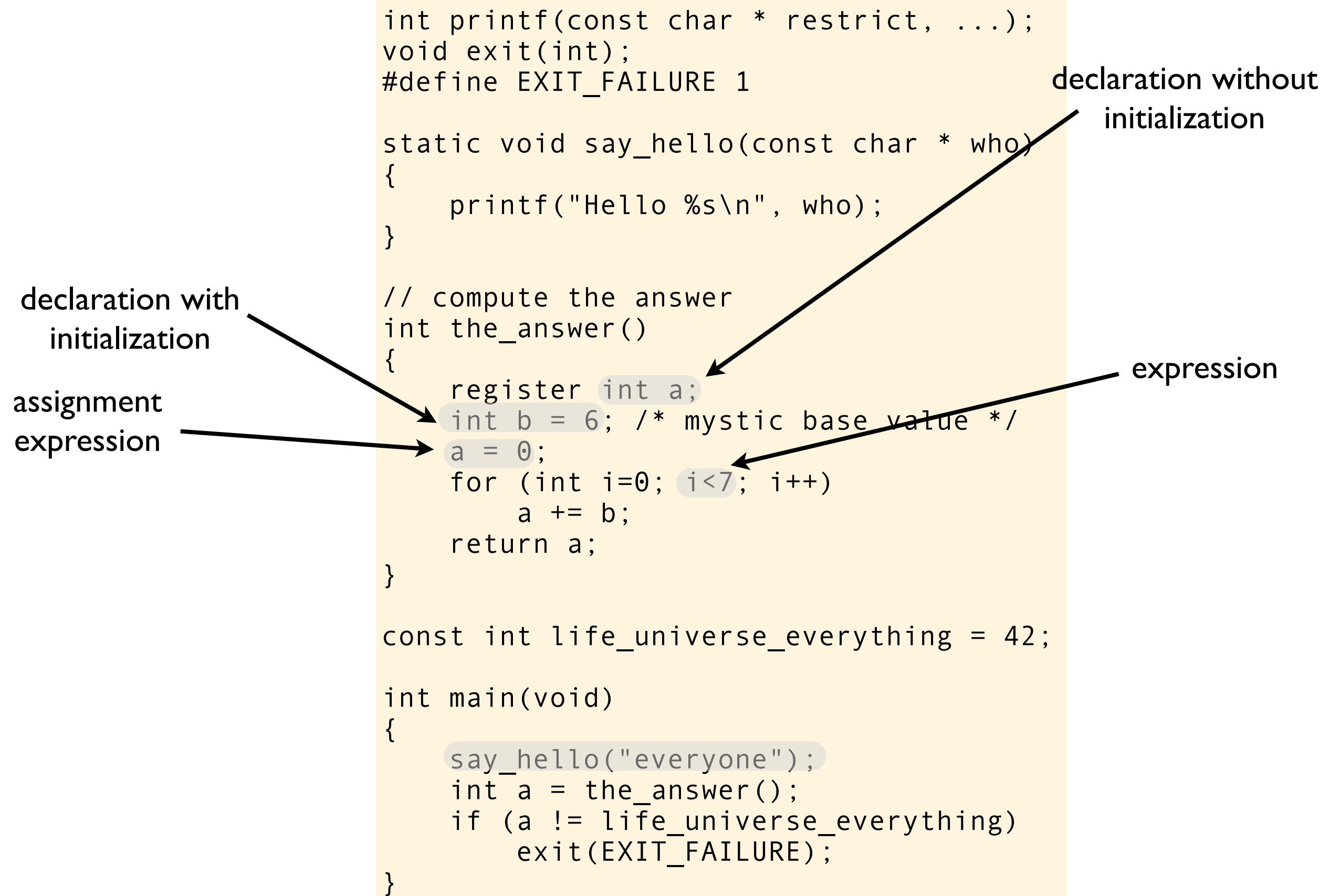
declaration without  
initialization

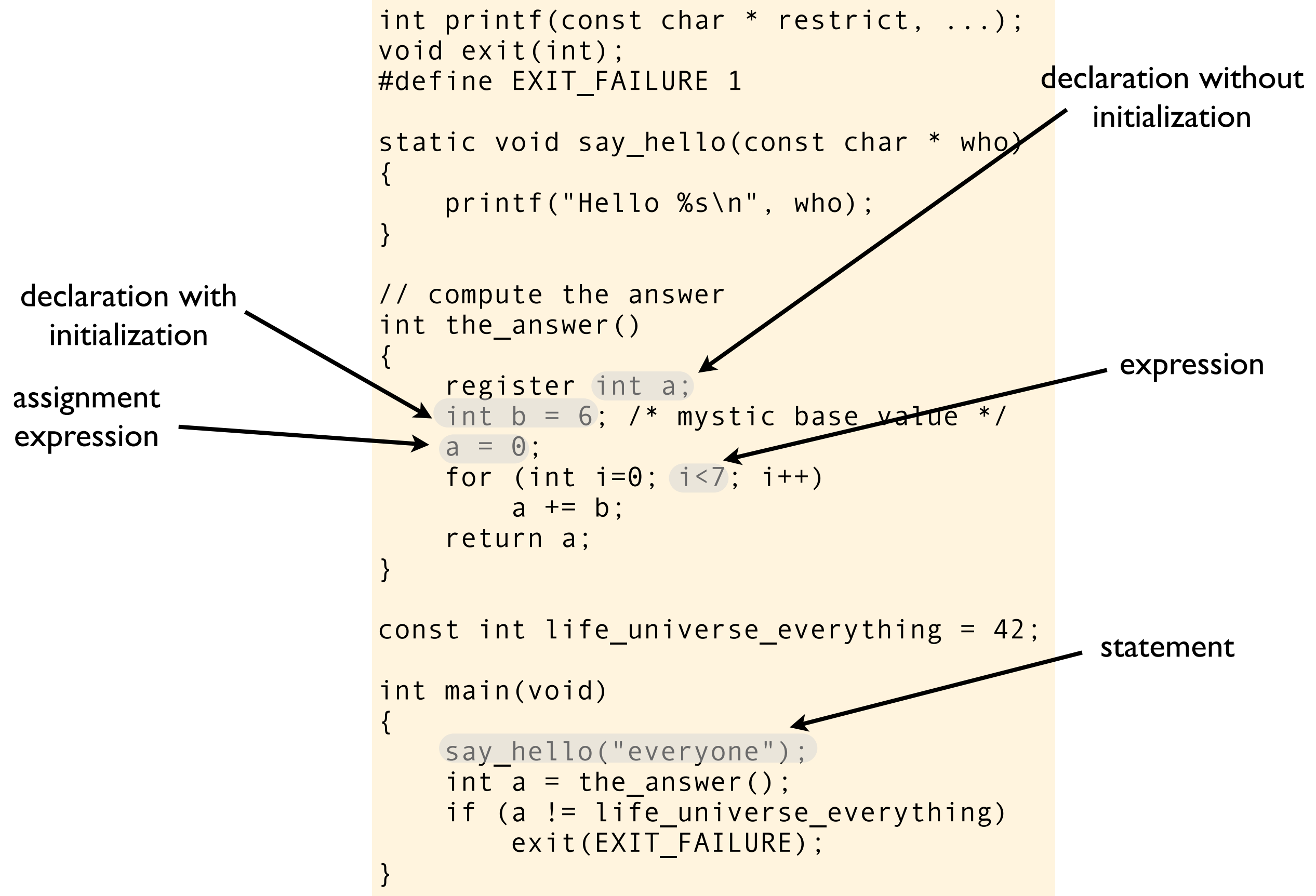
declaration with  
initialization

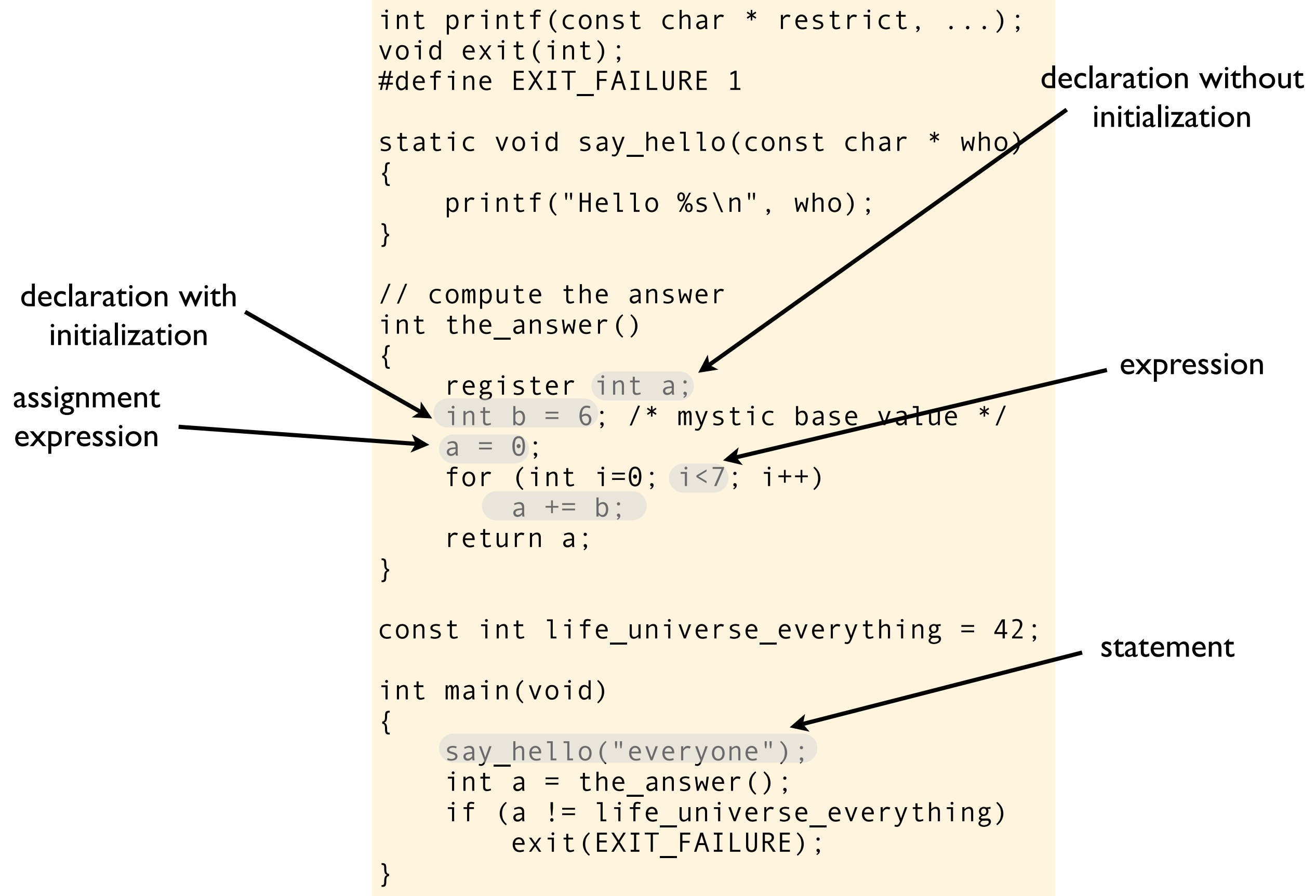
expression

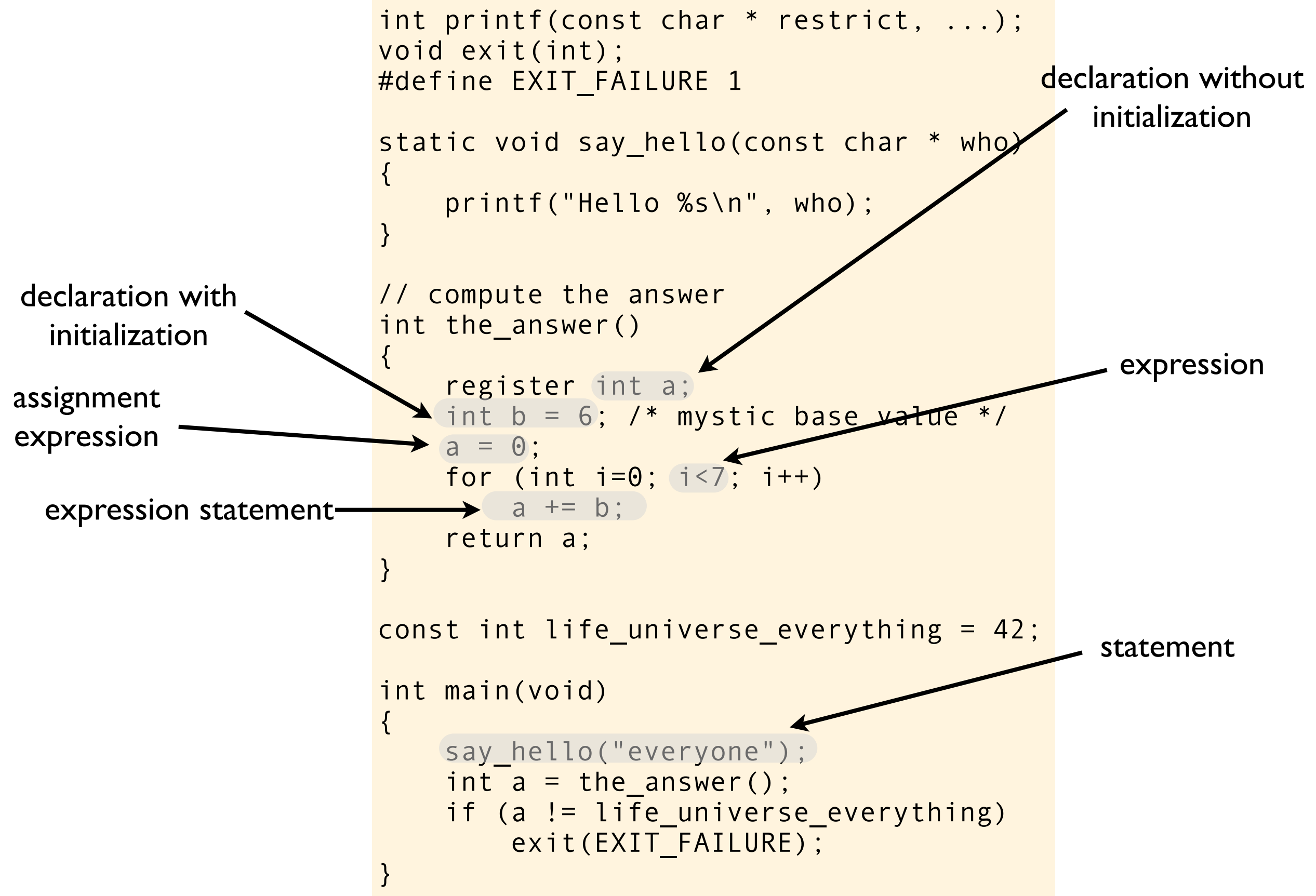












```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

storage class specifier

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```



storage class specifier

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```



```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

storage class specifier

type qualifier

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

storage class specifier

type qualifier

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1
```

```
static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}
```

```
// compute the answer
int the_answer()
{
```

storage class specifier

```
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}
```

type qualifier

```
    const int life_universe_everything = 42;
```

type specifier

```
int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

```
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1

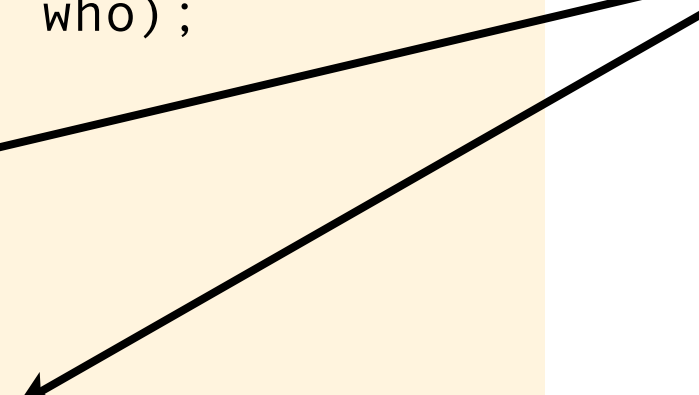
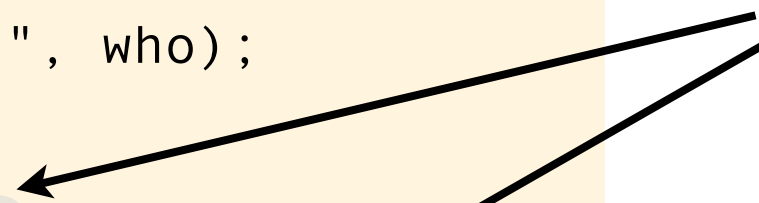
static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// compute the answer
int the_answer()
{
    register int a;
    int b = 6; /* mystic base value */
    a = 0;
    for (int i=0; i<7; i++)
        a += b;
    return a;
}

const int life_universe_everything = 42;

int main(void)
{
    say_hello("everyone");
    int a = the_answer();
    if (a != life_universe_everything)
        exit(EXIT_FAILURE);
}
```

comment



pre-processor directive

```
int printf(const char * restrict, ...);  
void exit(int);  
#define EXIT_FAILURE 1
```

```
static void say_hello(const char * who)  
{  
    printf("Hello %s\n", who);  
}
```

```
// compute the answer
```

```
int the_answer()  
{
```

```
    register int a;
```

```
    int b = 6; /* mystic base value */
```

```
    a = 0;
```

```
    for (int i=0; i<7; i++)
```

```
        a += b;
```

```
    return a;
```

```
}
```

```
const int life_universe_everything = 42;
```

```
int main(void)  
{
```

```
    say_hello("everyone");
```

```
    int a = the_answer();
```

```
    if (a != life_universe_everything)
```

```
        exit(EXIT_FAILURE);
```

```
}
```

comment

pre-processor directive

```
int printf(const char * restrict, ...);  
void exit(int);  
#define EXIT_FAILURE 1
```

comment

```
static void say_hello(const char * who)  
{  
    printf("Hello %s\n", who);  
}
```

```
// compute the answer
```

```
int the_answer()  
{
```

```
    register int a;
```

```
    int b = 6; /* mystic base value */
```

```
    a = 0;
```

```
    for (int i=0; i<7; i++)
```

```
        a += b;
```

```
    return a;
```

```
}
```

```
const int life_universe_everything = 42;
```

```
int main(void)  
{
```

```
    say_hello("everyone");
```

```
    int a = the_answer();
```

```
    if (a != life_universe_everything)
```

```
        exit(EXIT_FAILURE);
```

```
}
```

macro name



function prototype

pre-processor directive

linkage  
specification

declaration without  
initialization

function definition

comment

internal linkage

external linkage

expression

storage class specifier

declaration with  
initialization

expression statement

keyword

assignment  
expression

type qualifier

statement

type specifier

macro name

```
int printf(const char * restrict, ...);  
void exit(int);  
#define EXIT_FAILURE 1  
  
static void say_hello(const char * who)  
{  
    printf("Hello %s\n", who);  
}  
  
// compute the answer  
int the_answer()  
{  
    register int a;  
    int b = 6; /* mystic base value */  
    a = 0;  
    for (int i=0; i<7; i++)  
        a += b;  
    return a;  
}  
  
const int life_universe_everything = 42;  
  
int main(void)  
{  
    say_hello("everyone");  
    int a = the_answer();  
    if (a != life_universe_everything)  
        exit(EXIT_FAILURE);  
}
```



A glimpse into tools often  
used when developing C

# Deep thought, Part I

dt.c

```
#include "dt.h"

int dt_base_value;
#define MULTIPLIER 7
static int dt_answer;

static void run_computer(void)
{
    dt_answer = dt_base_value * MULTIPLIER;
}

int dt_get_answer(void)
{
    run_computer();
    return dt_answer;
}
```

dt.h

```
extern int dt_base_value;
int dt_get_answer(void);
```

theanswer.c

```
#include "dt.h"
#include <stdio.h>

int main(void)
{
    dt_base_value = 6;
    int answer = dt_get_answer();
    printf("The answer is %d\n",
           answer);
}
```

# Deep thought, Part I

dt.c

```
#include "dt.h"

int dt_base_value;
#define MULTIPLIER 7
static int dt_answer;

static void run_computer(void)
{
    dt_answer = dt_base_value * MULTIPLIER;
}

int dt_get_answer(void)
{
    run_computer();
    return dt_answer;
}
```

dt.h

```
extern int dt_base_value;
int dt_get_answer(void);
```

theanswer.c

```
#include "dt.h"
#include <stdio.h>

int main(void)
{
    dt_base_value = 6;
    int answer = dt_get_answer();
    printf("The answer is %d\n",
           answer);
}
```

```
$ cc -c dt.c
```

# Deep thought, Part I

dt.c

```
#include "dt.h"

int dt_base_value;
#define MULTIPLIER 7
static int dt_answer;

static void run_computer(void)
{
    dt_answer = dt_base_value * MULTIPLIER;
}

int dt_get_answer(void)
{
    run_computer();
    return dt_answer;
}
```

dt.h

```
extern int dt_base_value;
int dt_get_answer(void);
```

theanswer.c

```
#include "dt.h"
#include <stdio.h>

int main(void)
{
    dt_base_value = 6;
    int answer = dt_get_answer();
    printf("The answer is %d\n",
           answer);
}
```

```
$ cc -c dt.c
$ cc -c theanswer.c
```

# Deep thought, Part I

dt.c

```
#include "dt.h"

int dt_base_value;
#define MULTIPLIER 7
static int dt_answer;

static void run_computer(void)
{
    dt_answer = dt_base_value * MULTIPLIER;
}

int dt_get_answer(void)
{
    run_computer();
    return dt_answer;
}
```

dt.h

```
extern int dt_base_value;
int dt_get_answer(void);
```

theanswer.c

```
#include "dt.h"
#include <stdio.h>

int main(void)
{
    dt_base_value = 6;
    int answer = dt_get_answer();
    printf("The answer is %d\n",
           answer);
}
```

```
$ cc -c dt.c
$ cc -c theanswer.c
$ cc -o theanswer theanswer.o dt.o
```

# Deep thought, Part I

dt.c

```
#include "dt.h"

int dt_base_value;
#define MULTIPLIER 7
static int dt_answer;

static void run_computer(void)
{
    dt_answer = dt_base_value * MULTIPLIER;
}

int dt_get_answer(void)
{
    run_computer();
    return dt_answer;
}
```

dt.h

```
extern int dt_base_value;
int dt_get_answer(void);
```

theanswer.c

```
#include "dt.h"
#include <stdio.h>

int main(void)
{
    dt_base_value = 6;
    int answer = dt_get_answer();
    printf("The answer is %d\n",
           answer);
}
```

```
$ cc -c dt.c
$ cc -c theanswer.c
$ cc -o theanswer theanswer.o dt.o
$ ./theanswer
```

# Deep thought, Part I

dt.c

```
#include "dt.h"

int dt_base_value;
#define MULTIPLIER 7
static int dt_answer;

static void run_computer(void)
{
    dt_answer = dt_base_value * MULTIPLIER;
}

int dt_get_answer(void)
{
    run_computer();
    return dt_answer;
}
```

dt.h

```
extern int dt_base_value;
int dt_get_answer(void);
```

theanswer.c

```
#include "dt.h"
#include <stdio.h>

int main(void)
{
    dt_base_value = 6;
    int answer = dt_get_answer();
    printf("The answer is %d\n",
           answer);
}
```

```
$ cc -c dt.c
$ cc -c theanswer.c
$ cc -o theanswer theanswer.o dt.o
$ ./theanswer
The answer is 42
```



# Deep thought, Part I

dt.c

```
#include "dt.h"

int dt_base_value;
#define MULTIPLIER 7
static int dt_answer;

static void run_computer(void)
{
    dt_answer = dt_base_value * MULTIPLIER;
}

int dt_get_answer(void)
{
    run_computer();
    return dt_answer;
}
```

dt.h

```
extern int dt_base_value;
int dt_get_answer(void);
```

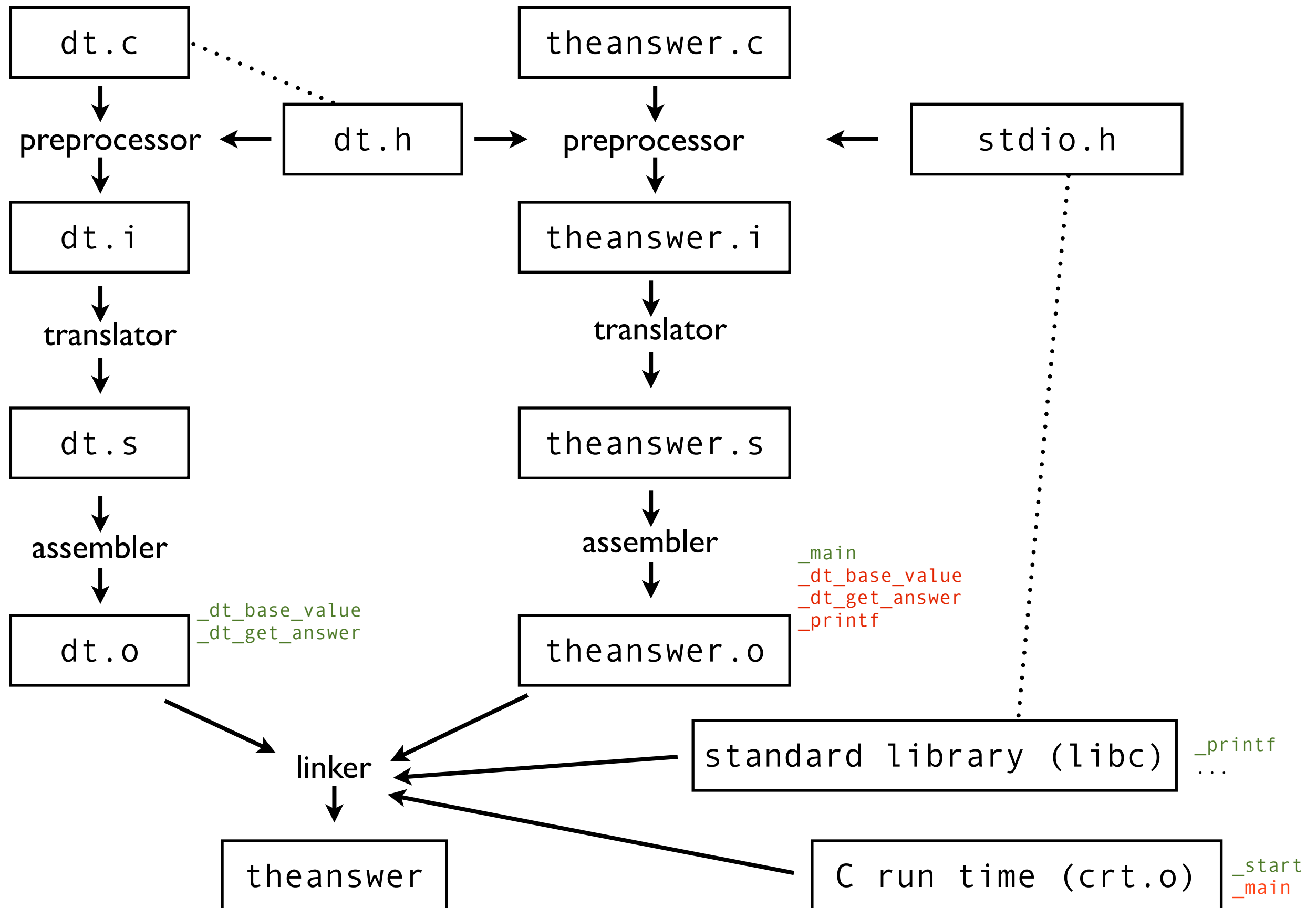
theanswer.c

```
#include "dt.h"
#include <stdio.h>

int main(void)
{
    dt_base_value = 6;
    int answer = dt_get_answer();
    printf("The answer is %d\n",
           answer);
}
```

```
$ cc -c dt.c
$ cc -c theanswer.c
$ cc -o theanswer theanswer.o dt.o
$ ./theanswer
The answer is 42
$
```





# Deep thought, Part 2

```
dt.c
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

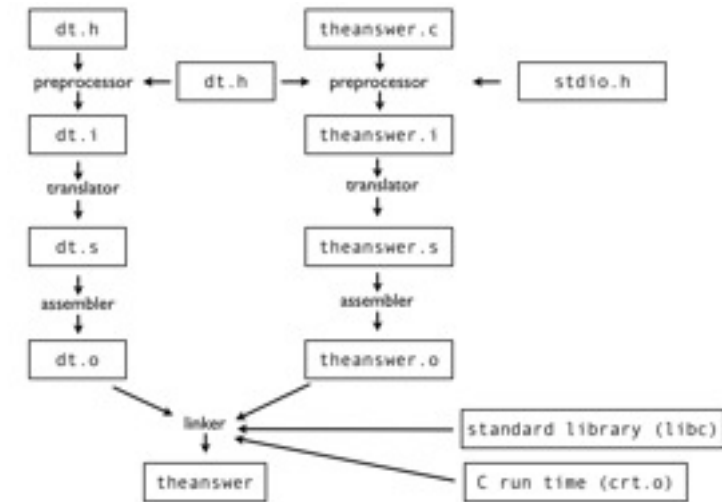
void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}

dt.h
void dt_init(void);
int dt_compute_answer(void);

theanswer.c
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
        answer);
}
```



# Deep thought, Part 2

```
dt.c
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

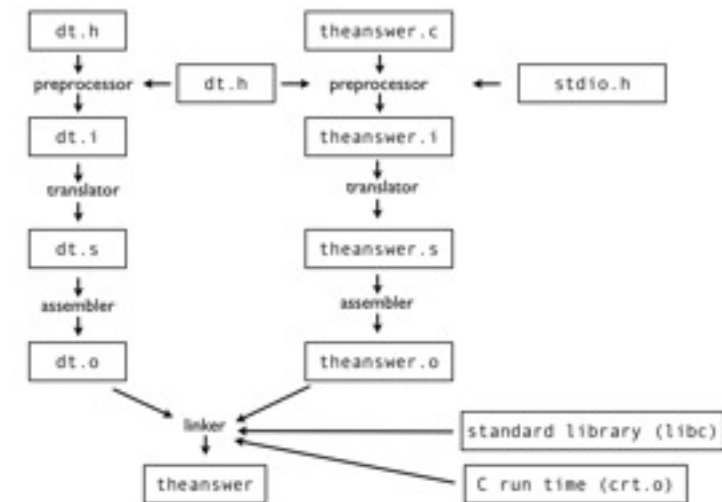
void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}

dt.h
void dt_init(void);
int dt_compute_answer(void);

theanswer.c
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
        answer);
}
```



```
$ cc -E dt.c >dt.i
```

# Deep thought, Part 2

```
dt.c
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

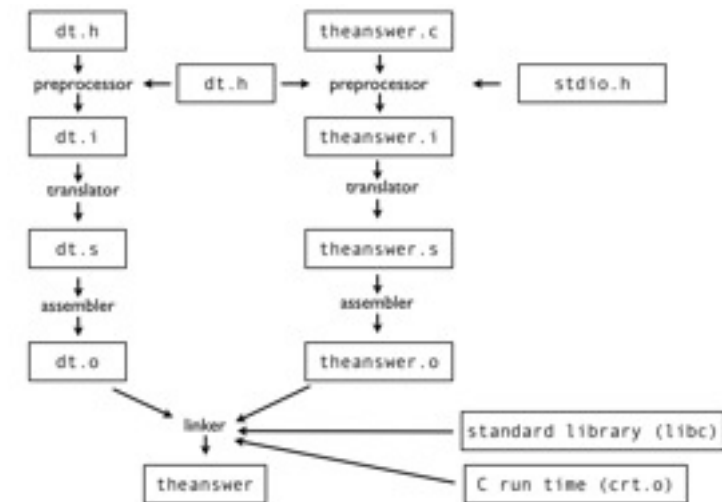
void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}

dt.h
void dt_init(void);
int dt_compute_answer(void);

theanswer.c
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
        answer);
}
```



```
$ cc -E dt.c >dt.i
$ cat dt.i
```

# Deep thought, Part 2

```
dt.c
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

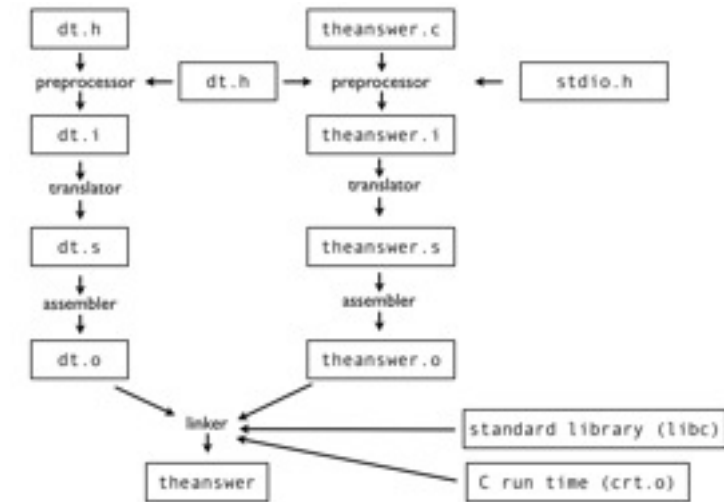
void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}

dt.h
void dt_init(void);
int dt_compute_answer(void);

theanswer.c
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
          answer);
}
```



```
$ cc -E dt.c >dt.i
$ cat dt.i
$ cc -S dt.i
```

# Deep thought, Part 2

```
dt.c
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

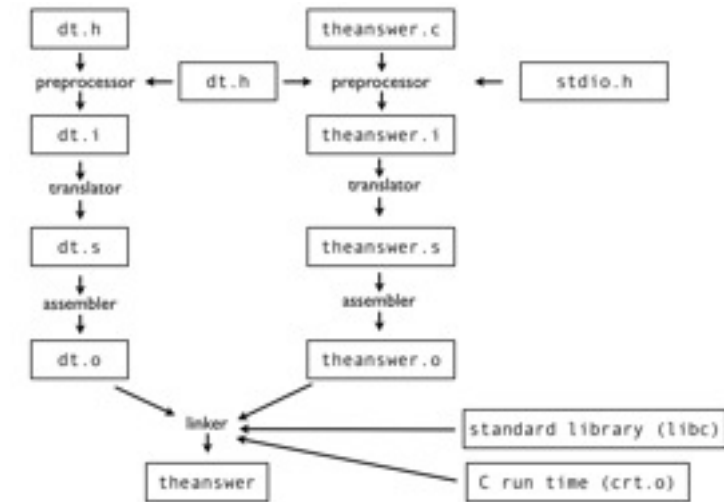
void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}

dt.h
void dt_init(void);
int dt_compute_answer(void);

theanswer.c
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
        answer);
}
```



```
$ cc -E dt.c >dt.i
$ cat dt.i
$ cc -S dt.i
$ cat dt.s
```

# Deep thought, Part 2

```
dt.c
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

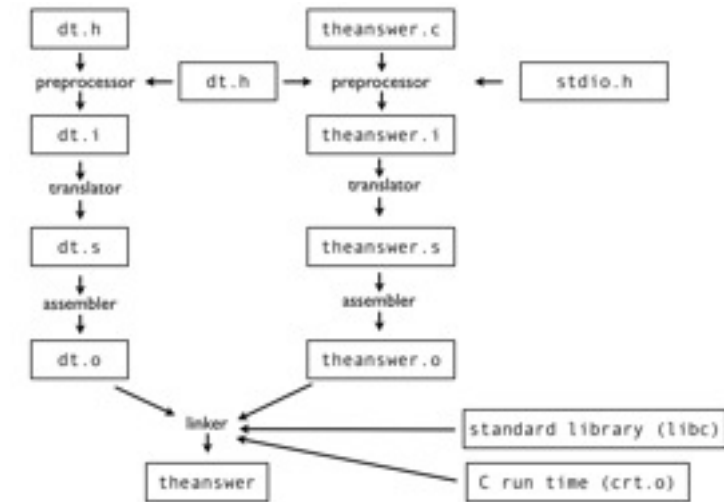
void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}

dt.h
void dt_init(void);
int dt_compute_answer(void);

theanswer.c
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
        answer);
}
```



```
$ cc -E dt.c >dt.i
$ cat dt.i
$ cc -S dt.i
$ cat dt.s
$ cc -c dt.s
```



# Deep thought, Part 2

```
dt.c
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

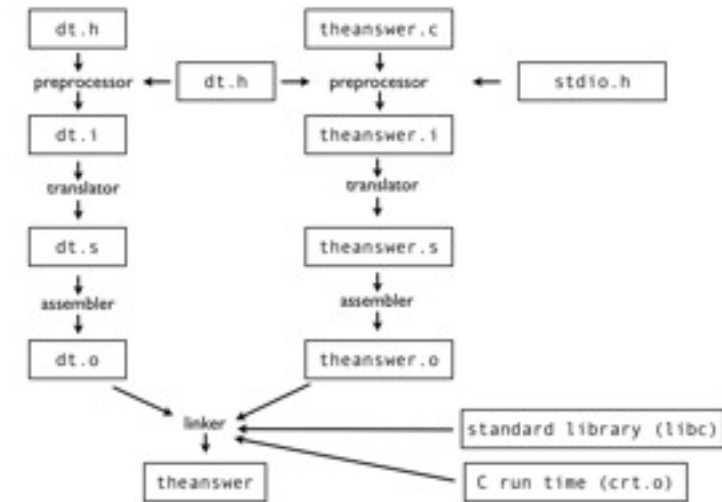
void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}

dt.h
void dt_init(void);
int dt_compute_answer(void);

theanswer.c
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
          answer);
}
```



```
$ cc -E dt.c >dt.i
$ cat dt.i
$ cc -S dt.i
$ cat dt.s
$ cc -c dt.s
$ nm dt.o
```



# Deep thought, Part 2

```
dt.c
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

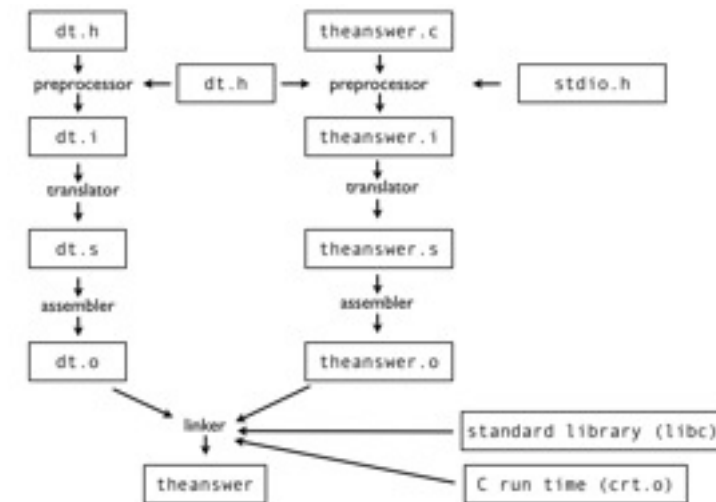
void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}

dt.h
void dt_init(void);
int dt_compute_answer(void);

theanswer.c
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
        answer);
}
```



```
$ cc -E dt.c >dt.i
$ cat dt.i
$ cc -S dt.i
$ cat dt.s
$ cc -c dt.s
$ nm dt.o

$ cc -c -save-temps theanswer.c
```

# Deep thought, Part 2

```
dt.c
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

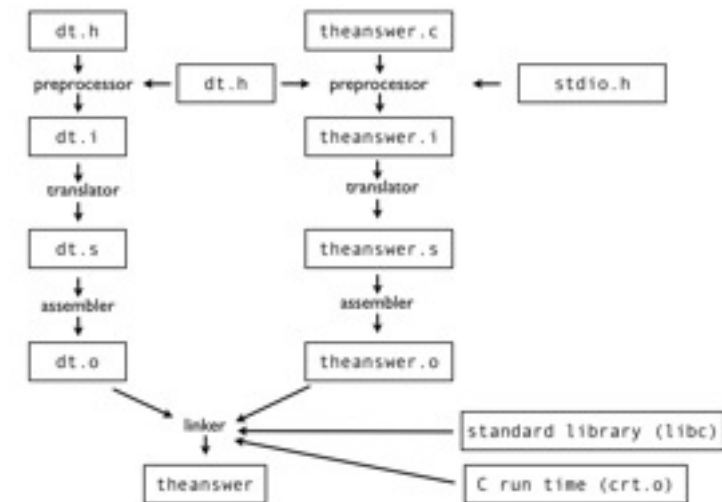
void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}

dt.h
void dt_init(void);
int dt_compute_answer(void);

theanswer.c
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
        answer);
}
```



```
$ cc -E dt.c >dt.i
$ cat dt.i
$ cc -S dt.i
$ cat dt.s
$ cc -c dt.s
$ nm dt.o
```

```
$ cc -c -save-temps theanswer.c
$ ls theanswer.*
```

# Deep thought, Part 2

```
dt.c
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

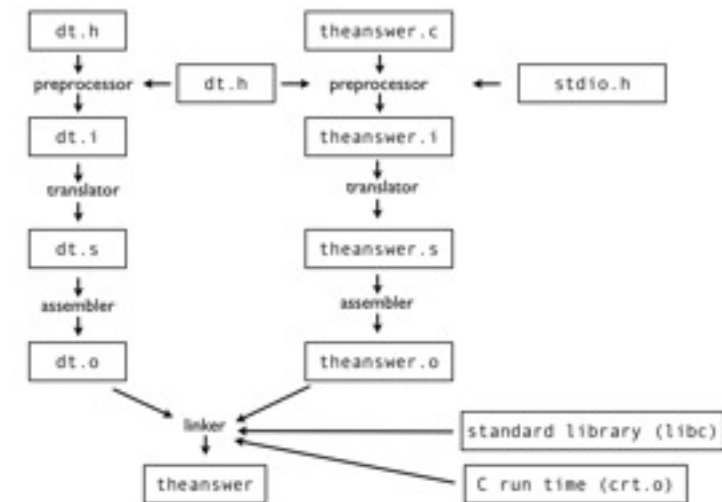
void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}

dt.h
void dt_init(void);
int dt_compute_answer(void);

theanswer.c
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
        answer);
}
```



```
$ cc -E dt.c >dt.i
$ cat dt.i
$ cc -S dt.i
$ cat dt.s
$ cc -c dt.s
$ nm dt.o
```

```
$ cc -c -save-temps theanswer.c
$ ls theanswer.*
$ nm theanswer.o
```

# Deep thought, Part 2

```
dt.c
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

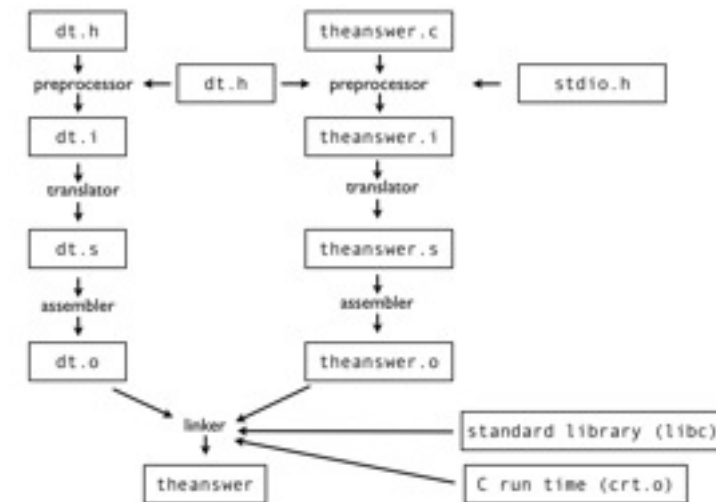
void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}

dt.h
void dt_init(void);
int dt_compute_answer(void);

theanswer.c
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
        answer);
}
```



```
$ cc -E dt.c >dt.i
$ cat dt.i
$ cc -S dt.i
$ cat dt.s
$ cc -c dt.s
$ nm dt.o
```

```
$ cc -c -save-temps theanswer.c
$ ls theanswer.*
$ nm theanswer.o
```

```
$ ld -lc -o theanswer dt.o theanswer.o /usr/lib/crt1.o
```

# Deep thought, Part 2

```
dt.c
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

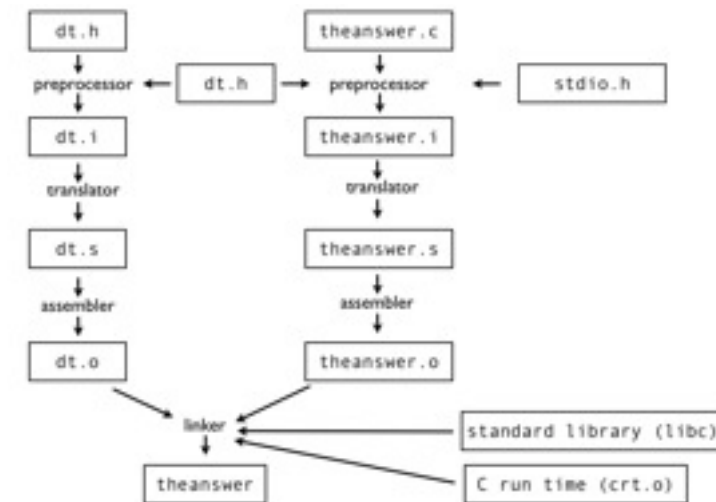
void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}

dt.h
void dt_init(void);
int dt_compute_answer(void);

theanswer.c
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
        answer);
}
```



```
$ cc -E dt.c >dt.i
$ cat dt.i
$ cc -S dt.i
$ cat dt.s
$ cc -c dt.s
$ nm dt.o
```

```
$ cc -c -save-temps theanswer.c
$ ls theanswer.*
$ nm theanswer.o
```

```
$ ld -lc -o theanswer dt.o theanswer.o /usr/lib/crt1.o
$ ./theanswer
```

# Deep thought, Part 2

```
dt.c
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

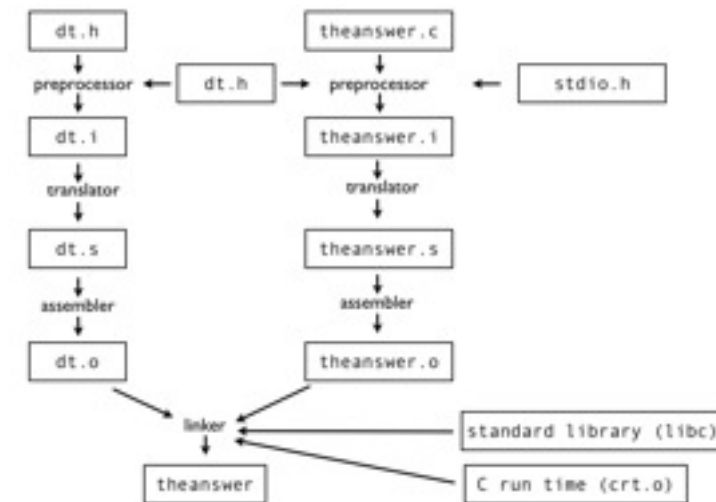
void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}

dt.h
void dt_init(void);
int dt_compute_answer(void);

theanswer.c
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
        answer);
}
```



```
$ cc -E dt.c >dt.i
$ cat dt.i
$ cc -S dt.i
$ cat dt.s
$ cc -c dt.s
$ nm dt.o
```

```
$ cc -c -save-temps theanswer.c
$ ls theanswer.*
$ nm theanswer.o
```

```
$ ld -lc -o theanswer dt.o theanswer.o /usr/lib/crt1.o
$ ./theanswer
The answer is 42
```

# Deep thought, Part 3

dt.c

```
#include "dt.h"

int dt_base_value;
static int dt_answer;

static void run_computer(int multiplier)
{
    dt_answer = dt_base_value * multiplier;
}

void dt_init(void)
{
    dt_base_value = 6;
}

int dt_compute_answer(void)
{
    run_computer(7);
    return dt_answer;
}
```

dt.h

```
void dt_init(void);
int dt_compute_answer(void);
```

theanswer.c

```
#include <stdio.h>
#include "dt.h"

int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
           answer);
}
```



# Deep thought, Part 3

| dt.c                                                                                                                                                                                                                                                                                      | dt.h                                                                                                                                                                                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#include "dt.h"  int dt_base_value; static int dt_answer;  static void run_computer(int multiplier) {     dt_answer = dt_base_value * multiplier; }  void dt_init(void) {     dt_base_value = 6; }  int dt_compute_answer(void) {     run_computer(7);     return dt_answer; }</pre> | <pre>void dt_init(void); int dt_compute_answer(void);</pre>                                                                                                                                 |
|                                                                                                                                                                                                                                                                                           | <pre>theanswer.c  #include &lt;stdio.h&gt; #include "dt.h"  int main(void) {     dt_init();     int answer = dt_compute_answer();     printf("The answer is %d\n",         answer); }</pre> |

```
$ cc -g -o theanswer dt.c theanswer.c
```



# Deep thought, Part 3

| dt.c                                                                                                                                                                                                                                                                                      | dt.h                                                                                                                                                                                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#include "dt.h"  int dt_base_value; static int dt_answer;  static void run_computer(int multiplier) {     dt_answer = dt_base_value * multiplier; }  void dt_init(void) {     dt_base_value = 6; }  int dt_compute_answer(void) {     run_computer(7);     return dt_answer; }</pre> | <pre>void dt_init(void); int dt_compute_answer(void);</pre>                                                                                                                                 |
|                                                                                                                                                                                                                                                                                           | <pre>theanswer.c  #include &lt;stdio.h&gt; #include "dt.h"  int main(void) {     dt_init();     int answer = dt_compute_answer();     printf("The answer is %d\n",         answer); }</pre> |

```
$ cc -g -o theanswer dt.c theanswer.c
$ gdb theanswer
```

# Deep thought, Part 3

| dt.c                                                                                                                                                                                                                                                                                      | dt.h                                                                                                                                                                                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#include "dt.h"  int dt_base_value; static int dt_answer;  static void run_computer(int multiplier) {     dt_answer = dt_base_value * multiplier; }  void dt_init(void) {     dt_base_value = 6; }  int dt_compute_answer(void) {     run_computer(7);     return dt_answer; }</pre> | <pre>void dt_init(void); int dt_compute_answer(void);</pre>                                                                                                                                 |
|                                                                                                                                                                                                                                                                                           | <pre>theanswer.c  #include &lt;stdio.h&gt; #include "dt.h"  int main(void) {     dt_init();     int answer = dt_compute_answer();     printf("The answer is %d\n",         answer); }</pre> |

```
$ cc -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
```

# Deep thought, Part 3

| dt.c                                                                                                                                                                                                                                                                                      | dt.h                                                                                                                                                                                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#include "dt.h"  int dt_base_value; static int dt_answer;  static void run_computer(int multiplier) {     dt_answer = dt_base_value * multiplier; }  void dt_init(void) {     dt_base_value = 6; }  int dt_compute_answer(void) {     run_computer(7);     return dt_answer; }</pre> | <pre>void dt_init(void); int dt_compute_answer(void);</pre>                                                                                                                                 |
|                                                                                                                                                                                                                                                                                           | <pre>theanswer.c  #include &lt;stdio.h&gt; #include "dt.h"  int main(void) {     dt_init();     int answer = dt_compute_answer();     printf("The answer is %d\n",         answer); }</pre> |

```
$ cc -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break main
```

# Deep thought, Part 3

| dt.c                                                                                                                                                                                                                                                                                      | dt.h                                                                                                                                                                                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#include "dt.h"  int dt_base_value; static int dt_answer;  static void run_computer(int multiplier) {     dt_answer = dt_base_value * multiplier; }  void dt_init(void) {     dt_base_value = 6; }  int dt_compute_answer(void) {     run_computer(7);     return dt_answer; }</pre> | <pre>void dt_init(void); int dt_compute_answer(void);</pre>                                                                                                                                 |
|                                                                                                                                                                                                                                                                                           | <pre>theanswer.c  #include &lt;stdio.h&gt; #include "dt.h"  int main(void) {     dt_init();     int answer = dt_compute_answer();     printf("The answer is %d\n",         answer); }</pre> |

```
$ cc -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break main
(gdb) run
```

# Deep thought, Part 3

| dt.c                                                                                                                                                                                                                                                                                      | dt.h                                                                                                                                                                                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#include "dt.h"  int dt_base_value; static int dt_answer;  static void run_computer(int multiplier) {     dt_answer = dt_base_value * multiplier; }  void dt_init(void) {     dt_base_value = 6; }  int dt_compute_answer(void) {     run_computer(7);     return dt_answer; }</pre> | <pre>void dt_init(void); int dt_compute_answer(void);</pre>                                                                                                                                 |
|                                                                                                                                                                                                                                                                                           | <pre>theanswer.c  #include &lt;stdio.h&gt; #include "dt.h"  int main(void) {     dt_init();     int answer = dt_compute_answer();     printf("The answer is %d\n",         answer); }</pre> |

```
$ cc -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break main
(gdb) run
(gdb) next
```

# Deep thought, Part 3

| dt.c                                                                                                                                                                                                                                                                                      | dt.h                                                                                                                                                                                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#include "dt.h"  int dt_base_value; static int dt_answer;  static void run_computer(int multiplier) {     dt_answer = dt_base_value * multiplier; }  void dt_init(void) {     dt_base_value = 6; }  int dt_compute_answer(void) {     run_computer(7);     return dt_answer; }</pre> | <pre>void dt_init(void); int dt_compute_answer(void);</pre>                                                                                                                                 |
|                                                                                                                                                                                                                                                                                           | <pre>theanswer.c  #include &lt;stdio.h&gt; #include "dt.h"  int main(void) {     dt_init();     int answer = dt_compute_answer();     printf("The answer is %d\n",         answer); }</pre> |

```
$ cc -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break main
(gdb) run
(gdb) next
(gdb) step
```

# Deep thought, Part 3

| dt.c                                                                                                                                                                                                                                                                                      | dt.h                                                                                                                                                                           |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#include "dt.h"  int dt_base_value; static int dt_answer;  static void run_computer(int multiplier) {     dt_answer = dt_base_value * multiplier; }  void dt_init(void) {     dt_base_value = 6; }  int dt_compute_answer(void) {     run_computer(7);     return dt_answer; }</pre> | <pre>void dt_init(void); int dt_compute_answer(void);</pre>                                                                                                                    |
|                                                                                                                                                                                                                                                                                           | theanswer.c                                                                                                                                                                    |
|                                                                                                                                                                                                                                                                                           | <pre>#include &lt;stdio.h&gt; #include "dt.h"  int main(void) {     dt_init();     int answer = dt_compute_answer();     printf("The answer is %d\n",         answer); }</pre> |

```
$ cc -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break main
(gdb) run
(gdb) next
(gdb) step
(gdb) cont
```



# Deep thought, Part 3

| dt.c                                                                                                                                                                                                                                                                                      | dt.h                                                                                                                                                                           |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#include "dt.h"  int dt_base_value; static int dt_answer;  static void run_computer(int multiplier) {     dt_answer = dt_base_value * multiplier; }  void dt_init(void) {     dt_base_value = 6; }  int dt_compute_answer(void) {     run_computer(7);     return dt_answer; }</pre> | <pre>void dt_init(void); int dt_compute_answer(void);</pre>                                                                                                                    |
|                                                                                                                                                                                                                                                                                           | theanswer.c                                                                                                                                                                    |
|                                                                                                                                                                                                                                                                                           | <pre>#include &lt;stdio.h&gt; #include "dt.h"  int main(void) {     dt_init();     int answer = dt_compute_answer();     printf("The answer is %d\n",         answer); }</pre> |

```
$ cc -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break main
(gdb) run
(gdb) next
(gdb) step
(gdb) cont
(gdb) set disassembly-flavor intel
```



# Deep thought, Part 3

| dt.c                                                                                                                                                                                                                                                                                      | dt.h                                                                                                                                                                                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#include "dt.h"  int dt_base_value; static int dt_answer;  static void run_computer(int multiplier) {     dt_answer = dt_base_value * multiplier; }  void dt_init(void) {     dt_base_value = 6; }  int dt_compute_answer(void) {     run_computer(7);     return dt_answer; }</pre> | <pre>void dt_init(void); int dt_compute_answer(void);</pre>                                                                                                                                 |
|                                                                                                                                                                                                                                                                                           | <pre>theanswer.c  #include &lt;stdio.h&gt; #include "dt.h"  int main(void) {     dt_init();     int answer = dt_compute_answer();     printf("The answer is %d\n",         answer); }</pre> |

```
$ cc -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break main
(gdb) run
(gdb) next
(gdb) step
(gdb) cont
(gdb) set disassembly-flavor intel
(gdb) disassemble run_computer
```

# Deep thought, Part 3

| dt.c                                                                                                                                                                                                                                                                                      | dt.h                                                                                                                                                                           |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#include "dt.h"  int dt_base_value; static int dt_answer;  static void run_computer(int multiplier) {     dt_answer = dt_base_value * multiplier; }  void dt_init(void) {     dt_base_value = 6; }  int dt_compute_answer(void) {     run_computer(7);     return dt_answer; }</pre> | <pre>void dt_init(void); int dt_compute_answer(void);</pre>                                                                                                                    |
|                                                                                                                                                                                                                                                                                           | theanswer.c                                                                                                                                                                    |
|                                                                                                                                                                                                                                                                                           | <pre>#include &lt;stdio.h&gt; #include "dt.h"  int main(void) {     dt_init();     int answer = dt_compute_answer();     printf("The answer is %d\n",         answer); }</pre> |

```
$ cc -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break main
(gdb) run
(gdb) next
(gdb) step
(gdb) cont
(gdb) set disassembly-flavor intel
(gdb) disassemble run_computer
(gdb) help
```

# Deep thought, Part 3

| dt.c                                                                                                                                                                                                                                                                                      | dt.h                                                        | theanswer.c                                                                                                                                                                    |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>#include "dt.h"  int dt_base_value; static int dt_answer;  static void run_computer(int multiplier) {     dt_answer = dt_base_value * multiplier; }  void dt_init(void) {     dt_base_value = 6; }  int dt_compute_answer(void) {     run_computer(7);     return dt_answer; }</pre> | <pre>void dt_init(void); int dt_compute_answer(void);</pre> | <pre>#include &lt;stdio.h&gt; #include "dt.h"  int main(void) {     dt_init();     int answer = dt_compute_answer();     printf("The answer is %d\n",         answer); }</pre> |

```
$ cc -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break main
(gdb) run
(gdb) next
(gdb) step
(gdb) cont
(gdb) set disassembly-flavor intel
(gdb) disassemble run_computer
(gdb) help
(gdb) quit
```

A few words about  
memory, activation frames and storage durations

# Memory Layout and Activation Record

Under the hood

*(This is a simplified view of how a possible memory layout might look like. Some architectures and run-time environment uses a very different layout and the C standard does not dictate how it should look like. However, without actually knowing the memory layout on your target machine, the description here is good enough as a conceptual model)*



# Memory Layout \*

It is sometimes useful to assume that a C program uses a memory model where the instructions are stored in a **text segment**, and static variables are stored in a **data segment**. Automatic variables are allocated when needed together with housekeeping variables on an **execution stack** that is growing towards low address. The remaining memory, the **heap** is used for allocated storage.

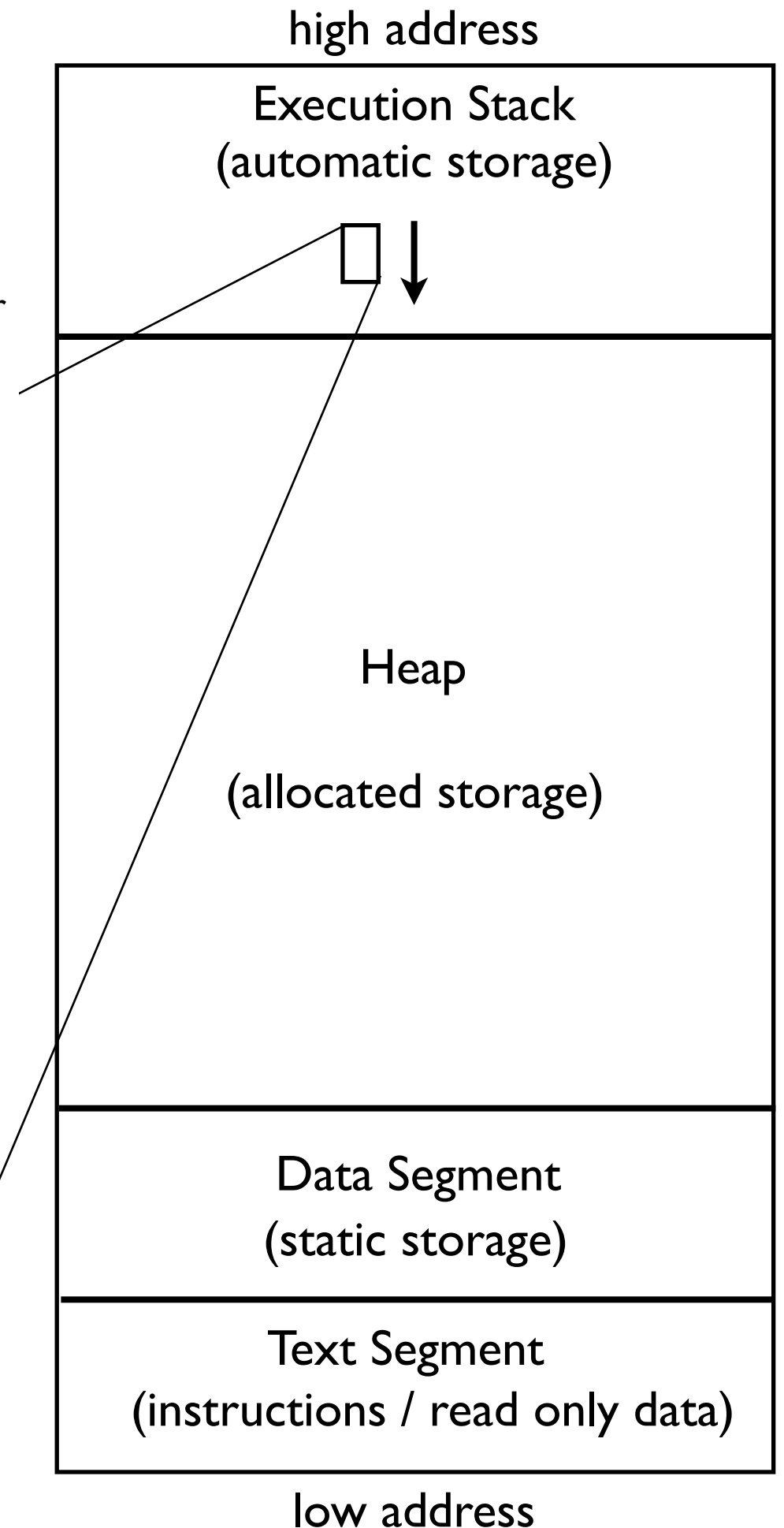
The stack and the heap is typically not cleaned up in any way at startup, or during execution, so before objects are explicitly initialized they typically get garbage values based on whatever is left in memory from discarded objects and previous executions. In other words, the programmer must do all the housekeeping on variables with automatic storage and allocated storage.

## Activation Record

And sometimes it is useful to assume that an **activation record** is created and pushed onto the execution stack every time a function is called. The activation record contains local auto variables, arguments to the functions, and housekeeping data such as pointer to the previous frame and the return address.

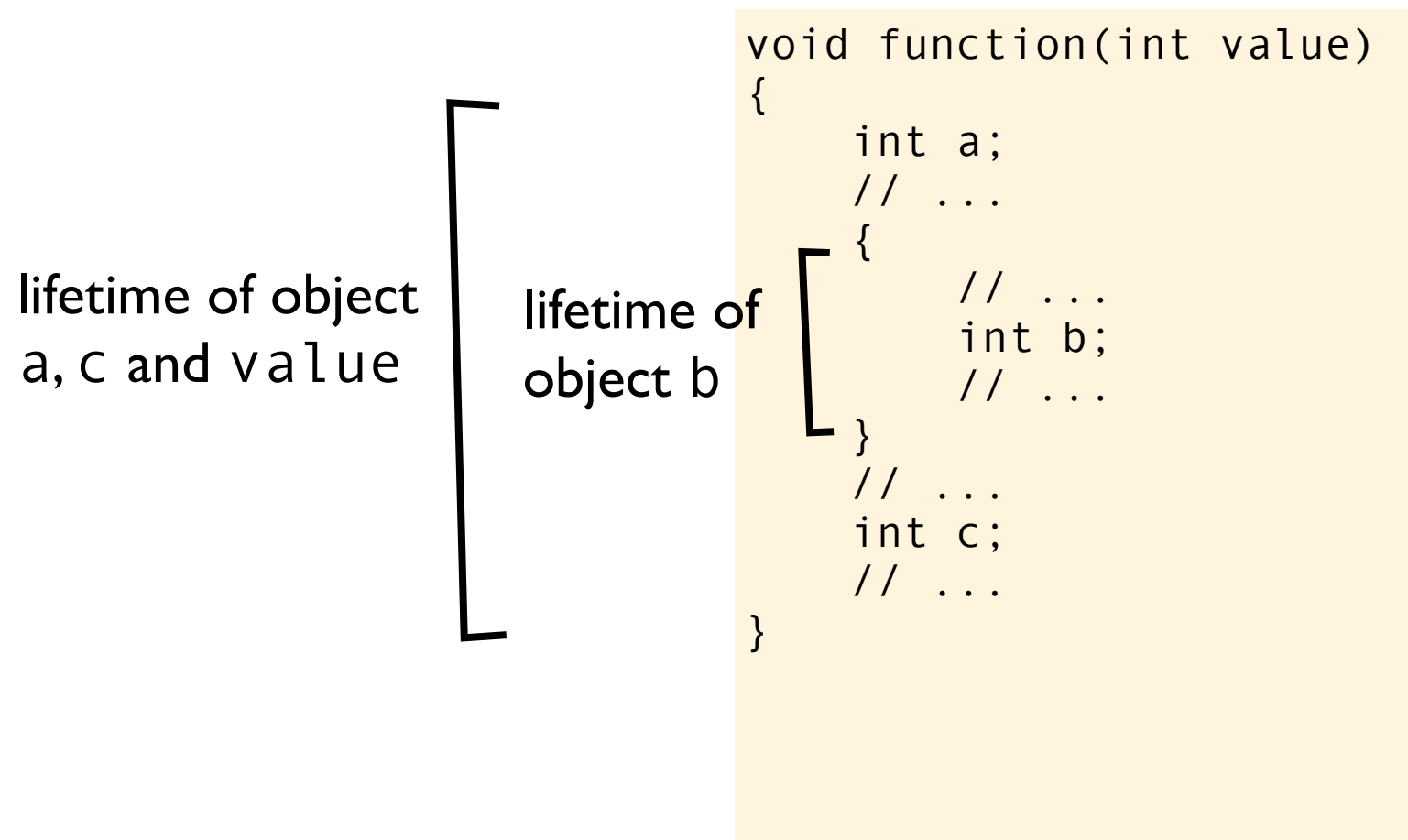


(\*) The C standard does not dictate any particular memory layout, so what is presented here is just a useful conceptual example model that is similar to what some architecture and run-time environments look like



# Automatic storage duration

Variables declared inside a “block”, eg the function body, come into existence when entering the block and disappears when exiting the block.\* These objects have automatic storage class and they get an indeterminate initial value. Reading an indeterminate value causes **undefined behavior**. Referring to an object outside it's lifetime also causes **undefined behavior**.



(\*) unless declared with the `static` keyword, see other slide



# Static storage duration

A variable declared outside a function scope, or with the `static` keyword inside a function, comes into existence and is initialized at program startup and its lifetime does not end until the program exits.

lifetime of static variables, from program startup to program exit

```
int global_var;

int next_value(void)
{
    static int local_var = 39;
    local_var += 1;
    return local_var;
}

int main(void)
{
    next_value();
    next_value();
    printf("The answer is %d\n",
           next_value());
    exit(0);
}
```

initialized to 0 at program startup

initialized to 39 at program startup

increase the value of `local_var` every time this line expression is evaluated.

The answer is 42



# Allocated storage duration

lifetime of the  
object that b is  
pointing to

[

```
#include <stdlib.h>
#include <stdio.h>

static int * b;

int main(void)
{
    printf("The answer is");
    b = malloc(sizeof b);
    *b = 42;
    printf(" %d\n", *b);
    free(b);
    b = 0;
}
```

The answer is 42



## Summary

- hello world!
- behaviour
- vocabulary of the language
- compiler, translator, assembler, linker
- standard library and C run-time
- memory layout and execution stack

# Exercise: Deep thought

dt.c

```
#include "dt.h"

int dt_base_value;
#define MULTIPLIER 7
static int dt_answer;

static void run_computer(void)
{
    dt_answer = dt_base_value * MULTIPLIER;
}

int dt_get_answer(void)
{
    run_computer();
    return dt_answer;
}
```

dt.h

```
extern int dt_base_value;
int dt_get_answer(void);
```

theanswer.c

```
#include "dt.h"
#include <stdio.h>

int main(void)
{
    dt_base_value = 6;
    int answer = dt_get_answer();
    printf("The answer is %d\n",
           answer);
}
```

Type in and run this program.

Discuss the steps involved in compiling, linking and executing a program.

Extra: Use the debugger to step through the code. Also try to debug optimized code.

# Control Flow



What does the following code print?

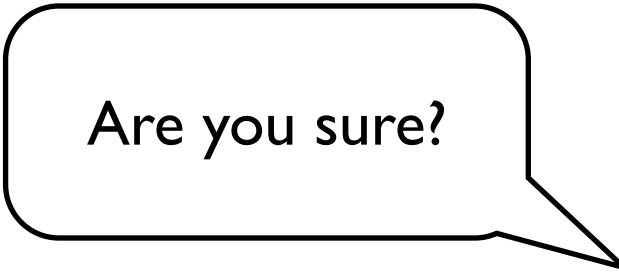
```
#include <stdio.h>

int main(void)
{
    for (int i = 3; i < 7; ++i)
        printf("%d\n", i);
}
```

What does the following code print?

```
#include <stdio.h>

int main(void)
{
    for (int i = 3; i < 7; ++i)
        printf("%d\n", i);
}
```



Are you sure?

What does the following code print?

```
#include <stdio.h>

int main(void)
{
    for (int i = 3; i < 7; ++i)
        printf("%d\n", i);
}
```

Are you sure?

Are you able to exactly describe the execution flow in this program?



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    if (remainder == 0)
        return true;
    else
        return false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    if (remainder == 0)
        return true;
    else
        return false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

Here is a small snippet that we will start with to illustrate basic control flow.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    if (remainder == 0)
        return true;
    else
        return false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

Here is a small snippet that we will start with to illustrate basic control flow.

We have defined a function `is_divisible_by()` that returns true if the first argument can be divided by the second argument without a remainder.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    if (remainder == 0)
        return true;
    else
        return false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

Here is a small snippet that we will start with to illustrate basic control flow.

We have defined a function `is_divisible_by()` that returns true if the first argument can be divided by the second argument without a remainder.

So in `main()`, if `n` is divisible by 2 then we print out a string saying that the number is even.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    if (remainder == 0)
        return true;
    else
        return false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    if (remainder == 0)
        return true;
    else
        return false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

*if ( condition ) statement*

*if ( condition ) statement **else** statement*

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    if (remainder == 0)
        return true;
    else
        return false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

These two examples illustrates both a simple if-statement and an if-else-statement.

*if ( condition ) statement*

*if ( condition ) statement **else** statement*

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    if (remainder == 0)
        return true;
    else
        return false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    if (remainder == 0)
        return true;
    else
        return false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

***return** expression<sub>opt</sub>;*

*condition ? expression : expression;*



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    if (remainder == 0)
        return true;
    else
        return false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

Another way of selecting between two alternatives is to use a conditional-expression.

***return** expression<sub>opt</sub>;*

*condition ? expression : expression;*

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    if (remainder == 0)
        return true;
    else
        return false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

Another way of selecting between two alternatives is to use a conditional-expression.

Here we can mix the conditional-expression directly into the return statement. Like this...

***return** expression<sub>opt</sub>;*

*condition ? expression : expression;*

```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    if (remainder == 0)
        return true;
    else
        return false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}

```

Another way of selecting between two alternatives is to use a conditional-expression.

Here we can mix the conditional-expression directly into the return statement. Like this...

***return** expression<sub>opt</sub>;*

*condition ? expression : expression;*

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0 ? true : false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

***return** expression<sub>opt</sub>;*

*condition ? expression : expression;*

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0 ? true : false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

***return** expression<sub>opt</sub>;*

*condition ? expression : expression;*

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0 ? true : false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

***return** expression<sub>opt</sub>;*

*condition ? expression : expression;*

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0 ? true : false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

***return** expression<sub>opt</sub>;*

*condition ? expression : expression;*



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0 ? true : false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

***return** expression<sub>opt</sub>;*

*condition ? expression : expression;*



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0 ? true : false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

The conditional-expression is sometimes just called the ternary-operator, as it is the only operator in C that takes three operands.

***return** expression<sub>opt</sub>;*

*condition ? expression : expression;*

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0 ? true : false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

The conditional-expression is sometimes just called the ternary-operator, as it is the only operator in C that takes three operands.

But the condition `remainder == 0` is also an expression with can be directly converted to a bool so in this particular case we can just as well write.

***return** expression<sub>opt</sub>;*

*condition ? expression : expression;*

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0 ? true : false;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

The conditional-expression is sometimes just called the ternary-operator, as it is the only operator in C that takes three operands.

But the condition `remainder == 0` is also an expression with can be directly converted to a bool so in this particular case we can just as well write.

***return** expression<sub>opt</sub>;*

*condition ? expression : expression;*

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

Suppose we want to write out  
all even numbers between 5 and  
9 inclusive. Then we can of  
course write...

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 42;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

Suppose we want to write out  
all even numbers between 5 and  
9 inclusive. Then we can of  
course write...



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
    ++n;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
    ++n;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
    ++n;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
    ++n;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
    ++n;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
    ++n;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
    ++n;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
    ++n;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
    ++n;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
    ++n;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
    ++n;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
    ++n;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
    ++n;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
    ++n;
    if (is_divisible_by(n, 2))
        printf("%d is even\n", n);
}
```

Hmm... this does not work for larger ranges. But hey, as I know how to program machine code, I have an idea!

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

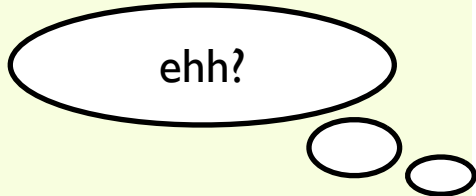
int main(void)
{
    int n = 5;
again:
    bool is_even = is_divisible_by(n, 2);
    if (!is_even)
        goto is_not_even;
    printf("%d is even\n", n);
is_not_even:
    ++n;
    if (n < 10)
        goto again;
}
```



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
again:
    bool is_even = is_divisible_by(n, 2);
    if (!is_even)
        goto is_not_even;
    printf("%d is even\n", n);
is_not_even:
    ++n;
    if (n < 10)
        goto again;
}
```



ehh?



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
again:
    bool is_even = is_divisible_by(n, 2);
    if (!is_even)
        goto is_not_even;
    printf("%d is even\n", n);
is_not_even:
    ++n;
    if (n < 10)
        goto again;
}
```

ehh?

Yes. C supports goto!

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
again:
    bool is_even = is_divisible_by(n, 2);
    if (!is_even)
        goto is_not_even;
    printf("%d is even\n", n);
is_not_even:
    ++n;
    if (n < 10)
        goto again;
}
```

ehh?

Yes. C supports goto!

This looks very similar to  
actual machine code executed  
by the CPU.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
again:
    bool is_even = is_divisible_by(n, 2);
    if (!is_even)
        goto is_not_even;
    printf("%d is even\n", n);
is_not_even:
    ++n;
    if (n < 10)
        goto again;
}
```

ehh?

Yes. C supports goto!

This looks very similar to  
actual machine code executed  
by the CPU.

But while C is sometimes described as just a portable  
assembler it does have some high-level language features.  
For example...

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
again:
    bool is_even = is_divisible_by(n, 2);
    if (!is_even)
        goto is_not_even;
    printf("%d is even\n", n);
is_not_even:
    ++n;
    if (n < 10)
        goto again;
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    again:
    bool is_even = is_divisible_by(n, 2);
    if (!is_even)
        goto is_not_even;
    printf("%d is even\n", n);
    is_not_even:
    ++n;
    if (n < 10)
        goto again;
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
again:
    bool is_even = is_divisible_by(n, 2);
    if ( is_even)

        printf("%d is even\n", n);

    ++n;
    if (n < 10)
        goto again;
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
again:
    bool is_even = is_divisible_by(n, 2);
    if (is_even)
        printf("%d is even\n", n);
    ++n;
    if (n < 10)
        goto again;
}
```



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
again:
    bool is_even = is_divisible_by(n, 2);
    if (is_even)
        printf("%d is even\n", n);
    ++n;
    if (n < 10)
        goto again;
}
```



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    again: 
    bool is_even = is_divisible_by(n, 2);
    if (is_even)
        printf("%d is even\n", n);
    ++n;
     if (n < 10)
        goto again;
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    again: { 
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
 } if (n < 10)
        goto again;
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
again: {
    bool is_even = is_divisible_by(n, 2);
    if (is_even)
        printf("%d is even\n", n);
    ++n;
    } if (n < 10)
        goto again;
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
again: {
    bool is_even = is_divisible_by(n, 2);
    if (is_even)
        printf("%d is even\n", n);
    ++n;
    } if (n < 10)
        goto again;
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
again: {
    bool is_even = is_divisible_by(n, 2);
    if (is_even)
        printf("%d is even\n", n);
    ++n;
} if (n < 10)
    goto again;
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    again: {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    } if (n < 10)
        goto again;
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    do {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    } while (n < 10);
}
```



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    do {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    } while (n < 10);
}
```

***do statement while ( expression ) ;***



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    do {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    } while (n < 10);
}
```

*do statement while ( expression ) ;*

This is a **do statement** for iteration. It is not used so much as it is often better to use an iteration statement where you can start by testing the **loop invariant**.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    do {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    } while (n < 10);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    while (n < 10) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}
```

***while** ( condition ) statement*

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    while (n < 10) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    while (n < 10) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}
```

This is a **while statement**.  
It is a versatile and very much  
used loop construct.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    while (n < 10) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}
```

This is a **while statement**.  
It is a versatile and very much  
used loop construct.

However, there is a certain  
way of using while-loops that  
is so common that there is a  
“short” hand notation for  
those types of while-loops.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

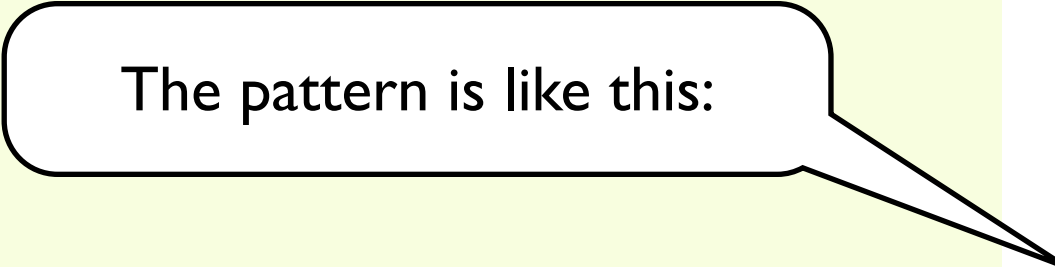
int main(void)
{
    int n = 5;
    while (n < 10) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}
```



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    while (n < 10) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}
```



The pattern is like this:



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    while (n < 10) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}
```

initializing a loop variable

testing loop variable

update loop variable

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    while (n < 10) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}
```

initializing a loop variable

testing loop variable

update loop variable

And this is sometimes better  
written as a for-loop

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    while (n < 10) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}
```

*init*  
**while** ( *check* )  
    *do something*  
    *update*



often equivalent ways  
of writing a loop

**for** (*init*; *check*; *update*)  
    *do something*

```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    while (n < 10) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}

```

A while-loop can often conveniently be rewritten into a for-loop, and vice versa. For loops are often used instead when there is a the pattern as just described.

*init*  
**while** ( *check* )  
 do something  
 update



often equivalent ways  
of writing a loop

**for** (*init*; *check*; *update*)  
 do something

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    while (n < 10) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}
```

*init*  
**while** ( *check* )  
  *do something*  
  *update*



often equivalent ways  
of writing a loop

**for** (*init*; *check*; *update*)  
  *do something*

```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    while ( n < 10 ) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}

```

*init*  
**while** ( *check* )  
     *do something*  
     *update*



often equivalent ways  
of writing a loop

**for** (*init*; *check*; *update*)  
     *do something*

```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    while ( n < 10 ) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}

```

*init*  
**while** ( *check* )  
     *do something*  
     *update*



often equivalent ways  
of writing a loop

**for** (*init*; *check*; *update*)  
     *do something*



```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    for (      ; n < 10;      ) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}

```

*init*  
**while** ( *check* )  
     *do something*  
     *update*



often equivalent ways  
of writing a loop

**for** (*init*; *check*; *update*)  
     *do something*



```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    int n = 5;
    for (      ; n < 10;      ) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}

```

*init*  
**while** ( *check* )  
     *do something*  
     *update*



often equivalent ways  
of writing a loop

**for** (*init*; *check*; *update*)  
     *do something*

```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 5; n < 10; ) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}

```

*init*  
**while** ( *check* )  
     *do something*  
     *update*



often equivalent ways  
of writing a loop

**for** (*init*; *check*; *update*)  
     *do something*

```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 5; n < 10;    ) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
        ++n;
    }
}

```

*init*  
**while** ( *check* )  
     *do something*  
     *update*



often equivalent ways  
of writing a loop

**for** (*init*; *check*; *update*)  
     *do something*

```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 5; n < 10; ++n) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
    }
}

```

*init*  
**while** ( *check* )  
     *do something*  
     *update*



often equivalent ways  
of writing a loop

**for** (*init*; *check*; *update*)  
     *do something*

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 5; n < 10; ++n) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
    }
}
```

*init*  
**while** ( *check* )  
do something  
*update*



often equivalent ways  
of writing a loop

**for** (*init*; *check*; *update*)  
do something

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 5; n < 10; ++n) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
    }
}
```

*init*  
**while** ( *check* )  
    *do something*  
    *update*



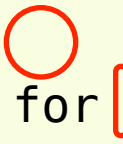
often equivalent ways  
of writing a loop

**for** (*init*; *check*; *update*)  
    *do something*

```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    
    for (int n = 5; n < 10; ++n) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
    }
}

```

*init*  
**while** ( *check* )  
     *do something*  
     *update*



often equivalent ways  
of writing a loop

**for** (*init*; *check*; *update*)  
     *do something*



```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 5; n < 10; ++n) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
    }
}

```

*init*  
**while** ( *check* )  
     *do something*  
     *update*



often equivalent ways  
of writing a loop

**for** (*init*; *check*; *update*)  
     *do something*



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 5; n < 10; ++n) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
    }
}
```

*init*  
**while** ( *check* )  
    *do something*  
    *update*



often equivalent ways  
of writing a loop

**for** (*init*; *check*; *update*)  
    *do something*

```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 5; n < 10; ++n) {
        bool is_even = is_divisible_by(n, 2);
        if (is_even)
            printf("%d is even\n", n);
    }
}

```

A significant problem with the for-loop however, is that, especially when used for more complex things it can become very hard to reason about what really happens. Then a straight while loops is often better to use.

*init*  
**while** ( *check* )  
*do something*  
*update*



often equivalent ways  
of writing a loop

**for** (*init; check; update*)  
*do something*



Let's expand our program a  
bit.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (is_divisible_by(n, 6) && is_divisible_by(n, 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (is_divisible_by(n, 6) && is_divisible_by(n, 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}
```

Initially we demonstrate a more elaborate if-else construct, and also to show that the if-condition can be a complex expression.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (is_divisible_by(n, 6) && is_divisible_by(n, 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}
```

Initially we demonstrate a more elaborate if-else construct, and also to show that the if-condition can be a complex expression.



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (is_divisible_by(n, 6) && is_divisible_by(n, 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}
```

Initially we demonstrate a more elaborate if-else construct, and also to show that the if-condition can be a complex expression.

If your math skills are solid you might immediately recognize that the first if-condition can be written like...



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (is_divisible_by(n, 6 * 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (is_divisible_by(n, 6 * 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}
```

But now you must question yourself, was it an improvement? Did it increase **readability**? Did it decrease **explainability**?

What do you think?

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (is_divisible_by(n, 6 * 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (is_divisible_by(n, 6 * 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}
```

Let's demonstrate two statements that are sometimes very useful, but often makes the code very hard to understand and reason about.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (is_divisible_by(n, 6 * 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}
```

Let's demonstrate two statements that are sometimes very useful, but often makes the code very hard to understand and reason about.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (is_divisible_by(n, 6 * 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (n == 42) {
            printf("42 is the answer\n");
            continue;
        }
        if (is_divisible_by(n, 6 * 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}
```



```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (n == 42) {
            printf("42 is the answer\n");
            continue; ←
        }
        if (is_divisible_by(n, 6 * 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}

```

§6.6.2 The **continue** statement shall occur only in an iteration-statement and causes control to pass to the loop-continuation portion of the smallest enclosing iteration-statement, that is, to the end of the loop.



```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (n == 42) {
            printf("42 is the answer\n");
            continue;
        }
        if (is_divisible_by(n, 6 * 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}

```

This is the **continue** statement.

But what does it do?

§6.6.2 The **continue** statement shall occur only in an iteration-statement and causes control to pass to the loop-continuation portion of the smallest enclosing iteration-statement, that is, to the end of the loop.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (n == 42) {
            printf("42 is the answer\n");
            continue;
        }
        if (is_divisible_by(n, 6 * 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (n == 42) {
            printf("42 is the answer\n");
            continue;
        }
        if (is_divisible_by(n, 6 * 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (n == 42) {
            printf("42 is the answer\n");
            break;
        }
        if (is_divisible_by(n, 6 * 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}
```

```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (n == 42) {
            printf("42 is the answer\n");
            break;
        }
        if (is_divisible_by(n, 6 * 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}

```

§6.6.1 The **break** statement shall occur only in an iteration-statement or a switch statement and causes termination of the smallest enclosing iteration-statement or switch statement; control passes to the statement following the terminated statement, if any.

```

#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

int main(void)
{
    for (int n = 30; n < 60; ++n) {
        if (n == 42) {
            printf("42 is the answer\n");
            break;
        }
        if (is_divisible_by(n, 6 * 7))
            printf("%d is divisible by 6 and 7\n", n);
        else if (is_divisible_by(n, 7))
            printf("%d is divisible by 7\n", n);
        else if (is_divisible_by(n, 6))
            printf("%d is divisible by 6\n", n);
    }
}

```

Here is an example of the  
**break statement**.

What does it do?

§6.6.1 The **break** statement shall occur only in an iteration-statement or a switch statement and causes termination of the smallest enclosing iteration-statement or switch statement; control passes to the statement following the terminated statement, if any.





Whenever you use `break` or `continue` you should consider if there is a better way of writing the code. Especially `continue` is inviting bugs into your code.



Whenever you use `break` or `continue` you should consider if there is a better way of writing the code. Especially `continue` is inviting bugs into your code.

However, there is one place where `break` is both essential and very much used. In a `switch` statement.

```

#include <stdio.h>

int main(void)
{
    while (1) {
        printf("* Type a single digit number (0 to exit): ");
        int num = 0;
        scanf("%d", &num);
        switch (num) {
            case 4:
                printf("%d is an even number.\n", num);
            case 1:
            case 9:
                printf("%d is a perfect square.\n", num);
                break;
            case 2:
                printf("%d is an even number.\n", num);
            case 3:
            case 5:
            case 7:
                printf("%d is a prime number.\n", num);
                break;
            case 6:
            case 8:
                printf("%d is an even number.\n", num);
                break;
            case 0:
                printf("bye, bye...\n");
                return 0;
            default:
                printf("Only single-digit numbers are allowed.\n");
        }
    }
}

```

## *switch ( condition ) statement*

```
#include <stdio.h>

int main(void)
{
    while (1) {
        printf("* Type a single digit number (0 to exit): ");
        int num = 0;
        scanf("%d", &num);
        switch (num) {
            case 4:
                printf("%d is an even number.\n", num);
            case 1:
            case 9:
                printf("%d is a perfect square.\n", num);
                break;
            case 2:
                printf("%d is an even number.\n", num);
            case 3:
            case 5:
            case 7:
                printf("%d is a prime number.\n", num);
                break;
            case 6:
            case 8:
                printf("%d is an even number.\n", num);
                break;
            case 0:
                printf("bye, bye...\n");
                return 0;
            default:
                printf("Only single-digit numbers are allowed.\n");
        }
    }
}
```

## *switch ( condition ) statement*

```
#include <stdio.h>

int main(void)
{
    while (1) {
        printf("* Type a single digit number (0 to exit): ");
        int num = 0;
        scanf("%d", &num);
        switch (num) {
            case 4:
                printf("%d is an even number.\n", num);
            case 1:
            case 9:
                printf("%d is a perfect square.\n", num);
                break;
            case 2:
                printf("%d is an even number.\n", num);
            case 3:
            case 5:
            case 7:
                printf("%d is a prime number.\n", num);
                break;
            case 6:
            case 8:
                printf("%d is an even number.\n", num);
                break;
            case 0:
                printf("bye, bye...\n");
                return 0;
            default:
                printf("Only single-digit numbers are allowed.\n");
        }
    }
}
```

The switch statement is difficult to formally describe, but easy to understand by just looking at an example. The case statements are labels and the switch works like an advanced parameterized goto. break is used to jump out of the case. Without the break, the control will just “fall through” to the next case.



Let's revisit our  
`is_divisible_by()` example

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2))
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2))
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

Here we have two functions (in addition to main).



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2))
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

Here we have two functions (in addition to main).

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2))
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

Here we have two functions (in addition to main).

Just by looking at their names, do you recognize a difference?

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2))
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

Here we have two functions (in addition to main).

Just by looking at their names, do you recognize a difference?

The first is asking to compute something and return an answer. It is a typical example of a **query function**. Query functions usually have no side-effects and they often try to be transient transparency.

The other function is a request for action, it is a typical **command function**, and they often do not return any value and are useful because of their side-effects.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2))
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

Here we have two functions (in addition to main).

Just by looking at their names, do you recognize a difference?

The first is asking to compute something and return an answer. It is a typical example of a **query function**. Query functions usually have no side-effects and they often try to be transient transparency.

The other function is a request for action, it is a typical **command function**, and they often do not return any value and are useful because of their side-effects.

When working with an imperative programming language it is often sensible to follow a principle where you try to make your functions to be *either* a query function or a command function... not both. This is often called the **command-query separation** principle.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2))
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2))
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

Another useful principle to follow is that whenever you feel a need to write a comment, you should rather try to express your code better.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2))
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

Another useful principle to follow is that whenever you feel a need to write a comment, you should rather try to express your code better.

For example...



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2))
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2)) 
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2)) // is even?
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2)) // is even?
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

Indeed, a very contrived example...

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2)) // is even?
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

Indeed, a very contrived example...

Then it is very often possible to express your code more explicitly by for example introducing a function. In this case, perhaps another query function `is_even()`

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}



---


void print_if_even(int number)
{
    if (is_divisible_by(number, 2)) // is even?
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}
```



```
void print_if_even(int number)
{
    if (is_divisible_by(number, 2)) // is even?
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

bool is_even(int number)
{
    return is_divisible_by(number, 2);
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2)) // is even?
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

bool is_even(int number)
{
    return is_divisible_by(number, 2);
}

void print_if_even(int number)
{
    if (is_divisible_by(number, 2)) // is even?
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```



```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

bool is_even(int number)
{
    return is_divisible_by(number, 2);
}

void print_if_even(int number)
{
    if (is_even(number)) // is even?
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

bool is_even(int number)
{
    return is_divisible_by(number, 2);
}

void print_if_even(int number)
{
    if (is_even(number)) // is even?
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

bool is_even(int number)
{
    return is_divisible_by(number, 2);
}

void print_if_even(int number)
{
    if (is_even(number)) // is even?
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

And now this comment just looks obviously unnecessary because we have expressed ourself much better in code.

```
#include <stdio.h>
#include <stdbool.h>

bool is_divisible_by(int numerator, int denominator)
{
    int remainder = numerator % denominator;
    return remainder == 0;
}

bool is_even(int number)
{
    return is_divisible_by(number, 2);
}

void print_if_even(int number)
{
    if (is_even(number))
        printf("%d is even\n", n);
}

int main(void)
{
    for (int n = 0; n < 10; ++n)
        print_if_even(n);
}
```

# Exercise:

Write a C program that prints out all the primes between 1234500 and 1234560 (there are 6).

Note: 1 is not a prime, while 2, 3, 5, 7, 11, 13, 17 and so on are primes. Negative numbers are not primes.

Hint: Here is a naive but simple algorithm that is perfectly fine for this exercise: If  $(n \% i \neq 0)$  for all values of  $i$  between 2 and  $n-1$  then  $n$  is a prime.

Work in pairs! When you are done, email your answer to me.

# Test-Driven Development

## a Yahtzee game kata



During the last decade Test-Driven Development has become an established practice for developing software in the industry.

Here we will demonstrate Test-Driven Development in C using a Yahtzee game kata.



# The requirement

Write a C library that can score the lower section of a game of yahtzee.



**Yahtzee** NAME \_\_\_\_\_

| UPPER SECTION                                     |                                                 | HOW TO SCORE              | GAME #1 | GAME #2 | GAME #3 | GAME #4 | GAME #5 | GAME #6 |
|---------------------------------------------------|-------------------------------------------------|---------------------------|---------|---------|---------|---------|---------|---------|
| Aces                                              | ● = 1                                           | Count and Add Only Aces   |         |         |         |         |         |         |
| Twos                                              | ●● = 2                                          | Count and Add Only Twos   |         |         |         |         |         |         |
| Threes                                            | ●●● = 3                                         | Count and Add Only Threes |         |         |         |         |         |         |
| Fours                                             | ●●●● = 4                                        | Count and Add Only Fours  |         |         |         |         |         |         |
| Fives                                             | ●●●●● = 5                                       | Count and Add Only Fives  |         |         |         |         |         |         |
| Sixes                                             | ●●●●●● = 6                                      | Count and Add Only Sixes  |         |         |         |         |         |         |
| TOTAL SCORE                                       |                                                 | →                         |         |         |         |         |         |         |
| BONUS <small>If total score is 63 or over</small> |                                                 | SCORE 35                  |         |         |         |         |         |         |
| TOTAL <small>Of Upper Section</small>             |                                                 | →                         |         |         |         |         |         |         |
| LOWER SECTION                                     |                                                 |                           |         |         |         |         |         |         |
| 3 of a kind                                       | Add Total Of All Dice                           |                           |         |         |         |         |         |         |
| 4 of a kind                                       | Add Total Of All Dice                           |                           |         |         |         |         |         |         |
| Full House                                        | SCORE 25                                        |                           |         |         |         |         |         |         |
| Sm. Straight <small>Sequence of 4</small>         | SCORE 30                                        |                           |         |         |         |         |         |         |
| Lg. Straight <small>Sequence of 5</small>         | SCORE 40                                        |                           |         |         |         |         |         |         |
| YAHTZEE <small>5 of a kind</small>                | SCORE 50                                        |                           |         |         |         |         |         |         |
| Chance                                            | Score Total Of All 5 Dice                       |                           |         |         |         |         |         |         |
| YAHTZEE BONUS                                     | <small>✓ FOR EACH BONUS SCORE 100 PER ✓</small> |                           |         |         |         |         |         |         |
| TOTAL <small>Of Lower Section</small>             | →                                               |                           |         |         |         |         |         |         |
| TOTAL <small>Of Upper Section</small>             | →                                               |                           |         |         |         |         |         |         |
| GRAND TOTAL                                       | →                                               |                           |         |         |         |         |         |         |

©1982, 1990, 1996 Milton Bradley Company. All Rights Reserved. EE100

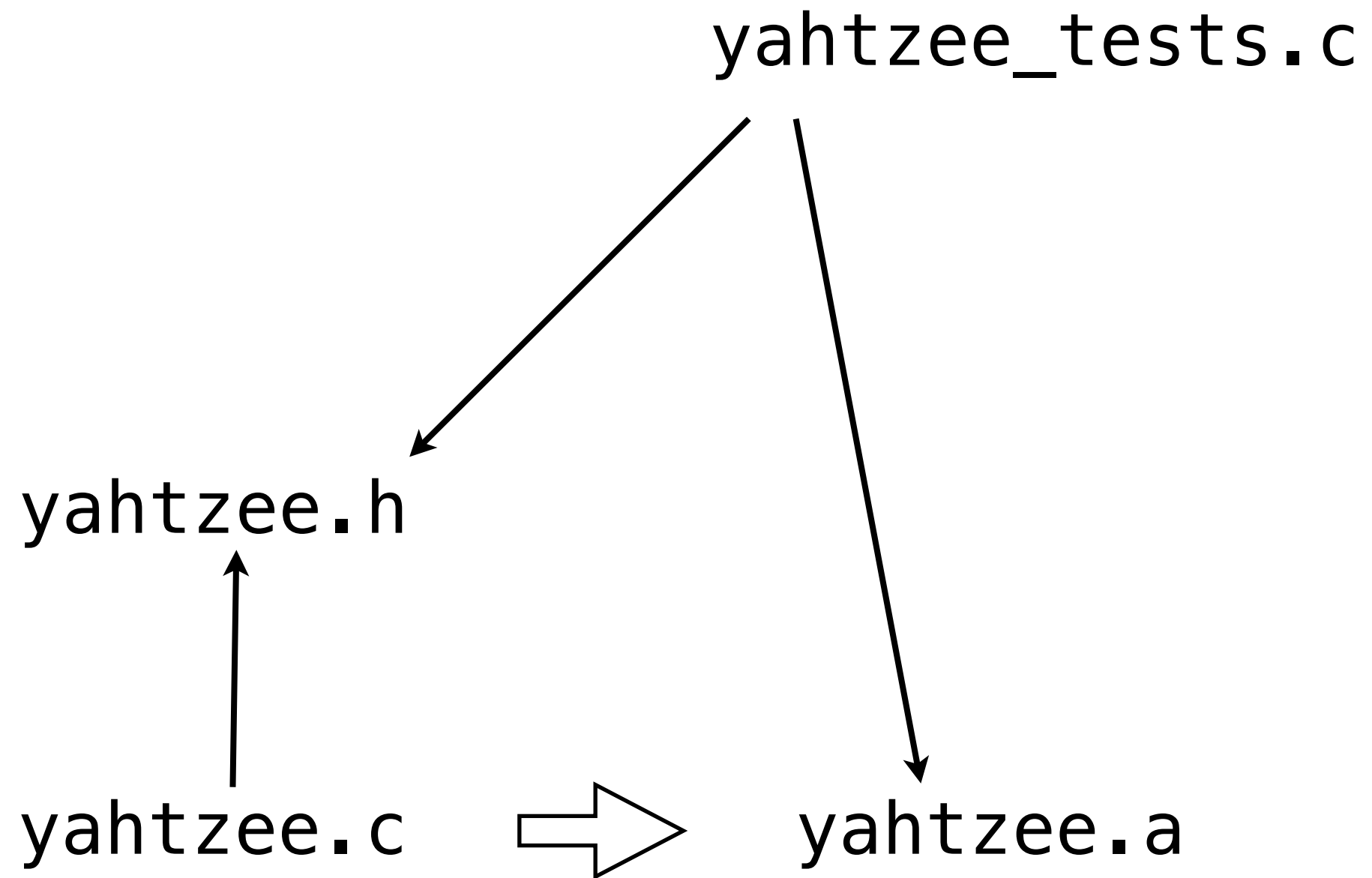


## LOWER SECTION

|                                               |                              |
|-----------------------------------------------|------------------------------|
| 3 of a kind                                   | Add Total<br>Of All Dice     |
| 4 of a kind                                   | Add Total<br>Of All Dice     |
| Full House                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small> | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small> | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small>        | SCORE 50                     |
| Chance                                        | Score Total<br>Of All 5 Dice |

## Examples





## Makefile

```
CC=gcc
CFLAGS=-std=c99 -O -Wall -Wextra -pedantic
LD=gcc

all: yahtzee.a

yahtzee.a: yahtzee.o
    ar -rcs $@ $^

check: yahtzee_tests
    ./yahtzee_tests

yahtzee.o: yahtzee.c yahtzee.h

yahtzee_tests.o: yahtzee_tests.c yahtzee.h

yahtzee_tests: yahtzee_tests.o yahtzee.a
    $(LD) -o $@ $^

clean:
    rm -f *.o *.a yahtzee_tests
```

yahtzee\_test.c

yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>
```

```
int main(void)
{
    assert(6*9 == 42);

    puts("Yahtzee tests OK");
}
```

yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(6*9 == 42);

    puts("Yahtzee tests OK");
}
```

```
$ make check
```

yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(6*9 == 42);

    puts("Yahtzee tests OK");
}
```

```
$ make check
gcc -std=c99 -O -Wall -Wextra -pedantic -c -o yahtzee_tests.o yahtzee_tests.c
```



yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(6*9 == 42);

    puts("Yahtzee tests OK");
}
```

```
$ make check
gcc -std=c99 -O -Wall -Wextra -pedantic -c -o yahtzee_tests.o yahtzee_tests.c
gcc -std=c99 -O -Wall -Wextra -pedantic -c -o yahtzee.o yahtzee.c
```



yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(6*9 == 42);

    puts("Yahtzee tests OK");
}
```

```
$ make check
gcc -std=c99 -O -Wall -Wextra -pedantic -c -o yahtzee_tests.o yahtzee_tests.c
gcc -std=c99 -O -Wall -Wextra -pedantic -c -o yahtzee.o yahtzee.c
ar -rcs yahtzee.a yahtzee.o
```

yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(6*9 == 42);

    puts("Yahtzee tests OK");
}
```

```
$ make check
gcc -std=c99 -O -Wall -Wextra -pedantic -c -o yahtzee_tests.o yahtzee_tests.c
gcc -std=c99 -O -Wall -Wextra -pedantic -c -o yahtzee.o yahtzee.c
ar -rcs yahtzee.a yahtzee.o
gcc -o yahtzee_tests yahtzee_tests.o yahtzee.a
```

yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(6*9 == 42);

    puts("Yahtzee tests OK");
}
```

```
$ make check
gcc -std=c99 -O -Wall -Wextra -pedantic -c -o yahtzee_tests.o yahtzee_tests.c
gcc -std=c99 -O -Wall -Wextra -pedantic -c -o yahtzee.o yahtzee.c
ar -rcs yahtzee.a yahtzee.o
gcc -o yahtzee_tests yahtzee_tests.o yahtzee.a
./yahtzee_tests
```

yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(6*9 == 42);

    puts("Yahtzee tests OK");
}
```

```
$ make check
gcc -std=c99 -O -Wall -Wextra -pedantic -c -o yahtzee_tests.o yahtzee_tests.c
gcc -std=c99 -O -Wall -Wextra -pedantic -c -o yahtzee.o yahtzee.c
ar -rcs yahtzee.a yahtzee.o
gcc -o yahtzee_tests yahtzee_tests.o yahtzee.a
./yahtzee_tests
Assertion failed: (6*9 == 42), function main, file yahtzee_tests.c, line 6.
```

yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(6*9 == 42);

    puts("Yahtzee tests OK");
}
```

```
$ make check
gcc -std=c99 -O -Wall yahtzee_tests.c yahtzee.o -o yahtzee_tests
ar -rcs yahtzee.a yahtzee.o
gcc -o yahtzee_tests yahtzee_tests.o yahtzee.a
./yahtzee_tests
Assertion failed: (6*9 == 42), function main, file yahtzee_tests.c, line 6.
```

Our first unit test failed. This is good! Exactly what we want. The rhythm of TDD is : fail, fix, pass... fail, fix, pass

yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(6*9 == 42);

    puts("Yahtzee tests OK");
}
```

```
$ make check
gcc -std=c99 -O -Wall yahtzee.c yahtzee_tests.c -o yahtzee.o yahtzee_tests.o
gcc -O yahtzee_tests yahtzee.o yahtzee.a
./yahtzee_tests
Assertion failed: (6*9 == 42), function main, file yahtzee_tests.c, line 6.
```

**Fail** - Fix - Pass

Our first unit test failed. This is good! Exactly what we want. The rhythm of TDD is : fail, fix, pass... fail, fix, pass

yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(6*9 == 42);
    puts("Yahtzee tests OK");
}
```



**Fail** - Fix - Pass

Our first unit test failed. This is good! Exactly what we want. The rhythm of TDD is : fail, fix, pass... fail, fix, pass

```
$ make check
gcc -std=c99 -O -Wall yahtzee_tests.c yahtzee.o -o yahtzee_tests
./yahtzee_tests
Assertion failed: (6*9 == 42), function main, file yahtzee_tests.c, line 6.
```



yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(6*7 == 42);

    puts("Yahtzee tests OK");
}
```



yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>
```

```
int main(void)
{
    assert(6*7 == 42);
    puts("Yahtzee tests OK");
}
```

Fail - **Fix** - Pass

yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>
```

```
int main(void)
{
    assert(6*7 == 42);

    puts("Yahtzee tests OK");
}
```

Fail - **Fix** - Pass

```
$ make check
gcc -std=c99 -O -Wall -Wextra -pedantic -c -o yahtzee_tests.o yahtzee_tests.c
gcc -o yahtzee_tests yahtzee_tests.o yahtzee.a
./yahtzee_tests
Yahtzee tests OK
```

Fail - Fix - **Pass**

yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>
```

```
int main(void)
{
    assert(6*7 == 42);
    puts("Yahtzee tests OK");
}
```

Fail - **Fix** - Pass

```
$ make check
gcc -std=c99 -O -Wall -Wextra -pedantic -c -o yahtzee_tests.o yahtzee_tests.c
gcc -o yahtzee_tests yahtzee_tests.o yahtzee.a
./yahtzee_tests
Yahtzee tests OK
```

yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>
```

```
int main(void)
{
    assert(6*7 == 42);
    puts("Yahtzee tests OK");
}
```

Fail - **Fix** - Pass

Fail - Fix - **Pass**

```
$ make check
gcc -std=c99 -O2 -o yahtzee_tests.o yahtzee_tests.c
gcc -o yahtzee_tests yahtzee_tests.o
./yahtzee_tests
Yahtzee tests OK
```

All tests are OK!

Let's write a proper unit test

yahtzee\_test.c

```
#include <assert.h>
#include <stdio.h>
```

```
int main(void)
{
    assert(6*7 == 42);
    puts("Yahtzee tests OK");
}
```

Fail - **Fix** - Pass

Fail - Fix - **Pass**

```
$ make check
gcc -std=c99 -O2 -o yahtzee_tests.o yahtzee_tests.c
gcc -o yahtzee_tests ./yahtzee_tests.o
Yahtzee tests OK
```

All tests are OK!

## LOWER SECTION

|                                               |                              |
|-----------------------------------------------|------------------------------|
| 3 of a kind                                   | Add Total<br>Of All Dice     |
| 4 of a kind                                   | Add Total<br>Of All Dice     |
| Full House                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small> | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small> | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small>        | SCORE 50                     |
| Chance                                        | Score Total<br>Of All 5 Dice |



## LOWER SECTION

**3 of a kind**

Add Total  
Of All Dice

**4 of a kind**

Add Total  
Of All Dice

**Full House**

SCORE 25

**Sm. Straight** Sequence  
of 4

SCORE 30

**Lg. Straight** Sequence  
of 5

SCORE 40

**YAHTZEE** 5 of  
a kind

SCORE 50

**Chance**

Score Total  
Of All 5 Dice





## LOWER SECTION

**3 of a kind**

**Add Total  
Of All Dice**

**4 of a kind**

**Add Total  
Of All Dice**

**Full House**

**SCORE 25**

**Sm. Straight** Sequence  
of 4

**SCORE 30**

**Lg. Straight** Sequence  
of 5

**SCORE 40**

**YAHTZEE** 5 of  
a kind

**SCORE 50**

**Chance**

**Score Total  
Of All 5 Dice**





yahtzee\_test.c

```
#include "yahtzee.h"  
#include <assert.h>  
#include <stdio.h>  
  
int main(void)  
{  
  
    puts("Yahtzee tests OK");  
}
```

yahtzee\_test.c

```
#include "yahtzee.h"  
#include <assert.h>  
#include <stdio.h>
```

```
int main(void)  
{
```



```
    puts("Yahtzee tests OK");  
}
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);

    puts("Yahtzee tests OK");
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"  
#include <assert.h>  
#include <stdio.h>
```

```
int main(void)  
{  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    puts("Yahtzee tests OK");  
}
```

yahtzee.c

```
#include "yahtzee.h"

int score_three_of_a_kind(const int dice[5])
{
    return ??
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);

    puts("Yahtzee tests OK");
}
```

yahtzee.c

```
#include "yahtzee.h"
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return ??  
}
```

what should we return?

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"  
#include <assert.h>  
#include <stdio.h>
```

```
int main(void)  
{  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    puts("Yahtzee tests OK");  
}
```

yahtzee.c

```
#include "yahtzee.h"
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return ??  
}
```

what should we return?

Remember fail-fix-pass? Return something that you know will fail.

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"  
#include <assert.h>  
#include <stdio.h>
```

```
int main(void)  
{  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    puts("Yahtzee tests OK");  
}
```



yahtzee.c

```
#include "yahtzee.h"

int score_three_of_a_kind(const int dice[5])
{
    return 0;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);

    puts("Yahtzee tests OK");
}
```



yahtzee.c

```
#include "yahtzee.h"

int score_three_of_a_kind(const int dice[5])
{
    return 0;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);

    puts("Yahtzee tests OK");
}
```

Assertion failed: (score\_three\_of\_a\_kind((int[5]){1,1,1,2,2}) == 3+2+2)

yahtzee.c

```
#include "yahtzee.h"

int score_three_of_a_kind(const int dice[5])
{
    return 0;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);

    puts("Yahtzee tests OK");
}
```

**Fail** - Fix - Pass

Assertion failed: (score\_three\_of\_a\_kind((int[5]){1,1,1,2,2}) == 3+2+2)

yahtzee.c

```
#include "yahtzee.h"

int score_three_of_a_kind(const int dice[5])
{
    return 0;
}
```



yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);

    puts("Yahtzee tests OK");
}
```

**Fail** - Fix - Pass

Assertion failed: (score\_three\_of\_a\_kind((int[5]){1,1,1,2,2}) == 3+2+2)

yahtzee.c

```
#include "yahtzee.h"

int score_three_of_a_kind(const int dice[5])
{
    return 7;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);

    puts("Yahtzee tests OK");
}
```

yahtzee.c

```
#include "yahtzee.h"

int score_three_of_a_kind(const int dice[5])
{
    return 7;
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);

    puts("Yahtzee tests OK");
}
```

yahtzee.c

```
#include "yahtzee.h"

int score_three_of_a_kind(const int dice[5])
{
    return 7;
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);

    puts("Yahtzee tests OK");
}
```

Yahtzee tests OK

yahtzee.c

```
#include "yahtzee.h"

int score_three_of_a_kind(const int dice[5])
{
    return 7;
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);

    puts("Yahtzee tests OK");
}
```

Fail - Fix - **Pass**

Yahtzee tests OK



yahtzee.c

```
#include "yahtzee.h"
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return 7;  
}
```

Congratulation. We have completed our first proper fail-fix-pass cycle.

Returning 7 is a minimal change to make it pass. This is OK because what we are concerned about now is just to make sure that the “wiring” of the test is OK. Is the test really being called and is it testing the right function?

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"  
#include <assert.h>  
#include <stdio.h>
```

```
int main(void)  
{  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    puts("Yahtzee tests OK");  
}
```

Fail - Fix - **Pass**

Yahtzee tests OK



yahtzee.c

```
#include "yahtzee.h"
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return 7;  
}
```

Congratulation. We have completed our first proper fail-fix-pass cycle.

Returning 7 is a minimal change to make it pass. This is OK because what we are concerned about now is just to make sure that the “wiring” of the test is OK. Is the test really being called and is it testing the right function?

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"  
#include <assert.h>  
#include <stdio.h>
```

```
int main(void)  
{  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    puts("Yahtzee tests OK");  
}
```

Let's add another unit test.

Fail - Fix - **Pass**

Yahtzee tests OK

yahtzee.c

```
#include "yahtzee.h"
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return 7;  
}
```

Congratulation. We have completed our first proper fail-fix-pass cycle.

Returning 7 is a minimal change to make it pass. This is OK because what we are concerned about now is just to make sure that the “wiring” of the test is OK. Is the test really being called and is it testing the right function?

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"  
#include <assert.h>  
#include <stdio.h>
```

```
int main(void)  
{  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    puts("Yahtzee tests OK");  
}
```

Let's add another unit test.

Fail - Fix - **Pass**

Yahtzee tests OK

yahtzee.c

```
#include "yahtzee.h"

int score_three_of_a_kind(const int dice[5])
{
    return 7;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    puts("Yahtzee tests OK");
}
```



yahtzee.c

```
#include "yahtzee.h"

int score_three_of_a_kind(const int dice[5])
{
    return 7;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);

    puts("Yahtzee tests OK");
}
```

yahtzee.c

```
#include "yahtzee.h"

int score_three_of_a_kind(const int dice[5])
{
    return 7;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);

    puts("Yahtzee tests OK");
}
```

Assertion failed: (score\_three\_of\_a\_kind((int[5]){1,1,1,3,4}) == 3+3+4)

yahtzee.c

```
#include "yahtzee.h"

int score_three_of_a_kind(const int dice[5])
{
    return 7;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);

    puts("Yahtzee tests OK");
}
```

**Fail** - Fix - Pass

Assertion failed: (score\_three\_of\_a\_kind((int[5]){1,1,1,3,4}) == 3+3+4)

yahtzee.c

```
#include "yahtzee.h"
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return 7;  
}
```

Should we “cheat” again and check for the last dice, if 4 then return 10 otherwise 7?

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"  
#include <assert.h>  
#include <stdio.h>
```

```
int main(void)  
{  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
  
    puts("Yahtzee tests OK");  
}
```

**Fail** - Fix - Pass

Assertion failed: (score\_three\_of\_a\_kind((int[5]){1,1,1,3,4}) == 3+3+4)



yahtzee.c

```
#include "yahtzee.h"
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return 7;  
}
```

Should we “cheat” again and check for the last dice, if 4 then return 10 otherwise 7?

yahtzee.h  
int score\_t

No! Another principle of TDD is that while you are supposed to do simple and “naive” increments you are not allowed to do “obviously stupid” stuff.

yahtzee\_test.c

```
#include "yahtzee.h"  
#include <assert.h>  
#include <stdio.h>
```

```
int main(void)  
{  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
  
    puts("Yahtzee tests OK");  
}
```

**Fail** - Fix - Pass

Assertion failed: (score\_three\_of\_a\_kind((int[5]){1,1,1,3,4}) == 3+3+4)



yahtzee.c

```
#include "yahtzee.h"
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return 7;  
}
```

Should we “cheat” again and check for the last dice, if 4 then return 10 otherwise 7?

yahtzee.h  
int score\_t

No! Another principle of TDD is that while you are supposed to do simple and “naive” increments you are not allowed to do “obviously stupid” stuff.

A simple and naive thing to do here is to just **sum the dice** and return the value. That would satisfy all the tests and we know the functionality will eventually be needed.

```
#include <assert.h>  
#include <stdio.h>
```

```
int main(void)  
{
```

```
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
```

```
    puts("Yahtzee tests OK");
```

```
}
```

**Fail - Fix - Pass**

Assertion failed: (score\_three\_of\_a\_kind((int[5]){1,1,1,3,4}) == 3+3+4)

yahtzee.c

```
#include "yahtzee.h"

int score_three_of_a_kind(const int dice[5])
{
    return 7;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);

    puts("Yahtzee tests OK");
}
```

yahtzee.c

```
#include "yahtzee.h"
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return 7;  
}
```

yahtzee.c

```
#include "yahtzee.h"
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return 7;  
}
```



yahtzee.c

```
#include "yahtzee.h"
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```

yahtzee.c

#include "yahtzee.h"



```
int score_three_of_a_kind(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```

yahtzee.c

```
#include "yahtzee.h"
```



```
int score_three_of_a_kind(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```



yahtzee.c

```
#include "yahtzee.h"
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (int die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```



yahtzee.c

```
#include "yahtzee.h"
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (int die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```



Fail - **Fix** - Pass

yahtzee.c

```
#include "yahtzee.h"
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (int die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```

Fail - **Fix** - Pass

Yahtzee tests OK

yahtzee.c

```
#include "yahtzee.h"
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (int die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```

Fail - **Fix** - Pass

Fail - Fix - **Pass**

Yahtzee tests OK

yahtzee.c

```
#include "yahtzee.h"
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (int die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```

But wait a minute. When indexing an array in C you should really use **size\_t**. Let's fix it now as all tests are OK

Fail - **Fix** - Pass

Fail - Fix - **Pass**

Yahtzee tests OK

yahtzee.c

```
#include "yahtzee.h"
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (int die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```

yahtzee.c

#include "yahtzee.h"



```
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (int die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}
```

```
int score_three_of_a_kind(const int dice[5])
{
    return sum_of_dice(dice);
}
```

yahtzee.c



```
#include "yahtzee.h"
```

```
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (int die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}
```

```
int score_three_of_a_kind(const int dice[5])
{
    return sum_of_dice(dice);
}
```



yahtzee.c

```
#include "yahtzee.h"  
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (int die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```



yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])
```

```
{
```

```
    int sum = 0;
```

```
    for (int die = 0; die < 5; die++)
```

```
        sum += dice[die];
```

```
    return sum;
```

```
}
```

```
int score_three_of_a_kind(const int dice[5])
```

```
{
```

```
    return sum_of_dice(dice);
```

```
}
```

yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])
```

```
{
```

```
    int sum = 0;
```

```
    for (size_t die = 0; die < 5; die++)
```

```
        sum += dice[die];
```

```
    return sum;
```

```
}
```

```
int score_three_of_a_kind(const int dice[5])
```

```
{
```

```
    return sum_of_dice(dice);
```

```
}
```

yahtzee.c

```
#include "yahtzee.h"  
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (size_t die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```

yahtzee.c

```
#include "yahtzee.h"
#include <stddef.h>

static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

int score_three_of_a_kind(const int dice[5])
{
    return sum_of_dice(dice);
}
```

Yahtzee tests OK

yahtzee.c

```
#include "yahtzee.h"
#include <stddef.h>

static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

int score_three_of_a_kind(const int dice[5])
{
    return sum_of_dice(dice);
}
```

Nice. Let's start a new  
fail-fix-pass cycle

Yahtzee tests OK

yahtzee.c

```
#include "yahtzee.h"
#include <stddef.h>

static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

int score_three_of_a_kind(const int dice[5])
{
    return sum_of_dice(dice);
}
```

Nice. Let's start a new  
fail-fix-pass cycle



Yahtzee tests OK

yahtzee.c

```
#include "yahtzee.h"
#include <stddef.h>

static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

int score_three_of_a_kind(const int dice[5])
{
    return sum_of_dice(dice);
}
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);

    puts("Yahtzee tests OK");
}
```



yahtzee.c

```
#include "yahtzee.h"
#include <stddef.h>

static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

int score_three_of_a_kind(const int dice[5])
{
    return sum_of_dice(dice);
}
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);

    puts("Yahtzee tests OK");
}
```





yahtzee.c

```
#include "yahtzee.h"
#include <stddef.h>

static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

int score_three_of_a_kind(const int dice[5])
{
    return sum_of_dice(dice);
}
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);

    puts("Yahtzee tests OK");
}
```

yahtzee.c

```
#include "yahtzee.h"
#include <stddef.h>

static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

int score_three_of_a_kind(const int dice[5])
{
    return sum_of_dice(dice);
}
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);

    puts("Yahtzee tests OK");
}
```

Assertion failed: (score\_three\_of\_a\_kind((int[5]){1,2,3,4,5}) == 0)

yahtzee.c

```
#include "yahtzee.h"
#include <stddef.h>

static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

int score_three_of_a_kind(const int dice[5])
{
    return sum_of_dice(dice);
}
```

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);

    puts("Yahtzee tests OK");
}
```

**Fail** - Fix - Pass

Assertion failed: (score\_three\_of\_a\_kind((int[5]){1,2,3,4,5}) == 0)

yahtzee.c

```
#include "yahtzee.h"
#include <stddef.h>

static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

int score_three_of_a_kind(const int dice[5])
{
    return sum_of_dice(dice);
}
```

So how do we fix this?

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);

    puts("Yahtzee tests OK");
}
```

**Fail** - Fix - Pass

Assertion failed: (score\_three\_of\_a\_kind((int[5]){1,2,3,4,5}) == 0)

yahtzee.c

```
#include "yahtzee.h"
#include <stddef.h>

static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

int score_three_of_a_kind(const int dice[5])
{
    return sum_of_dice(dice);
}
```

So how do we fix this?

We need to check that we really have three of a kind.

yahtzee\_test.c

```
#include "yahtzee.h"
#include <assert.h>
#include <stdio.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);

    puts("Yahtzee tests OK");
}
```

**Fail** - Fix - Pass

Assertion failed: (score\_three\_of\_a\_kind((int[5]){1,2,3,4,5}) == 0)

yahtzee.c

```
#include "yahtzee.h"  
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (size_t die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```

yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])
```

```
{
```

```
    int sum = 0;
```

```
    for (size_t die = 0; die < 5; die++)
```

```
        sum += dice[die];
```

```
    return sum;
```

```
}
```

```
int score_three_of_a_kind(const int dice[5])
```

```
{
```

```
    return sum_of_dice(dice);
```

```
}
```



yahtzee.c

```
#include "yahtzee.h"  
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (size_t die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```



yahtzee.c

```
#include "yahtzee.h"  
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (size_t die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```

yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (size_t die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    if (have_three_of_a_kind(dice))  
        return sum_of_dice(dice);  
}
```

yahtzee.c

```
#include "yahtzee.h"
#include <stddef.h>

static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_three_of_a_kind(dice))
        return sum_of_dice(dice);
}
```



yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (size_t die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    if (have_three_of_a_kind(dice))  
        return sum_of_dice(dice);  
}
```

yahtzee.c

```
#include "yahtzee.h"
#include <stddef.h>

static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_three_of_a_kind(dice))
        return sum_of_dice(dice);
}
```



yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (size_t die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    if (have_three_of_a_kind(dice))  
        return sum_of_dice(dice);  
    return 0;  
}
```



yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (size_t die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    if (have_three_of_a_kind(dice))  
        return sum_of_dice(dice);  
    return 0;  
}
```



yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (size_t die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```



```
int score_three_of_a_kind(const int dice[5])  
{  
    if (have_three_of_a_kind(dice))  
        return sum_of_dice(dice);  
    return 0;  
}
```



yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (size_t die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```



```
int score_three_of_a_kind(const int dice[5])  
{  
    if (have_three_of_a_kind(dice))  
        return sum_of_dice(dice);  
    return 0;  
}
```

yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}
```



```
static bool have_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
            return true;
    return false;
}
```

```
int score_three_of_a_kind(const int dice[5])
{
    if (have_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}
```

```
static bool have_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
             return true;
    return false;
}
```

```
int score_three_of_a_kind(const int dice[5])
{
    if (have_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}
```



```
static bool have_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
            return true;
    return false;
}
```

```
int score_three_of_a_kind(const int dice[5])
{
    if (have_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])
```

```
{
```

```
    int sum = 0;
```

```
    for (size_t die = 0; die < 5; die++)
```

```
        sum += dice[die];
```

```
    return sum;
```

```
}
```

```
static bool have_three_of_a_kind(const int dice[5])
```

```
{
```

```
    for (int face = 1; face <= 6; face++)
```

```
        if (count_face(face, dice) == 3)
```

```
            return true;
```

```
    return false;
```

```
}
```

```
int score_three_of_a_kind(const int dice[5])
```

```
{
```

```
    if (have_three_of_a_kind(dice))
```

```
        return sum_of_dice(dice);
```

```
    return 0;
```

```
}
```

yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}
```



```
static bool have_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
            return true;
    return false;
}
```

```
int score_three_of_a_kind(const int dice[5])
{
    if (have_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])  
{
```

```
    int sum = 0;
```

```
    for (size_t die = 0; die < 5; die++)
```

```
        sum += dice[die];
```

```
    return sum;
```

```
}
```

```
static int count_face(int face, const int dice[5])
```

```
{
```

```
    int count = 0;
```

```
    for (size_t die = 0; die < 5; die++)
```

```
        if (dice[die] == face)
```

```
            count++;
```

```
    return count;
```

```
}
```

```
static bool have_three_of_a_kind(const int dice[5])
```

```
{
```

```
    for (int face = 1; face <= 6; face++)
```

```
        if (count_face(face, dice) == 3)
```

```
            return true;
```

```
    return false;
```

```
}
```

```
int score_three_of_a_kind(const int dice[5])
```

```
{
```

```
    if (have_three_of_a_kind(dice))
```

```
        return sum_of_dice(dice);
```

```
    return 0;
```

```
}
```



yahtzee.c

```
#include "yahtzee.h"
```

```
#include <stddef.h>
```

```
static int sum_of_dice(const int dice[5])  
{
```

```
    int sum = 0;
```

```
    for (size_t die = 0; die < 5; die++)
```

```
        sum += dice[die];
```

```
    return sum;
```

```
}
```

```
static int count_face(int face, const int dice[5])
```

```
{
```

```
    int count = 0;
```

```
    for (size_t die = 0; die < 5; die++)
```

```
        if (dice[die] == face)
```

```
            count++;
```

```
    return count;
```

```
}
```

```
static bool have_three_of_a_kind(const int dice[5])
```

```
{
```

```
    for (int face = 1; face <= 6; face++)
```

```
        if (count_face(face, dice) == 3)
```

```
            return true;
```

```
    return false;
```

```
}
```

```
int score_three_of_a_kind(const int dice[5])
```

```
{
```

```
    if (have_three_of_a_kind(dice))
```

```
        return sum_of_dice(dice);
```

```
    return 0;
```

```
}
```



yahtzee.c

#include "yahtzee.h"

#include <stddef.h>

static int sum\_of\_dice(const int dice[5])  
{

int sum = 0;

for (size\_t die = 0; die < 5; die++)

sum += dice[die];

return sum;

}

static int count\_face(int face, const int dice[5])

{

int count = 0;

for (size\_t die = 0; die < 5; die++)

if (dice[die] == face)

count++;

return count;

}

static bool have\_three\_of\_a\_kind(const int dice[5])

{

for (int face = 1; face <= 6; face++)

if (count\_face(face, dice) == 3)

return true;

return false;

}

int score\_three\_of\_a\_kind(const int dice[5])

{

if (have\_three\_of\_a\_kind(dice))

return sum\_of\_dice(dice);

return 0;

}

yahtzee.c

#include "yahtzee.h"

#include <stddef.h>

static int sum\_of\_dice(const int dice[5])

{

int sum = 0;

for (size\_t die = 0; die < 5; die++)

sum += dice[die];

return sum;

}

static int count\_face(int face, const int dice[5])

{

int count = 0;

for (size\_t die = 0; die < 5; die++)

if (dice[die] == face)

count++;

return count;

}

static bool have\_three\_of\_a\_kind(const int dice[5])

{

for (int face = 1; face <= 6; face++)

if (count\_face(face, dice) == 3)

return true;

return false;

}

int score\_three\_of\_a\_kind(const int dice[5])

{

if (have\_three\_of\_a\_kind(dice))

return sum\_of\_dice(dice);

return 0;

}

yahtzee.c



```
#include "yahtzee.h"  
#include <stddef.h>  
#include <stdbool.h>
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (size_t die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
static int count_face(int face, const int dice[5])  
{  
    int count = 0;  
    for (size_t die = 0; die < 5; die++)  
        if (dice[die] == face)  
            count++;  
    return count;  
}
```

```
static bool have_three_of_a_kind(const int dice[5])  
{  
    for (int face = 1; face <= 6; face++)  
        if (count_face(face, dice) == 3)  
            return true;  
    return false;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    if (have_three_of_a_kind(dice))  
        return sum_of_dice(dice);  
    return 0;  
}
```

yahtzee.c

```
#include "yahtzee.h"
#include <stddef.h>
#include <stdbool.h>
```

```
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}
```

```
static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}
```

```
static bool have_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
            return true;
    return false;
}
```

```
int score_three_of_a_kind(const int dice[5])
{
    if (have_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

yahtzee.c

```
#include "yahtzee.h"  
#include <stddef.h>  
#include <stdbool.h>
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (size_t die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
static int count_face(int face, const int dice[5])  
{  
    int count = 0;  
    for (size_t die = 0; die < 5; die++)  
        if (dice[die] == face)  
            count++;  
    return count;  
}
```

```
static bool have_three_of_a_kind(const int dice[5])  
{  
    for (int face = 1; face <= 6; face++)  
        if (count_face(face, dice) == 3)  
            return true;  
    return false;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    if (have_three_of_a_kind(dice))  
        return sum_of_dice(dice);  
    return 0;  
}
```

Yahtzee tests OK

yahtzee.c

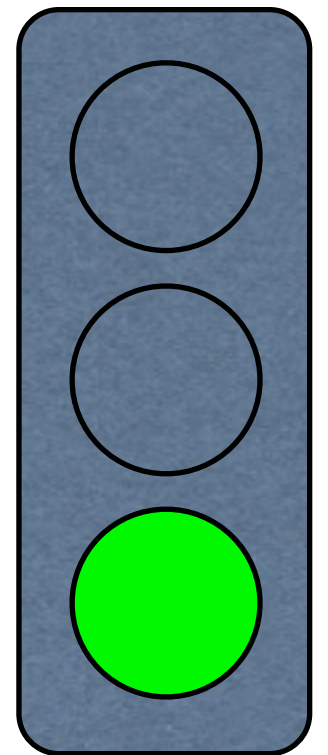
```
#include "yahtzee.h"  
#include <stddef.h>  
#include <stdbool.h>
```

```
static int sum_of_dice(const int dice[5])  
{  
    int sum = 0;  
    for (size_t die = 0; die < 5; die++)  
        sum += dice[die];  
    return sum;  
}
```

```
static int count_face(int face, const int dice[5])  
{  
    int count = 0;  
    for (size_t die = 0; die < 5; die++)  
        if (dice[die] == face)  
            count++;  
    return count;  
}
```

```
static bool have_three_of_a_kind(const int dice[5])  
{  
    for (int face = 1; face <= 6; face++)  
        if (count_face(face, dice) == 3)  
            return true;  
    return false;  
}
```

```
int score_three_of_a_kind(const int dice[5])  
{  
    if (have_three_of_a_kind(dice))  
        return sum_of_dice(dice);  
    return 0;  
}
```



Yahtzee tests OK

yahtzee\_test.c

```
...  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
```



yahtzee\_test.c

...

```
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
```





yahtzee\_test.c

...

```
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
```

yahtzee\_test.c

```
...  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
```

Yahtzee tests OK



yahtzee\_test.c

...

```
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
```



Yahtzee tests OK



yahtzee\_test.c

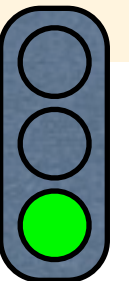
...

```
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
```

yahtzee\_test.c

```
...  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
```

Yahtzee tests OK

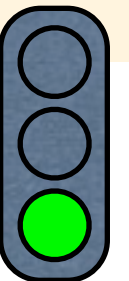


Looking good! Looking good!

yahtzee\_test.c

```
...  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
```

Yahtzee tests OK

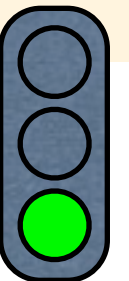


Looking good! Looking good!

yahtzee\_test.c

```
...  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
```

Yahtzee tests OK



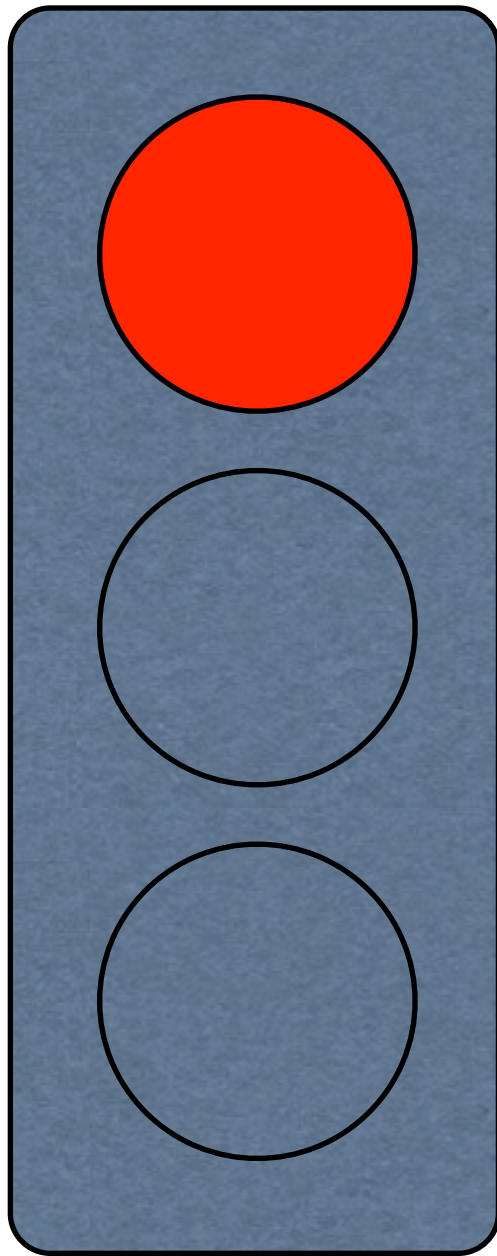


Looking good! Looking good!

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);
```



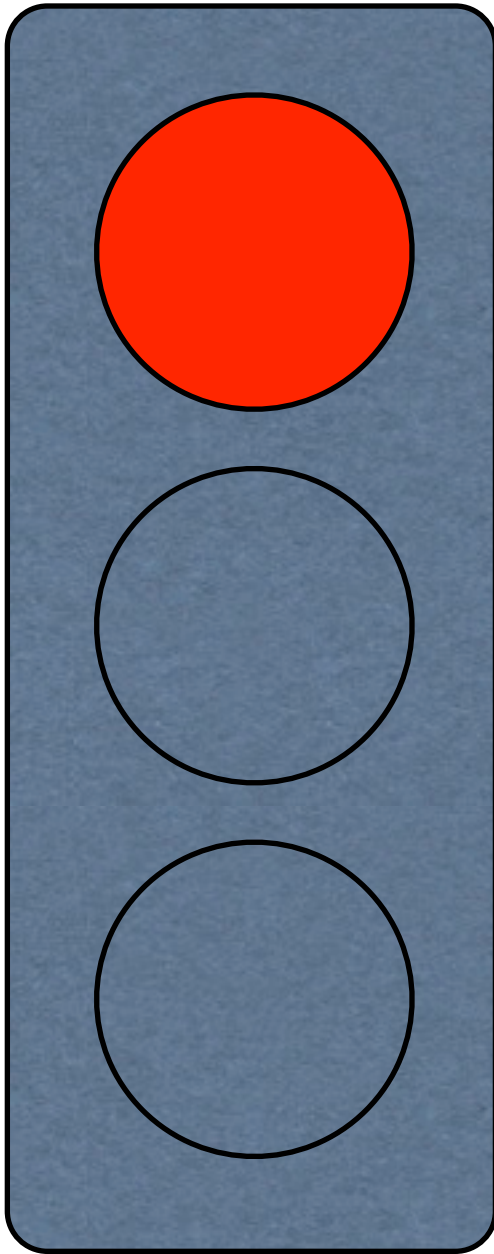


Looking good! Looking good!

yahtzee\_test.c

```
...  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);  
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);
```

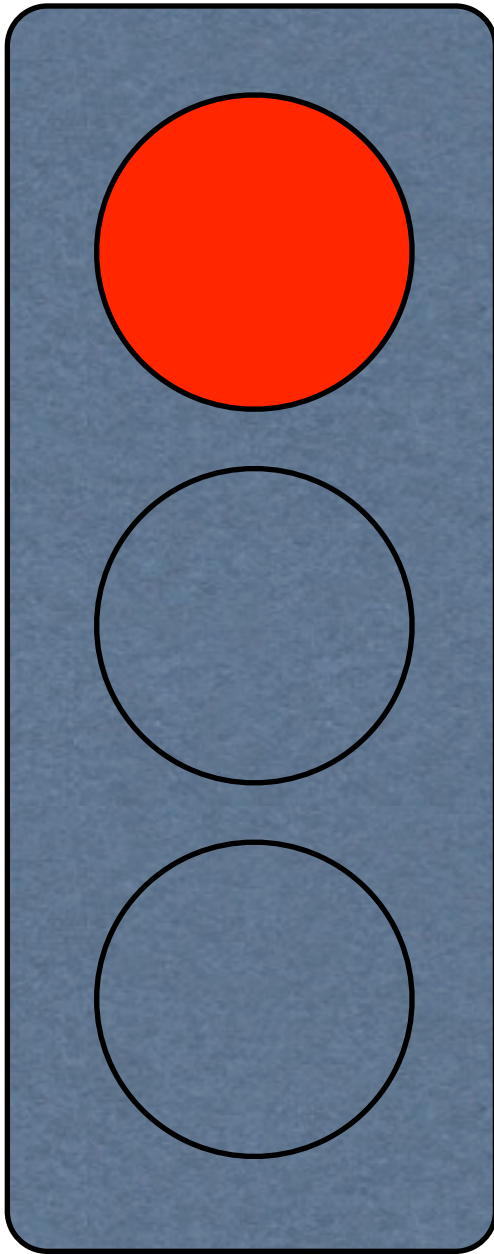
Assertion failed: (score\_three\_of\_a\_kind((int[5]){6,1,6,6,6}) == 18+7)



yahtzee\_test.c

```
...  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);  
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);
```

Assertion failed: (score\_three\_of\_a\_kind((int[5]){6,1,6,6,6}) == 18+7)



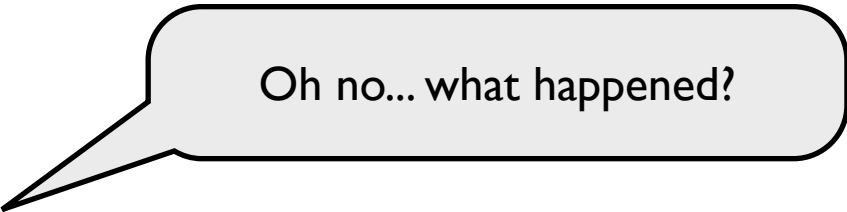
yahtzee\_test.c

...

```
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);  
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);
```

Oh no... what happened?

Assertion failed: (score\_three\_of\_a\_kind((int[5]){6,1,6,6,6}) == 18+7)



Oh no... what happened?

```
Assertion failed: (score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7)
```

yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

Oh no... what happened?

Assertion failed: (score\_three\_of\_a\_kind((int[5]){6,1,6,6,6}) == 18+7)

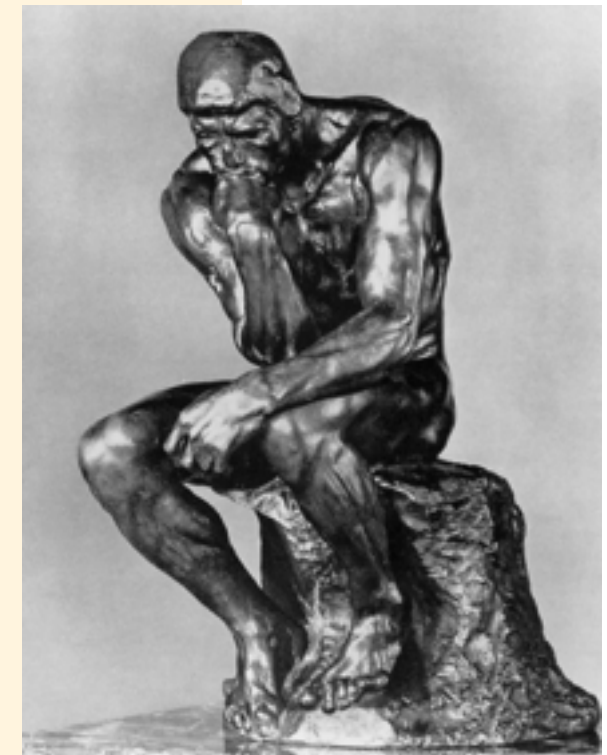
yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```



Oh no... what happened?

Assertion failed: (score\_three\_of\_a\_kind((int[5]){6,1,6,6,6}) == 18+7)

yahtzee.c

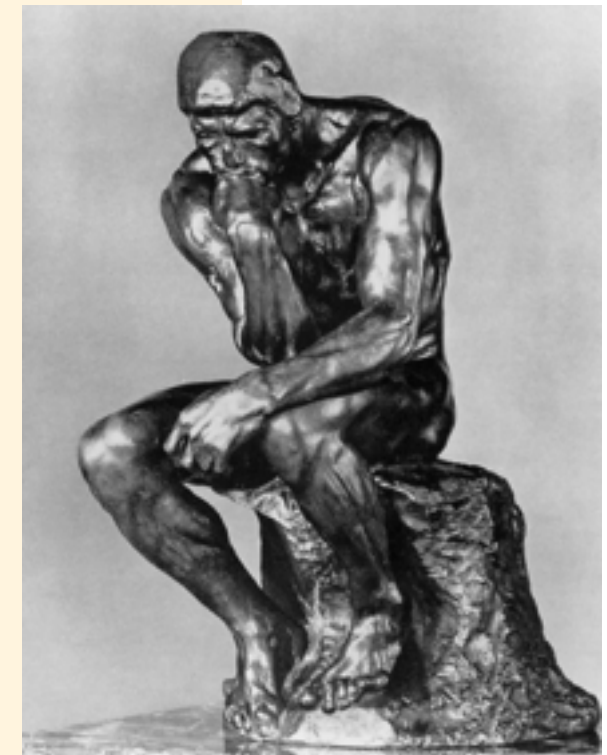
```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

But of course...



Oh no... what happened?

Assertion failed: (score\_three\_of\_a\_kind((int[5]){6,1,6,6,6}) == 18+7)



yahtzee.c

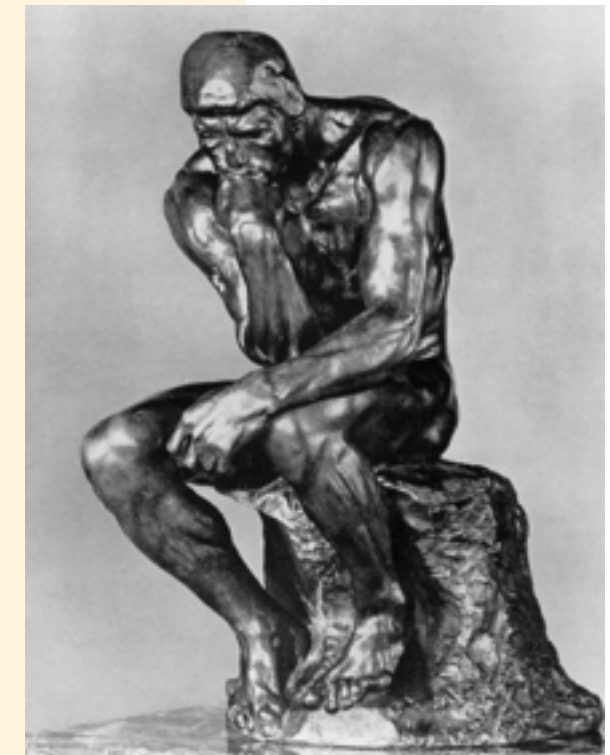
```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

But of course...



Oh no... what happened?

Assertion failed: (score\_three\_of\_a\_kind((int[5]){6,1,6,6,6}) == 18+7)



yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

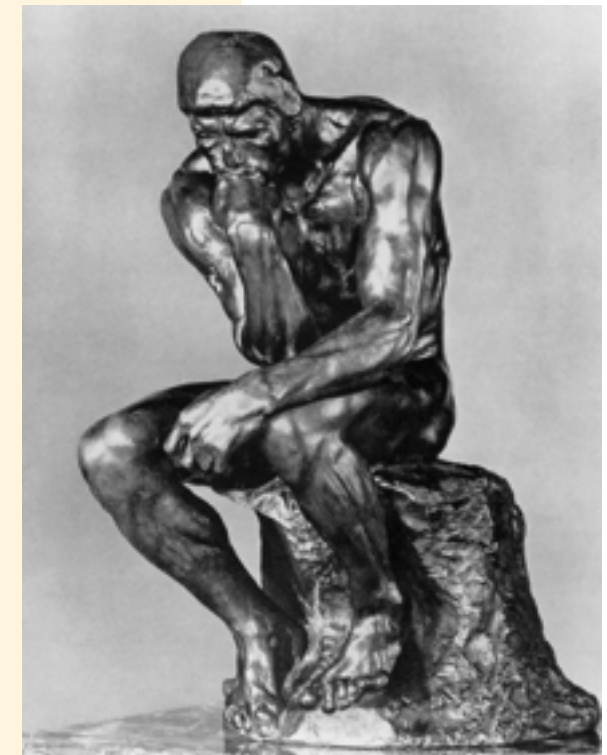
static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

But of course...

it should be **at least** three...



Oh no... what happened?

Assertion failed: (score\_three\_of\_a\_kind((int[5]){6,1,6,6,6}) == 18+7)

yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```



yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_at_least_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_at_least_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_at_least_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_at_least_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_at_least_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_at_least_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```



yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_at_least_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) >= 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_at_least_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```



yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_at_least_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) >= 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_at_least_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

Yahtzee tests OK

yahtzee.c

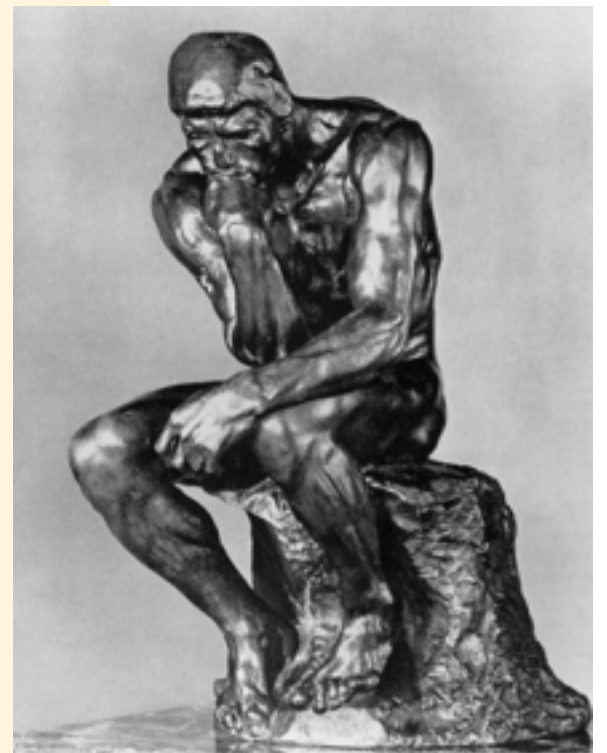
```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_at_least_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) >= 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_at_least_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

phew!



Yahtzee tests OK

yahtzee\_test.c

...

```
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);  
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);
```

Now we have a  
decent  
implementation of  
three of a kind

yahtzee\_test.c

```
...  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);  
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);
```

## LOWER SECTION

|                                               |                              |
|-----------------------------------------------|------------------------------|
| 3 of a kind                                   | Add Total<br>Of All Dice     |
| 4 of a kind                                   | Add Total<br>Of All Dice     |
| Full House                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small> | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small> | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small>        | SCORE 50                     |
| Chance                                        | Score Total<br>Of All 5 Dice |



## LOWER SECTION

|                                                                                               |                              |
|-----------------------------------------------------------------------------------------------|------------------------------|
| 3 of a kind  | Add Total<br>Of All Dice     |
| 4 of a kind                                                                                   | Add Total<br>Of All Dice     |
| Full House                                                                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small>                                                 | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small>                                                 | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small>                                                        | SCORE 50                     |
| Chance                                                                                        | Score Total<br>Of All 5 Dice |





## LOWER SECTION

3 of a kind ✓

Add Total  
Of All Dice

4 of a kind

Add Total  
Of All Dice

Full House

SCORE 25

Sm. Straight Sequence  
of 4

SCORE 30

Lg. Straight Sequence  
of 5

SCORE 40

YAHTZEE 5 of  
a kind

SCORE 50

Chance

Score Total  
Of All 5 Dice



## LOWER SECTION

3 of a kind ✓

Add Total  
Of All Dice

4 of a kind

Add Total  
Of All Dice

Full House

SCORE 25

Sm. Straight Sequence  
of 4

SCORE 30

Lg. Straight Sequence  
of 5

SCORE 40

YAHTZEE 5 of  
a kind

SCORE 50

Chance

Score Total  
Of All 5 Dice





yahtzee\_test.c

```
...  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);  
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);
```

yahtzee\_test.c

...

```
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);  
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);
```



yahtzee\_test.c

```
...  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);  
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);  
  
    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

...

```
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);  
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);  
  
    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```

yahtzee.h



```
int score_three_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

...

```
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);  
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);  
  
assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);  
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);  
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);  
  
    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return ??
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```



yahtzee.c

```
int score_four_of_a_kind(const int dice[5])  
{  
    return ??  
}
```

what should we return?

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);  
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);  
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);  
  
    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```



yahtzee.c

```
int score_four_of_a_kind(const int dice[5])  
{  
    return ??  
}
```

what should we return?

Remember fail-fix-pass? First we want a failing test

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);  
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);  
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);  
  
    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 0;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 0;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```

Assertion failed: (score\_four\_of\_a\_kind((int[5]){1,1,1,1,5}) == 9)

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 0;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```

**Fail** - Fix - Pass

Assertion failed: (score\_four\_of\_a\_kind((int[5]){1,1,1,1,5}) == 9)

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 0; 
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```

**Fail** - Fix - Pass

Assertion failed: (score\_four\_of\_a\_kind((int[5]){1,1,1,1,5}) == 9)

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```



yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```

Yahtzee tests OK



yahtzee.c

```
int score_four_of_a_kind(const int dice[5])  
{  
    return 9;  
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);  
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...  
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);  
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);  
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);  
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);  
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);  
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);  
  
    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```

Fail - Fix - **Pass**

Yahtzee tests OK

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
```



yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```



yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{

}
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    if (have_at_least_fou

}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```



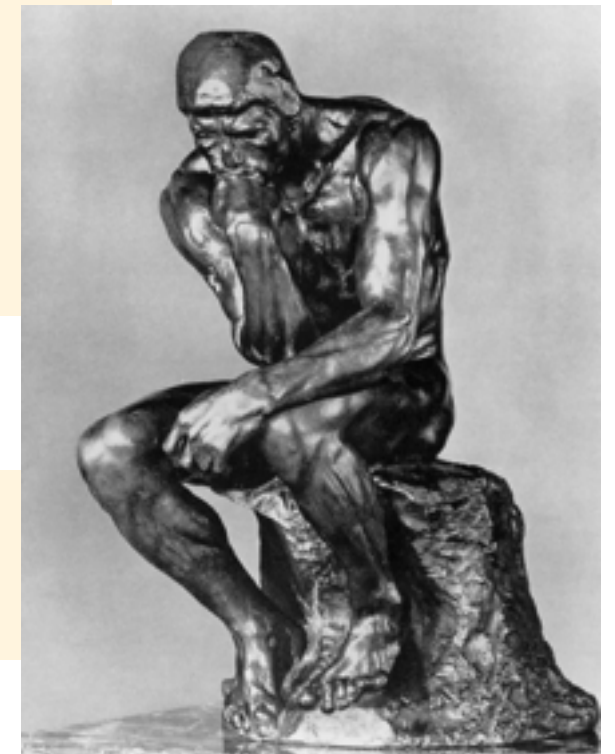
yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    if (have_at_least_fou
}
}
```

hmm...

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```



yahtzee\_test.c

```
...
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```



yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    if (have_at_least_fou
}
}
```

Perhaps we need a  
have\_at\_least\_n\_of\_a\_kind()?

hmm...

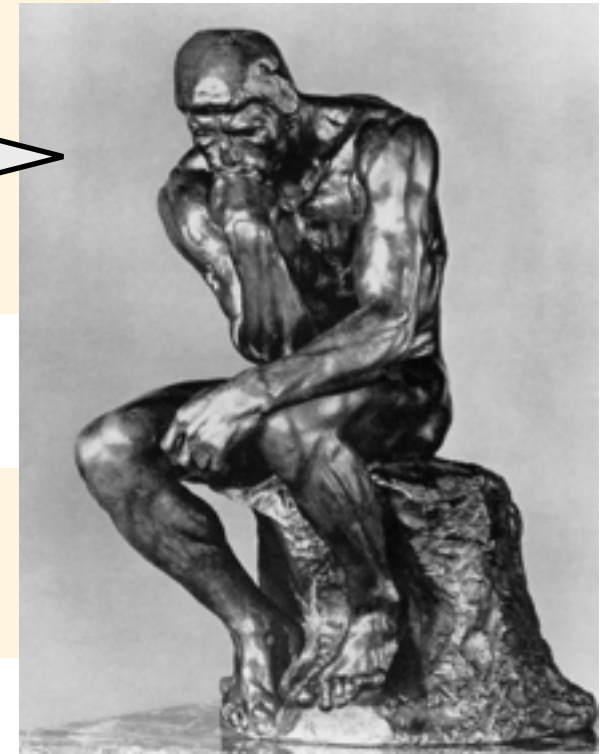
yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```



yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{

}
}
```

Perhaps we need a  
have\_at\_least\_n\_of\_a\_kind()?

hmm...

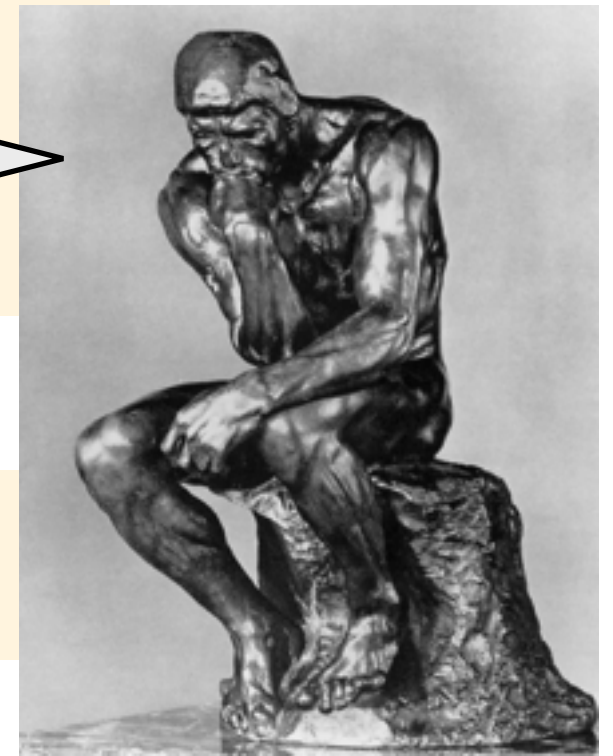
yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```



yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{

}
}
```

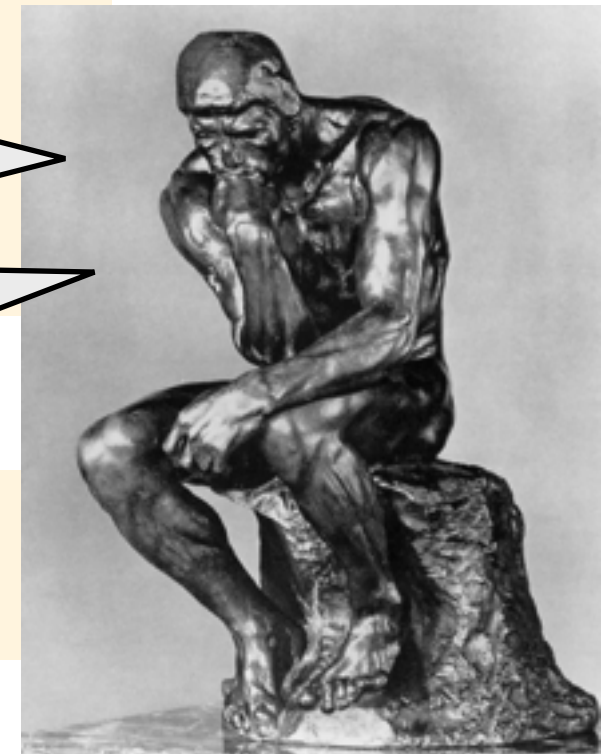
Perhaps we need a  
have\_at\_least\_n\_of\_a\_kind()?

Let's go back to "green"

hmm...

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```



yahtzee\_test.c

```
...
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{

}
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    → assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    //assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```



yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
//assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

Yahtzee tests OK



yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

And now that we are at **green** we  
can do some **refactoring**

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
//assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

Yahtzee tests OK

yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_at_least_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) >= 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_at_least_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_at_least_three_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) >= 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_at_least_three_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_at_least_n_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) >= 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_at_least_n_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```

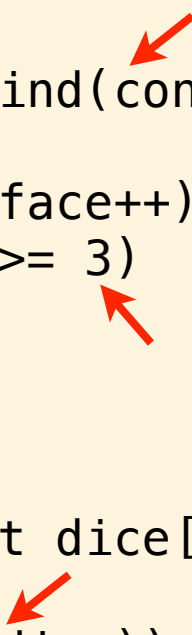
yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_at_least_n_of_a_kind(const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) >= 3)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_at_least_n_of_a_kind(dice))
        return sum_of_dice(dice);
    return 0;
}
```



yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_at_least_n_of_a_kind(int n, const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) >= n)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_at_least_n_of_a_kind(3, dice))
        return sum_of_dice(dice);
    return 0;
}
```



yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_at_least_n_of_a_kind(int n, const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) >= n)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_at_least_n_of_a_kind(3, dice))
        return sum_of_dice(dice);
    return 0;
}
```

Yahtzee tests OK



yahtzee.c

```
...
static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_at_least_n_of_a_kind(int n, const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) >= n)
            return true;
    return false;
}

int score_three_of_a_kind(const int dice[5])
{
    if (have_at_least_n_of_a_kind(3, dice))
        return sum_of_dice(dice);
    return 0;
}
```

And now we have prepared the ground for implementing score\_four\_of\_a\_kind()

Yahtzee tests OK



yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    //assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

Yahtzee tests OK

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    //assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

Yahtzee tests OK

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

Assertion failed: (score\_four\_of\_a\_kind((int[5]){1,1,1,1,5}) == 9)

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

**Fail** - Fix - Pass

Assertion failed: (score\_four\_of\_a\_kind((int[5]){1,1,1,1,5}) == 9)

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    return 9;
}
```



yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

**Fail** - Fix - Pass

Assertion failed: (score\_four\_of\_a\_kind((int[5]){1,1,1,1,5}) == 9)

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    if (have_at_least_n_of_a_kind(4, dice))
        return sum_of_dice(dice);
    return 0;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```



yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    if (have_at_least_n_of_a_kind(4, dice))
        return sum_of_dice(dice);
    return 0;
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```



yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    if (have_at_least_n_of_a_kind(4, dice))
        return sum_of_dice(dice);
    return 0;
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

Yahtzee tests OK

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    if (have_at_least_n_of_a_kind(4, dice))
        return sum_of_dice(dice);
    return 0;
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

Fail - Fix - **Pass**

Yahtzee tests OK

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    if (have_at_least_n_of_a_kind(4, dice))
        return sum_of_dice(dice);
    return 0;
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
```

Fail - Fix - **Pass**

Yahtzee tests OK

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    if (have_at_least_n_of_a_kind(4, dice))
        return sum_of_dice(dice);
    return 0;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
    assert(score_four_of_a_kind((int[5]){1,1,1,1,1}) == 5);
```

yahtzee.c

```
int score_four_of_a_kind(const int dice[5])
{
    if (have_at_least_n_of_a_kind(4, dice))
        return sum_of_dice(dice);
    return 0;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
    assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
    assert(score_four_of_a_kind((int[5]){1,1,1,1,1}) == 5);
```

Yahtzee tests OK

## LOWER SECTION

|                                                                                               |                              |
|-----------------------------------------------------------------------------------------------|------------------------------|
| 3 of a kind  | Add Total<br>Of All Dice     |
| 4 of a kind                                                                                   | Add Total<br>Of All Dice     |
| Full House                                                                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small>                                                 | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small>                                                 | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small>                                                        | SCORE 50                     |
| Chance                                                                                        | Score Total<br>Of All 5 Dice |





## LOWER SECTION

|                                                                                               |                              |
|-----------------------------------------------------------------------------------------------|------------------------------|
| 3 of a kind  | Add Total<br>Of All Dice     |
| 4 of a kind  | Add Total<br>Of All Dice     |
| Full House                                                                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small>                                                 | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small>                                                 | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small>                                                        | SCORE 50                     |
| Chance                                                                                        | Score Total<br>Of All 5 Dice |



## LOWER SECTION

3 of a kind ✓

Add Total  
Of All Dice

4 of a kind ✓

Add Total  
Of All Dice

Full House

SCORE 25

Sm. Straight Sequence  
of 4

SCORE 30

Lg. Straight Sequence  
of 5

SCORE 40

YAHTZEE 5 of  
a kind

SCORE 50

Chance

Score Total  
Of All 5 Dice





## LOWER SECTION

3 of a kind ✓

Add Total  
Of All Dice

4 of a kind ✓

Add Total  
Of All Dice

Full House

SCORE 25

Sm. Straight Sequence  
of 4

SCORE 30

Lg. Straight Sequence  
of 5

SCORE 40

YAHTZEE 5 of  
a kind

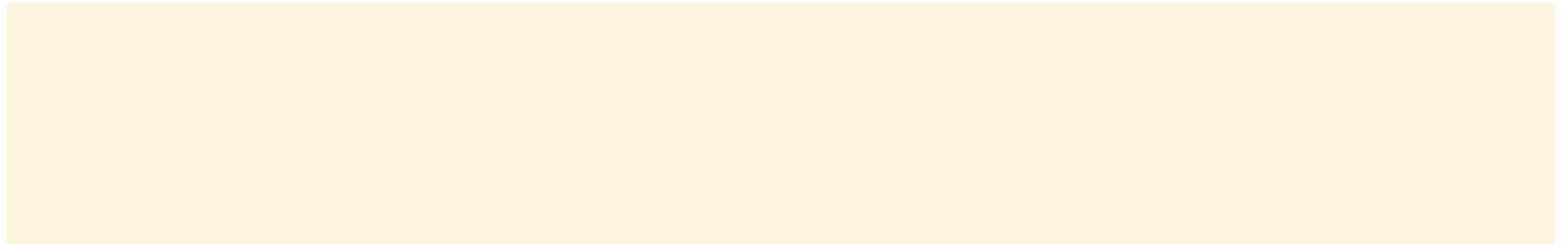
SCORE 50

Chance

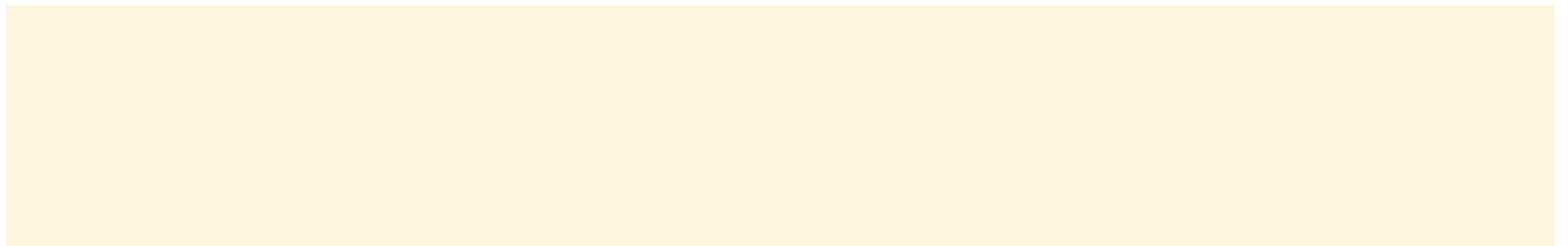
Score Total  
Of All 5 Dice



yahtzee\_test.c



yahtzee\_test.c



yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);  
int score_four_of_a_kind(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);  
int score_four_of_a_kind(const int dice[5]);
```



yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);  
int score_four_of_a_kind(const int dice[5]);  
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
```



yahtzee.c

```
int score_full_house(const int dice[5])
{
    return 0;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
```



yahtzee.c

```
int score_full_house(const int dice[5])
{
    return 0;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
```

Assertion failed: (score\_full\_house((int[5]){3,3,3,5,5}) == 25)

yahtzee.c

```
int score_full_house(const int dice[5])  
{  
    return 0;  
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);  
int score_four_of_a_kind(const int dice[5]);  
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
```

**Fail** - Fix - Pass

Assertion failed: (score\_full\_house((int[5]){3,3,3,5,5}) == 25)

yahtzee.c

```
int score_full_house(const int dice[5])  
{  
    return 0;  
}
```



yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);  
int score_four_of_a_kind(const int dice[5]);  
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
```

**Fail** - Fix - Pass

Assertion failed: (score\_full\_house((int[5]){3,3,3,5,5}) == 25)

yahtzee.c

```
int score_full_house(const int dice[5])
{
    return 25;
}
```

yahtzee.h

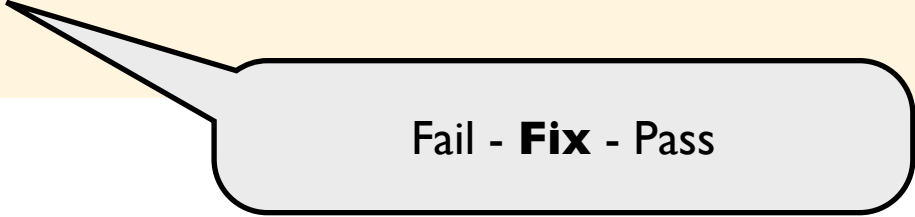
```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
```

yahtzee.c

```
int score_full_house(const int dice[5])  
{  
    return 25;  
}
```



Fail - **Fix** - Pass

yahtzee.h

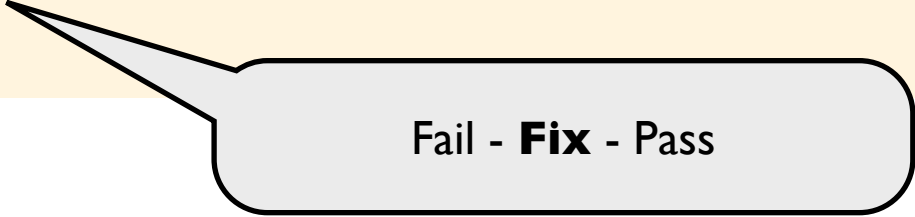
```
int score_three_of_a_kind(const int dice[5]);  
int score_four_of_a_kind(const int dice[5]);  
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
```

yahtzee.c

```
int score_full_house(const int dice[5])  
{  
    return 25;  
}
```



Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);  
int score_four_of_a_kind(const int dice[5]);  
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
```

Yahtzee tests OK

yahtzee.c

```
int score_full_house(const int dice[5])  
{  
    return 25;  
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);  
int score_four_of_a_kind(const int dice[5]);  
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
```

Fail - Fix - **Pass**

Yahtzee tests OK



yahtzee.c

```
int score_full_house(const int dice[5])  
{  
    return 25;  
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);  
int score_four_of_a_kind(const int dice[5]);  
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
```

Fail - Fix - **Pass**

Yahtzee tests OK



yahtzee.c

```
int score_full_house(const int dice[5])
{
    return 25;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
assert(score_full_house((int[5]){3,3,1,5,5}) == 0);
```

yahtzee.c

```
int score_full_house(const int dice[5])
{
    return 25;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
assert(score_full_house((int[5]){3,3,1,5,5}) == 0);
```

Assertion failed: (score\_full\_house((int[5]){3,3,1,5,5}) == 0)

yahtzee.c

```
int score_full_house(const int dice[5])  
{  
    return 25;  
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);  
int score_four_of_a_kind(const int dice[5]);  
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);  
assert(score_full_house((int[5]){3,3,1,5,5}) == 0);
```

**Fail** - Fix - Pass

Assertion failed: (score\_full\_house((int[5]){3,3,1,5,5}) == 0)

yahtzee.c

```
int score_full_house(const int dice[5])
{
    return 25;
}
```

Fix?

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
assert(score_full_house((int[5]){3,3,1,5,5}) == 0);
```

**Fail** - Fix - Pass

Assertion failed: (score\_full\_house((int[5]){3,3,1,5,5}) == 0)

yahtzee.c

```
int score_full_house(const int dice[5])
{
    return 25;
}
```

Fix?

We know how to do this!

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
assert(score_full_house((int[5]){3,3,1,5,5}) == 0);
```

**Fail** - Fix - Pass

Assertion failed: (score\_full\_house((int[5]){3,3,1,5,5}) == 0)

yahtzee.c

```
int score_full_house(const int dice[5])
{
    return 25;
}
```

Fix?

We know how to do this!

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
assert(score_full_house((int[5]){3,3,1,5,5}) == 0);
```

**Fail** - Fix - Pass

Assertion failed: (score\_full\_house((int[5]){3,3,1,5,5}) == 0)

yahtzee.c

```
static bool have_exactly_n_of_a_kind(int n, const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face,dice) == n)
            return true;
    return false;
}

int score_full_house(const int dice[5])
{
    if (have_exactly_n_of_a_kind(3, dice) &&
        have_exactly_n_of_a_kind(2, dice))
        return 25;
    return 0;
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
assert(score_full_house((int[5]){3,3,1,5,5}) == 0);
```



yahtzee.c

```
static bool have_exactly_n_of_a_kind(int n, const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face,dice) == n)
            return true;
    return false;
}

int score_full_house(const int dice[5])
{
    if (have_exactly_n_of_a_kind(3, dice) &&
        have_exactly_n_of_a_kind(2, dice))
        return 25;
    return 0;
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
assert(score_full_house((int[5]){3,3,1,5,5}) == 0);
```



yahtzee.c

```
static bool have_exactly_n_of_a_kind(int n, const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face,dice) == n)
            return true;
    return false;
}

int score_full_house(const int dice[5])
{
    if (have_exactly_n_of_a_kind(3, dice) &&
        have_exactly_n_of_a_kind(2, dice))
        return 25;
    return 0;
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
assert(score_full_house((int[5]){3,3,1,5,5}) == 0);
```

Yahtzee tests OK

yahtzee.c

```
static bool have_exactly_n_of_a_kind(int n, const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face,dice) == n)
            return true;
    return false;
}

int score_full_house(const int dice[5])
{
    if (have_exactly_n_of_a_kind(3, dice) &&
        have_exactly_n_of_a_kind(2, dice))
        return 25;
    return 0;
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
assert(score_full_house((int[5]){3,3,1,5,5}) == 0);
```

Fail - Fix - **Pass**

Yahtzee tests OK

## LOWER SECTION

|                                                                                               |                              |
|-----------------------------------------------------------------------------------------------|------------------------------|
| 3 of a kind  | Add Total<br>Of All Dice     |
| 4 of a kind  | Add Total<br>Of All Dice     |
| Full House                                                                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small>                                                 | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small>                                                 | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small>                                                        | SCORE 50                     |
| Chance                                                                                        | Score Total<br>Of All 5 Dice |



## LOWER SECTION

|                                               |                              |
|-----------------------------------------------|------------------------------|
| 3 of a kind ✓                                 | Add Total<br>Of All Dice     |
| 4 of a kind ✓                                 | Add Total<br>Of All Dice     |
| Full House ✓                                  | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small> | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small> | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small>        | SCORE 50                     |
| Chance                                        | Score Total<br>Of All 5 Dice |





## LOWER SECTION

3 of a kind ✓

Add Total  
Of All Dice

4 of a kind ✓

Add Total  
Of All Dice

Full House ✓

SCORE 25

Sm. Straight Sequence  
of 4

SCORE 30

Lg. Straight Sequence  
of 5

SCORE 40

YAHTZEE 5 of  
a kind

SCORE 50

Chance

Score Total  
Of All 5 Dice



## LOWER SECTION

|                               |                              |
|-------------------------------|------------------------------|
| 3 of a kind ✓                 | Add Total<br>Of All Dice     |
| 4 of a kind ✓                 | Add Total<br>Of All Dice     |
| Full House ✓                  | SCORE 25                     |
| Sm. Straight Sequence<br>of 4 | SCORE 30                     |
| Lg. Straight Sequence<br>of 5 | SCORE 40                     |
| YAHTZEE 5 of<br>a kind        | SCORE 50                     |
| Chance                        | Score Total<br>Of All 5 Dice |





yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```



yahtzee.c

```
int score_small_straight(const int dice[5])
{
    return 0;
}
```

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```

yahtzee.c

```
int score_small_straight(const int dice[5])
{
    return 0;
}
```

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```

Assertion failed: (score\_small\_straight((int[5]){1,2,3,4,1}) == 30)

yahtzee.c

```
int score_small_straight(const int dice[5])
{
    return 0;
}
```

It is OK to plan ahead when  
implementing a fix... so...

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```

Assertion failed: (score\_small\_straight((int[5]){1,2,3,4,1}) == 30)

yahtzee.c

```
int score_small_straight(const int dice[5])  
{  
    return 0;  
}
```



It is OK to plan ahead when  
implementing a fix... so...

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```

Assertion failed: (score\_small\_straight((int[5]){1,2,3,4,1}) == 30)

yahtzee.c

```
int score_small_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(4,dice))
        return 30;
    return 0;
}
```

It is OK to plan ahead when  
implementing a fix... so...

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```

yahtzee.c

```
int score_small_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(4,dice))
        return 30;
    return 0;
}
```

It is OK to plan ahead when  
implenting a fix... so...

we know that we need **five  
in a row** soon. So therefor  
we do **n in a row** now

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```

yahtzee.c

```
int score_small_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(4,dice))
        return 30;
    return 0;
}
```

It is OK to plan ahead when  
implenting a fix... so...

we know that we need **five  
in a row** soon. So therefor  
we do **n in a row** now

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```

Assertion failed: (score\_small\_straight((int[5]){1,2,3,4,1}) == 30)



yahtzee.c

```
int have_at_least_n_in_a_row(int n, const int dice[5])
{

}

int score_small_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(4,dice))
        return 30;
    return 0;
}
```

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```



yahtzee.c

```
int have_at_least_n_in_a_row(int n, const int dice[5])  
{
```



```
}  
  
int score_small_straight(const int dice[5])  
{  
    if (have_at_least_n_in_a_row(4,dice))  
        return 30;  
    return 0;  
}
```

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```

yahtzee.c

```
int have_at_least_n_in_a_row(int n, const int dice[5])  
{
```

?

```
}  
  
int score_small_straight(const int dice[5])  
{  
    if (have_at_least_n_in_a_row(4,dice))  
        return 30;  
    return 0;  
}
```

But how to implement this?

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```

yahtzee.c

```
int have_at_least_n_in_a_row(int n, const int dice[5])  
{
```

?

But how to implement this?

Use your head and think for a while...  
TDD is **not** about not thinking!

```
}  
  
int score_small_straight(const int dice[5])  
{  
    if (have_at_least_n_in_a_row(4,dice))  
        return 30;  
    return 0;  
}
```

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```

yahtzee.c

```
int have_at_least_n_in_a_row(int n, const int dice[5])  
{
```

?

But how to implement this?

Use your head and think for a while...  
TDD is **not** about not thinking!

```
}  
  
int score_small_straight(const int dice[5])  
{  
    if (have_at_least_n_in_a_row(4,dice))  
        return 30;  
    return 0;  
}
```

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```

Assertion failed: (score\_small\_straight((int[5]){1,2,3,4,1}) == 30)

yahtzee.c

```
int have_at_least_n_in_a_row(int n, const int dice[5])
{
    int max_seq_len = 0;
    int seq_len = 0;
    for (int face = 1; face <= 6; face++) {
        if (count_face(face,dice) == 0)
            seq_len = 0;
        else
            seq_len++;
        if (seq_len > max_seq_len)
            max_seq_len = seq_len;
    }
    return max_seq_len >= n;
}

int score_small_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(4,dice))
        return 30;
    return 0;
}
```

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```

Assertion failed: (score\_small\_straight((int[5]){1,2,3,4,1}) == 30)

yahtzee.c

```
int have_at_least_n_in_a_row(int n, const int dice[5])
{
    int max_seq_len = 0;
    int seq_len = 0;
    for (int face = 1; face <= 6; face++) {
        if (count_face(face,dice) == 0)
            seq_len = 0;
        else
            seq_len++;
        if (seq_len > max_seq_len)
            max_seq_len = seq_len;
    }
    return max_seq_len >= n;
}

int score_small_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(4,dice))
        return 30;
    return 0;
}
```

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```

**Fail** - Fix - Pass

Assertion failed: (score\_small\_straight((int[5]){1,2,3,4,1}) == 30)

yahtzee.c

```
int have_at_least_n_in_a_row(int n, const int dice[5])
{
    int max_seq_len = 0;
    int seq_len = 0;
    for (int face = 1; face <= 6; face++) {
        if (count_face(face,dice) == 0)
            seq_len = 0;
        else
            seq_len++;
        if (seq_len > max_seq_len)
            max_seq_len = seq_len;
    }
    return max_seq_len >= n;
}

int score_small_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(4,dice))
        return 30;
    return 0;
}
```

Fail - **Fix** - Pass

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```

**Fail** - Fix - Pass

Assertion failed: (score\_small\_straight((int[5]){1,2,3,4,1}) == 30)



yahtzee.c

```
int have_at_least_n_in_a_row(int n, const int dice[5])
{
    int max_seq_len = 0;
    int seq_len = 0;
    for (int face = 1; face <= 6; face++) {
        if (count_face(face,dice) == 0)
            seq_len = 0;
        else
            seq_len++;
        if (seq_len > max_seq_len)
            max_seq_len = seq_len;
    }
    return max_seq_len >= n;
}

int score_small_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(4,dice))
        return 30;
    return 0;
}
```

Fail - **Fix** - Pass

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
```

Fail - Fix - **Pass**

Yahtzee tests OK



yahtzee.c

```
int have_at_least_n_in_a_row(int n, const int dice[5])
{
    int max_seq_len = 0;
    int seq_len = 0;
    for (int face = 1; face <= 6; face++) {
        if (count_face(face,dice) == 0)
            seq_len = 0;
        else
            seq_len++;
        if (seq_len > max_seq_len)
            max_seq_len = seq_len;
    }
    return max_seq_len >= n;
}

int score_small_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(4,dice))
        return 30;
    return 0;
}
```

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
assert(score_small_straight((int[5]){1,1,1,1,1}) == 0);
```

yahtzee.c

```
int have_at_least_n_in_a_row(int n, const int dice[5])
{
    int max_seq_len = 0;
    int seq_len = 0;
    for (int face = 1; face <= 6; face++) {
        if (count_face(face,dice) == 0)
            seq_len = 0;
        else
            seq_len++;
        if (seq_len > max_seq_len)
            max_seq_len = seq_len;
    }
    return max_seq_len >= n;
}

int score_small_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(4,dice))
        return 30;
    return 0;
}
```

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
assert(score_small_straight((int[5]){1,1,1,1,1}) == 0);
```

Yahtzee tests OK

yahtzee.c

```
int have_at_least_n_in_a_row(int n, const int dice[5])
{
    int max_seq_len = 0;
    int seq_len = 0;
    for (int face = 1; face <= 6; face++) {
        if (count_face(face,dice) == 0)
            seq_len = 0;
        else
            seq_len++;
        if (seq_len > max_seq_len)
            max_seq_len = seq_len;
    }
    return max_seq_len >= n;
}

int score_small_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(4,dice))
        return 30;
    return 0;
}
```

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
assert(score_small_straight((int[5]){1,1,1,1,1}) == 0);
assert(score_small_straight((int[5]){6,5,4,3,2}) == 30);
```

yahtzee.c

```
int have_at_least_n_in_a_row(int n, const int dice[5])
{
    int max_seq_len = 0;
    int seq_len = 0;
    for (int face = 1; face <= 6; face++) {
        if (count_face(face,dice) == 0)
            seq_len = 0;
        else
            seq_len++;
        if (seq_len > max_seq_len)
            max_seq_len = seq_len;
    }
    return max_seq_len >= n;
}

int score_small_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(4,dice))
        return 30;
    return 0;
}
```

yahtzee\_test.c

```
assert(score_small_straight((int[5]){1,2,3,4,1}) == 30);
assert(score_small_straight((int[5]){1,1,1,1,1}) == 0);
assert(score_small_straight((int[5]){6,5,4,3,2}) == 30);
```

Yahtzee tests OK

## LOWER SECTION

|                                               |                              |
|-----------------------------------------------|------------------------------|
| 3 of a kind ✓                                 | Add Total<br>Of All Dice     |
| 4 of a kind ✓                                 | Add Total<br>Of All Dice     |
| Full House ✓                                  | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small> | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small> | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small>        | SCORE 50                     |
| Chance                                        | Score Total<br>Of All 5 Dice |



## LOWER SECTION

|                                                 |                              |
|-------------------------------------------------|------------------------------|
| 3 of a kind ✓                                   | Add Total<br>Of All Dice     |
| 4 of a kind ✓                                   | Add Total<br>Of All Dice     |
| Full House ✓                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small> ✓ | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small>   | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small>          | SCORE 50                     |
| Chance                                          | Score Total<br>Of All 5 Dice |





## LOWER SECTION

|                                                 |                              |
|-------------------------------------------------|------------------------------|
| 3 of a kind ✓                                   | Add Total<br>Of All Dice     |
| 4 of a kind ✓                                   | Add Total<br>Of All Dice     |
| Full House ✓                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small> ✓ | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small>   | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small>          | SCORE 50                     |
| Chance                                          | Score Total<br>Of All 5 Dice |



yahtzee\_test.c

```
assert(score_large_straight((int[5]){1,2,3,4,5}) == 30);
```



yahtzee.c

```
int score_large_straight(const int dice[5])
{
    return 0;
}
```

yahtzee.c

```
int score_large_straight(const int dice[5])  
{  
    return 0;  
}
```

Assertion failed: (score\_large\_straight((int[5]){1,2,3,4,5}) == 30)

yahtzee.c

```
int score_large_straight(const int dice[5])
{
    return 0;
}
```

**Fail** - Fix - Pass

Assertion failed: (score\_large\_straight((int[5]){1,2,3,4,5}) == 30)

yahtzee.c

```
int score_large_straight(const int dice[5])
{
    return 0;
}
```

yahtzee.c

```
int score_large_straight(const int dice[5])  
{  
    return 0;  
}
```

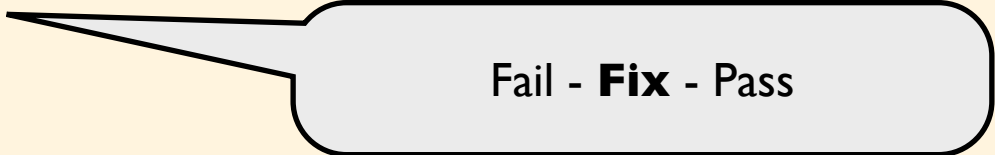


yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```



Fail - **Fix** - Pass



yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

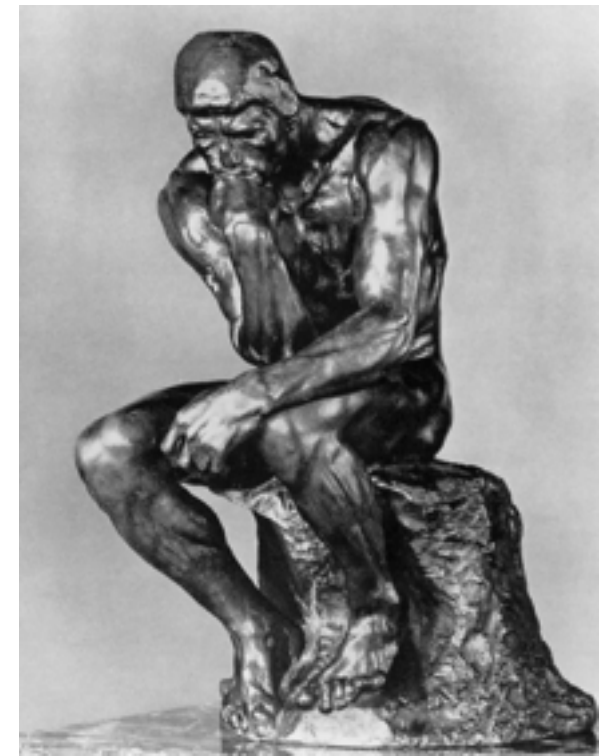
Fail - **Fix** - Pass

What?  
Inconceivable

Assertion failed: (score\_large\_straight((int[5]){1,2,3,4,5}) == 30)



hmm...



yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

Fail - **Fix** - Pass

What?  
Inconceivable

Assertion failed: (score\_large\_straight((int[5]){1,2,3,4,5}) == 30)

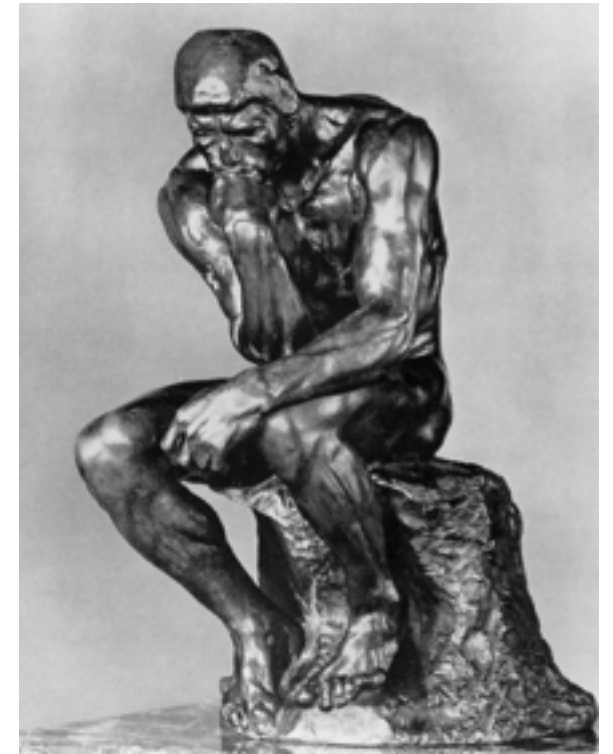
```
int have_at_least_n_in_a_row(int n, const int dice[5])
{
    int max_seq_len = 0;
    int seq_len = 0;
    for (int face = 1; face <= 6; face++) {
        if (count_face(face,dice) == 0)
            seq_len = 0;
        else
            seq_len++;
        if (seq_len > max_seq_len)
            max_seq_len = seq_len;
    }
    return max_seq_len >= n;
}
```

yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

Fail - **Fix** - Pass

hmm...



What?  
Inconceivable

Assertion failed: (score\_large\_straight((int[5]){1,2,3,4,5}) == 30)

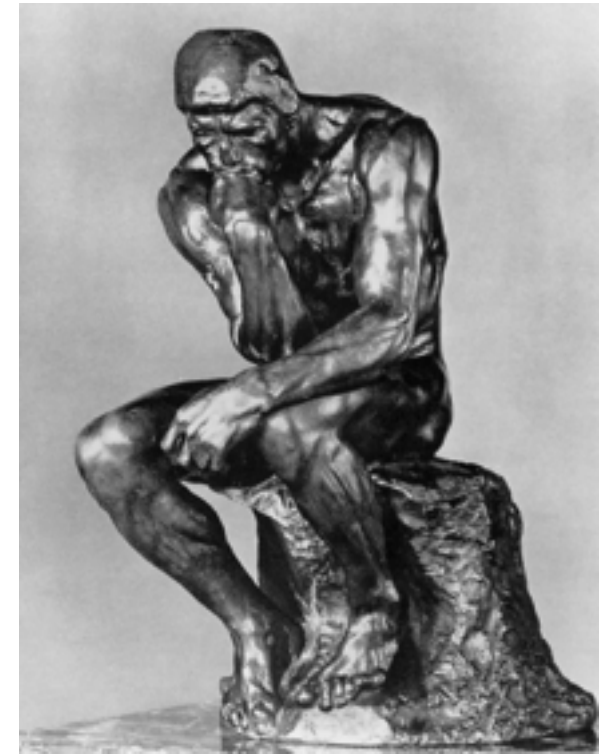
```
int have_at_least_n_in_a_row(int n, const int dice[5])
{
    int max_seq_len = 0;
    int seq_len = 0;
    for (int face = 1; face <= 6; face++) {
        if (count_face(face,dice) == 0)
            seq_len = 0;
        else
            seq_len++;
        if (seq_len > max_seq_len)
            max_seq_len = seq_len;
    }
    return max_seq_len >= n;
}
```

yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

Fail - **Fix** - Pass

hmm...



What?  
Inconceivable

Assertion failed: (score\_large\_straight((int[5]){1,2,3,4,5}) == 30)

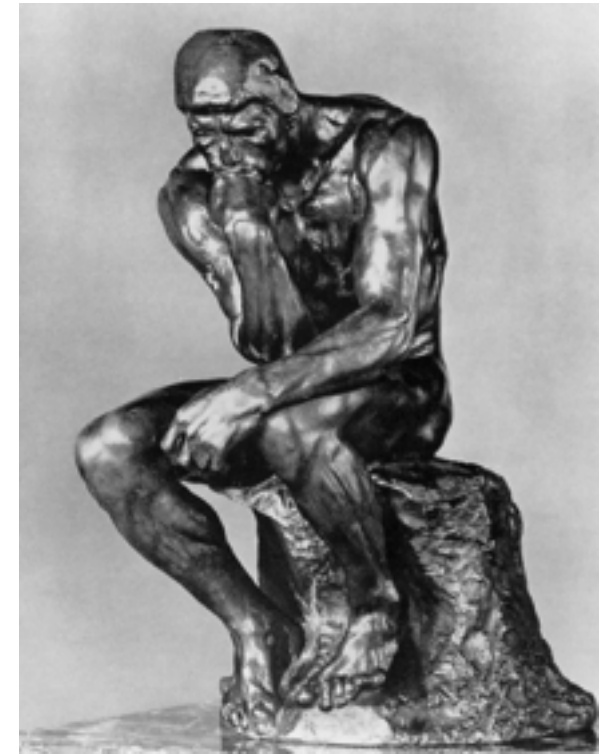
```
int have_at_least_n_in_a_row(int n, const int dice[5])
{
    int max_seq_len = 0;
    int seq_len = 0;
    for (int face = 1; face <= 6; face++) {
        if (count_face(face,dice) == 0)
            seq_len = 0;
        else
            seq_len++;
        if (seq_len > max_seq_len)
            max_seq_len = seq_len;
    }
    return max_seq_len >= n;
}
```

yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

Fail - **Fix** - Pass

hmm...



**What?**  
**Inconceivable**

yahtzee\_test.c

```
assert(score_large_straight((int[5]){1,2,3,4,5}) == 30);
```

Assertion failed: (score\_large\_straight((int[5]){1,2,3,4,5}) == 30)

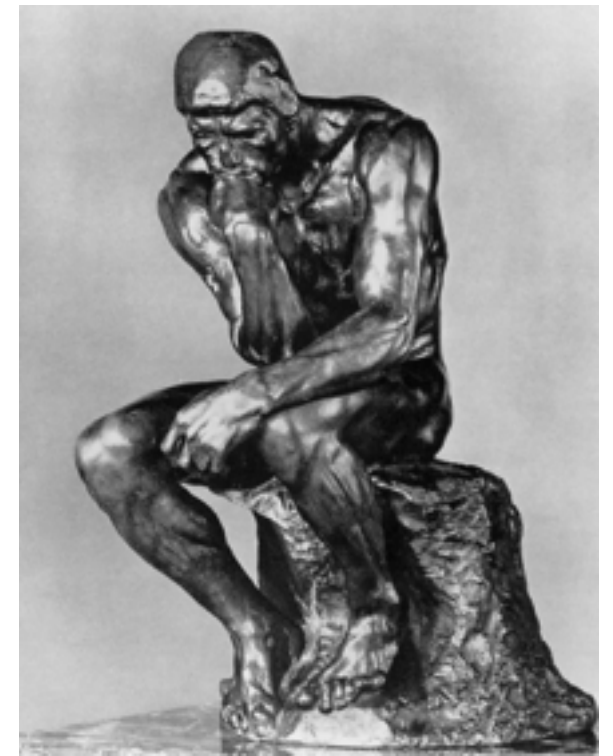
```
int have_at_least_n_in_a_row(int n, const int dice[5])
{
    int max_seq_len = 0;
    int seq_len = 0;
    for (int face = 1; face <= 6; face++) {
        if (count_face(face,dice) == 0)
            seq_len = 0;
        else
            seq_len++;
        if (seq_len > max_seq_len)
            max_seq_len = seq_len;
    }
    return max_seq_len >= n;
}
```

yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

Fail - **Fix** - Pass

hmm...



There is a bug in the test.

yahtzee\_test.c

```
assert(score_large_straight((int[5]){1,2,3,4,5}) == 30);
```

**What?**  
**Inconceivable**

Assertion failed: (score\_large\_straight((int[5]){1,2,3,4,5}) == 30)

yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

yahtzee\_test.c

```
assert(score_large_straight((int[5]){1,2,3,4,5}) == 30);
```

yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

yahtzee\_test.c

```
assert(score_large_straight((int[5]){1,2,3,4,5}) == 30);
```





yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

yahtzee\_test.c

```
assert(score_large_straight((int[5]){1,2,3,4,5}) == 40);
```



yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

Fail - **Fix** - Pass

yahtzee\_test.c

```
assert(score_large_straight((int[5]){1,2,3,4,5}) == 40);
```

yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

Fail - **Fix** - Pass

yahtzee\_test.c

```
assert(score_large_straight((int[5]){1,2,3,4,5}) == 40);
```

Fail - Fix - **Pass**

Yahtzee tests OK

yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

Fail - **Fix** - Pass

yahtzee\_test.c

```
assert(score_large_straight((int[5]){1,2,3,4,5}) == 40);
```

Fail - Fix - **Pass**

Yahtzee tests OK

yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

yahtzee\_test.c

```
assert(score_large_straight((int[5]){1,2,3,4,5}) == 40);
assert(score_large_straight((int[5]){6,6,6,6,6}) == 0);
assert(score_large_straight((int[5]){6,5,4,2,3}) == 40);
```

yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

yahtzee\_test.c

```
assert(score_large_straight((int[5]){1,2,3,4,5}) == 40);
assert(score_large_straight((int[5]){6,6,6,6,6}) == 0);
assert(score_large_straight((int[5]){6,5,4,2,3}) == 40);
```

Yahtzee tests OK

## LOWER SECTION

|                                                 |                              |
|-------------------------------------------------|------------------------------|
| 3 of a kind ✓                                   | Add Total<br>Of All Dice     |
| 4 of a kind ✓                                   | Add Total<br>Of All Dice     |
| Full House ✓                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small> ✓ | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small>   | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small>          | SCORE 50                     |
| Chance                                          | Score Total<br>Of All 5 Dice |



## LOWER SECTION

|                                                 |                              |
|-------------------------------------------------|------------------------------|
| 3 of a kind ✓                                   | Add Total<br>Of All Dice     |
| 4 of a kind ✓                                   | Add Total<br>Of All Dice     |
| Full House ✓                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small> ✓ | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small> ✓ | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small>          | SCORE 50                     |
| Chance                                          | Score Total<br>Of All 5 Dice |





## LOWER SECTION

3 of a kind ✓

Add Total  
Of All Dice

4 of a kind ✓

Add Total  
Of All Dice

Full House ✓

SCORE 25

Sm. Straight ✓ Sequence  
of 4

SCORE 30

Lg. Straight ✓ Sequence  
of 5

SCORE 40

**YAHTZEE** 5 of  
a kind

SCORE 50

Chance

Score Total  
Of All 5 Dice







yahtzee\_test.c

```
assert(score_yahtzee((int[5]){1,1,1,1,1}) == 50);  
assert(score_yahtzee((int[5]){6,6,6,6,6}) == 50);  
assert(score_yahtzee((int[5]){1,1,4,1,1}) == 0);
```

yahtzee.c

```
int score_yahtzee(const int dice[5])
{
    return 0;
}
```

yahtzee\_test.c

```
assert(score_yahtzee((int[5]){1,1,1,1,1}) == 50);
assert(score_yahtzee((int[5]){6,6,6,6,6}) == 50);
assert(score_yahtzee((int[5]){1,1,4,1,1}) == 0);
```

yahtzee.c

```
int score_yahtzee(const int dice[5])
{
    return 0;
}
```

yahtzee\_test.c

```
assert(score_yahtzee((int[5]){1,1,1,1,1}) == 50);
assert(score_yahtzee((int[5]){6,6,6,6,6}) == 50);
assert(score_yahtzee((int[5]){1,1,4,1,1}) == 0);
```

Assertion failed: (score\_yahtzee((int[5]){1,1,1,1,1}) == 50)

yahtzee.c

```
int score_yahtzee(const int dice[5])  
{  
    return 0;  
}
```

yahtzee\_test.c

```
assert(score_yahtzee((int[5]){1,1,1,1,1}) == 50);  
assert(score_yahtzee((int[5]){6,6,6,6,6}) == 50);  
assert(score_yahtzee((int[5]){1,1,4,1,1}) == 0);
```

**Fail** - Fix - Pass

Assertion failed: (score\_yahtzee((int[5]){1,1,1,1,1}) == 50)

yahtzee.c

```
int score_yahtzee(const int dice[5])
{
    return 0;
}
```



yahtzee\_test.c

```
assert(score_yahtzee((int[5]){1,1,1,1,1}) == 50);
assert(score_yahtzee((int[5]){6,6,6,6,6}) == 50);
assert(score_yahtzee((int[5]){1,1,4,1,1}) == 0);
```

**Fail** - Fix - Pass

Assertion failed: (score\_yahtzee((int[5]){1,1,1,1,1}) == 50)

yahtzee.c

```
int score_yahtzee(const int dice[5])
{
    return have_exactly_n_of_a_kind(5,dice) ? 50 : 0;
}
```

yahtzee\_test.c

```
assert(score_yahtzee((int[5]){1,1,1,1,1}) == 50);
assert(score_yahtzee((int[5]){6,6,6,6,6}) == 50);
assert(score_yahtzee((int[5]){1,1,4,1,1}) == 0);
```



yahtzee.c

```
int score_yahtzee(const int dice[5])
{
    return have_exactly_n_of_a_kind(5,dice) ? 50 : 0;
}
```

Fail - **Fix** - Pass

yahtzee\_test.c

```
assert(score_yahtzee((int[5]){1,1,1,1,1}) == 50);
assert(score_yahtzee((int[5]){6,6,6,6,6}) == 50);
assert(score_yahtzee((int[5]){1,1,4,1,1}) == 0);
```



yahtzee.c

```
int score_yahtzee(const int dice[5])
{
    return have_exactly_n_of_a_kind(5,dice) ? 50 : 0;
}
```

Fail - **Fix** - Pass

yahtzee\_test.c

```
assert(score_yahtzee((int[5]){1,1,1,1,1}) == 50);
assert(score_yahtzee((int[5]){6,6,6,6,6}) == 50);
assert(score_yahtzee((int[5]){1,1,4,1,1}) == 0);
```

Yahtzee tests OK

yahtzee.c

```
int score_yahtzee(const int dice[5])  
{  
    return have_exactly_n_of_a_kind(5,dice) ? 50 : 0;  
}
```

Fail - **Fix** - Pass

yahtzee\_test.c

```
assert(score_yahtzee((int[5]){1,1,1,1,1}) == 50);  
assert(score_yahtzee((int[5]){6,6,6,6,6}) == 50);  
assert(score_yahtzee((int[5]){1,1,4,1,1}) == 0);
```

Fail - Fix - **Pass**

Yahtzee tests OK

## LOWER SECTION

|                                                 |                              |
|-------------------------------------------------|------------------------------|
| 3 of a kind ✓                                   | Add Total<br>Of All Dice     |
| 4 of a kind ✓                                   | Add Total<br>Of All Dice     |
| Full House ✓                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small> ✓ | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small> ✓ | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small>          | SCORE 50                     |
| Chance                                          | Score Total<br>Of All 5 Dice |



## LOWER SECTION

|                                                 |                              |
|-------------------------------------------------|------------------------------|
| 3 of a kind ✓                                   | Add Total<br>Of All Dice     |
| 4 of a kind ✓                                   | Add Total<br>Of All Dice     |
| Full House ✓                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small> ✓ | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small> ✓ | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small> ✓        | SCORE 50                     |
| Chance                                          | Score Total<br>Of All 5 Dice |





## LOWER SECTION

|                                                 |                              |
|-------------------------------------------------|------------------------------|
| 3 of a kind ✓                                   | Add Total<br>Of All Dice     |
| 4 of a kind ✓                                   | Add Total<br>Of All Dice     |
| Full House ✓                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small> ✓ | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small> ✓ | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small> ✓        | SCORE 50                     |
| Chance                                          | Score Total<br>Of All 5 Dice |





yahtzee.c

```
int score_chance(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```

yahtzee.c

```
int score_chance(const int dice[5])
{
    return sum_of_dice(dice);
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
int score_small_straight(const int dice[5]);
int score_large_straight(const int dice[5]);
int score_yahtzee(const int dice[5]);
```



yahtzee.c

```
int score_chance(const int dice[5])  
{  
    return sum_of_dice(dice);  
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);  
int score_four_of_a_kind(const int dice[5]);  
int score_full_house(const int dice[5]);  
int score_small_straight(const int dice[5]);  
int score_large_straight(const int dice[5]);  
int score_yahtzee(const int dice[5]);
```



yahtzee.c

```
int score_chance(const int dice[5])
{
    return sum_of_dice(dice);
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
int score_small_straight(const int dice[5]);
int score_large_straight(const int dice[5]);
int score_yahtzee(const int dice[5]);
int score_chance(const int dice[5]);
```

yahtzee.c

```
int score_chance(const int dice[5])
{
    return sum_of_dice(dice);
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
int score_small_straight(const int dice[5]);
int score_large_straight(const int dice[5]);
int score_yahtzee(const int dice[5]);
int score_chance(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_chance((int[5]){1,1,4,1,1}) == 8);
assert(score_chance((int[5]){2,3,4,5,6}) == 20);
assert(score_chance((int[5]){6,6,6,6,6}) == 30);
```

yahtzee.c

```
int score_chance(const int dice[5])
{
    return sum_of_dice(dice);
}
```

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
int score_small_straight(const int dice[5]);
int score_large_straight(const int dice[5]);
int score_yahtzee(const int dice[5]);
int score_chance(const int dice[5]);
```

yahtzee\_test.c

```
assert(score_chance((int[5]){1,1,4,1,1}) == 8);
assert(score_chance((int[5]){2,3,4,5,6}) == 20);
assert(score_chance((int[5]){6,6,6,6,6}) == 30);
```

Yahtzee tests OK

## LOWER SECTION

|                                                 |                              |
|-------------------------------------------------|------------------------------|
| 3 of a kind ✓                                   | Add Total<br>Of All Dice     |
| 4 of a kind ✓                                   | Add Total<br>Of All Dice     |
| Full House ✓                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small> ✓ | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small> ✓ | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small> ✓        | SCORE 50                     |
| Chance                                          | Score Total<br>Of All 5 Dice |



## LOWER SECTION

|                                                 |                              |
|-------------------------------------------------|------------------------------|
| 3 of a kind ✓                                   | Add Total<br>Of All Dice     |
| 4 of a kind ✓                                   | Add Total<br>Of All Dice     |
| Full House ✓                                    | SCORE 25                     |
| Sm. Straight <small>Sequence<br/>of 4</small> ✓ | SCORE 30                     |
| Lg. Straight <small>Sequence<br/>of 5</small> ✓ | SCORE 40                     |
| YAHTZEE <small>5 of<br/>a kind</small> ✓        | SCORE 50                     |
| Chance ✓                                        | Score Total<br>Of All 5 Dice |





## yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
int score_four_of_a_kind(const int dice[5]);
int score_full_house(const int dice[5]);
int score_small_straight(const int dice[5]);
int score_large_straight(const int dice[5]);
int score_yahtzee(const int dice[5]);
int score_chance(const int dice[5]);
```

## Makefile

```
CC=gcc
CFLAGS=-std=c99 -O -Wall -Wextra -pedantic
LD=gcc

all: yahtzee.a

yahtzee.a: yahtzee.o
        ar -rcs $@ $^

check: yahtzee_tests
        ./yahtzee_tests

yahtzee.o: yahtzee.c yahtzee.h

yahtzee_tests.o: yahtzee_tests.c yahtzee.h

yahtzee_tests: yahtzee_tests.o yahtzee.a
        $(LD) -o $@ $^

clean:
        rm -f *.o *.a yahtzee_tests
```

## yahtzee\_tests.h

```
#include "yahtzee.h"
#include <stdio.h>
#include <assert.h>

int main(void)
{
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,6,4}) == 3+10);
    assert(score_three_of_a_kind((int[5]){1,1,6,1,4}) == 3+10);
    assert(score_three_of_a_kind((int[5]){1,6,1,2,4}) == 0);
    assert(score_three_of_a_kind((int[5]){6,6,6,6,6}) == 30);

    assert(score_four_of_a_kind((int[5]){1,1,1,1,5}) == 9);
    assert(score_four_of_a_kind((int[5]){1,1,3,1,5}) == 0);
    assert(score_four_of_a_kind((int[5]){1,1,1,1,1}) == 5);

    assert(score_full_house((int[5]){3,3,3,5,5}) == 25);
    assert(score_full_house((int[5]){3,3,5,5,5}) == 25);
    assert(score_full_house((int[5]){3,3,1,5,5}) == 0);

    assert(score_small_straight((int[5]){1,2,3,4,6}) == 30);
    assert(score_small_straight((int[5]){2,3,4,5,6}) == 30);
    assert(score_small_straight((int[5]){2,3,1,5,6}) == 0);
    assert(score_small_straight((int[5]){6,5,4,3,3}) == 30);

    assert(score_large_straight((int[5]){1,2,3,4,5}) == 40);
    assert(score_large_straight((int[5]){6,6,6,6,6}) == 0);
    assert(score_large_straight((int[5]){6,5,4,2,3}) == 40);

    assert(score_yahtzee((int[5]){1,1,1,1,1}) == 50);
    assert(score_yahtzee((int[5]){6,6,6,6,6}) == 50);
    assert(score_yahtzee((int[5]){1,1,4,1,1}) == 0);

    assert(score_chance((int[5]){1,1,4,1,1}) == 8);
    assert(score_chance((int[5]){2,3,4,5,6}) == 20);
    assert(score_chance((int[5]){6,6,6,6,6}) == 30);

    puts("Yahtzee tests OK");
}
```

# yahtzee.c

```
#include "yahtzee.h"
#include <stdbool.h>
#include <stddef.h>

static int sum_of_dice(const int dice[5])
{
    int sum = 0;
    for (size_t die = 0; die < 5; die++)
        sum += dice[die];
    return sum;
}

static int count_face(int face, const int dice[5])
{
    int count = 0;
    for (size_t die = 0; die < 5; die++)
        if (dice[die] == face)
            count++;
    return count;
}

static bool have_at_least_n_of_a_kind(int n, const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) >= n)
            return true;
    return false;
}

static bool have_exactly_n_of_a_kind(int n, const int dice[5])
{
    for (int face = 1; face <= 6; face++)
        if (count_face(face, dice) == n)
            return true;
    return false;
}

int have_at_least_n_in_a_row(int n, const int dice[5])
{
    int max_seq_len = 0;
    int seq_len = 0;
    for (int face = 1; face <= 6; face++) {
        if (count_face(face, dice) == 0)
            seq_len = 0;
        else
            seq_len++;
        if (seq_len > max_seq_len)
            max_seq_len = seq_len;
    }
    return max_seq_len >= n;
}
```

```
int score_three_of_a_kind(const int dice[5])
{
    if (have_at_least_n_of_a_kind(3, dice))
        return sum_of_dice(dice);
    return 0;
}

int score_four_of_a_kind(const int dice[5])
{
    if (have_at_least_n_of_a_kind(4, dice))
        return sum_of_dice(dice);
    return 0;
}

int score_full_house(const int dice[5])
{
    if (have_exactly_n_of_a_kind(3, dice) &&
        have_exactly_n_of_a_kind(2, dice))
        return 25;
    return 0;
}

int score_small_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(4, dice))
        return 30;
    return 0;
}

int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5, dice))
        return 40;
    return 0;
}

int score_yahtzee(const int dice[5])
{
    if (have_exactly_n_of_a_kind(5, dice))
        return 50;
    return 0;
}

int score_chance(const int dice[5])
{
    return sum_of_dice(dice);
}
```



# Exercise: simple phone book library

Your task is to write a very simple library for keeping a list of names and numbers (phonebook.a), where you can add and remove entries, and it should also be possible to print all the entries. The API to the library should be declared in a header file (phonebook.h)

Write proper unit tests (phonebook\_test.c), that checks that the phone book library is working correctly.

Use make to automate the build.

You may pretend like this library is for a very simple mobile phone. For the initial requirements you can assume that all numbers can be represented as a positive integer with 3 to 8 digits, ie no support for international calls. Names are limited to 15 characters. It is OK to assume max 30 phone book entries.

Please do not over-engineer the solution.

Work in pairs! When you are done, email your answer to me.

# Summary

# Course Objectives

To learn something about:

- Tools and vocabulary
- Writing correct C code
- Undefined and unspecified behavior
- The history and spirit of C
- The memory model
- Modern development techniques

# The spirit of C

## **trust the programmer**

- let them do what needs to be done
- the programmer is in charge not the compiler

## **keep the language small and simple**

- small amount of code → small amount of assembler
- provide only one way to do an operation
- new inventions are not entertained

## **make it fast, even if its not portable**

- target efficient code generation
- int preference, int promotion rules
- sequence points, maximum leeway to compiler

## **rich expression support**

- lots of operators
- expressions combine into larger expressions

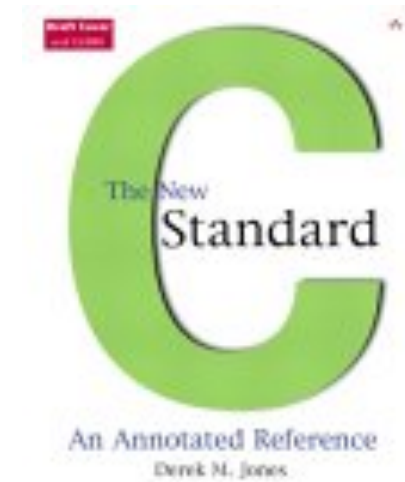
# recommended reading



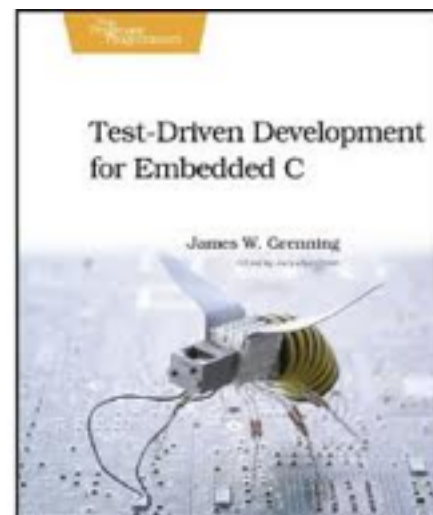
ISO/IEC 9899:1999



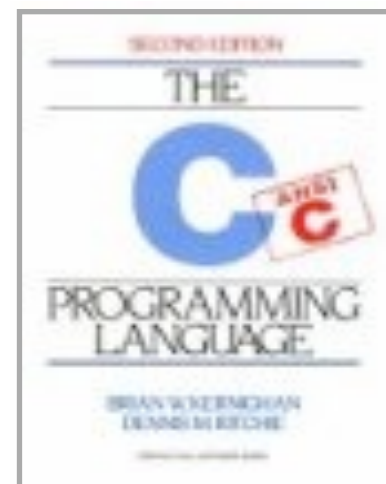
Peter van der Linden



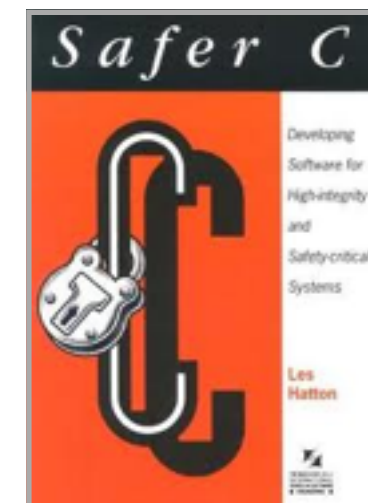
Derek M. Jones



James Grenning



Dennis Ritchie



Les Hatton

!