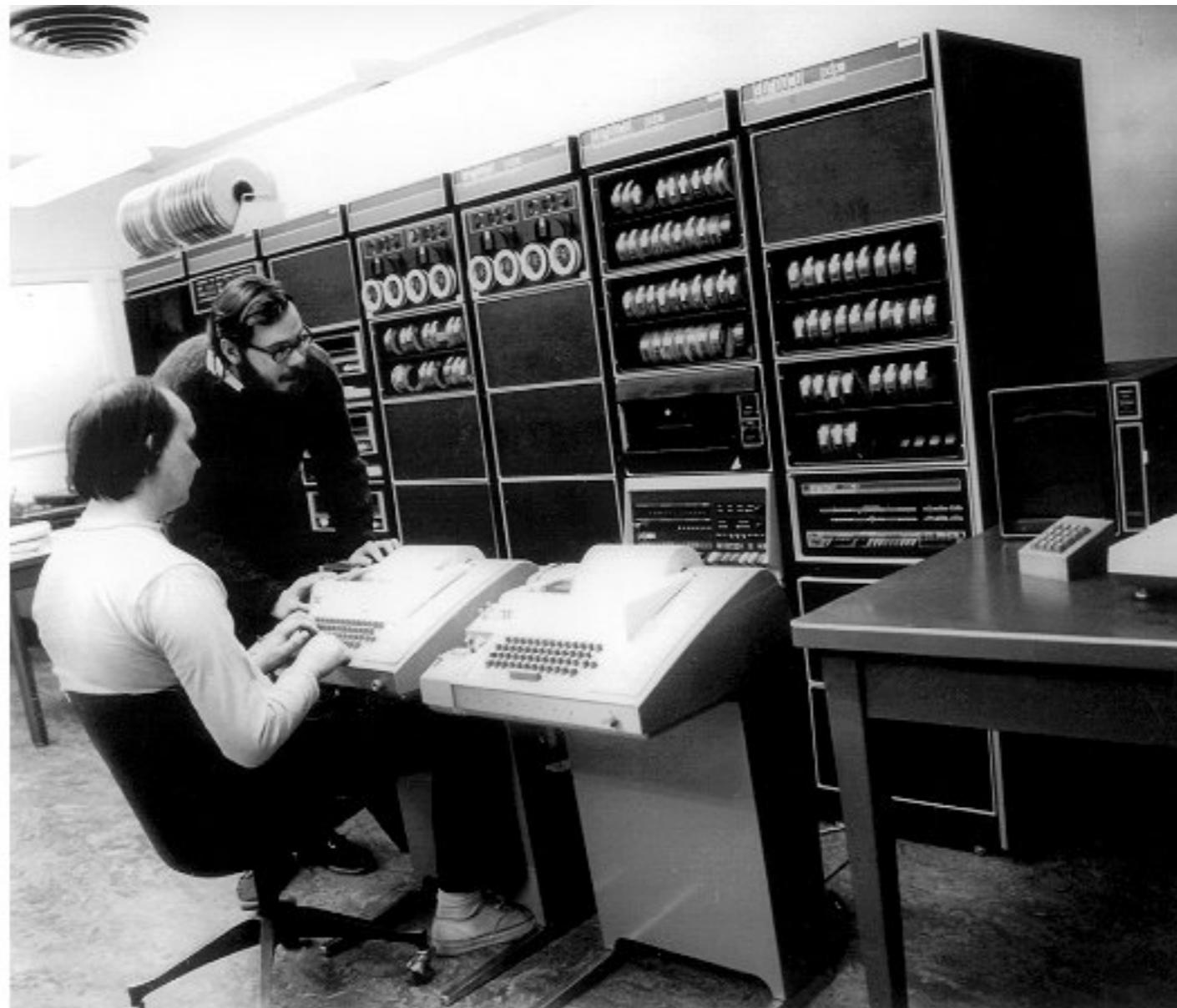


# Deep C - a 3 day course

“to get a deeper understanding of the language”



by  
Jon Jagger & Olve Maudal  
(March 28, 2012)

This 3-day Deep C course is for those who have been programming in C for a while but would benefit from starting from the beginning and relearn the basics to get a deeper understanding of the language.

The course is based on a 2-day course, “C Foundation”, developed by Jon Jagger and Olve Maudal, but in this version the idea is to relax the pace of the course and do more hands-on exercises and have more time for questions and answers. We might also skip some more advanced and esoteric topics. Target group for this version of the course is new hires from university and also engineers that are experts in other subjects and that have been writing code in C just once in a while.

We assume that the students are familiar with working in Unix like environments.

Unless noted otherwise, all the examples and text refers to the C99 standard.

# Course Outline

Day 1: 0930-1500

A Brief Tour

Lunch

Building blocks

Functions and Program Structure

Exercise (calc\_stat)

Day 2: 0900-1500

Exercise walk through (calc\_stat)

Test-Driven Development

Types, Operators and Expressions

Lunch

Control Flow

Pointers and Arrays

Structures

Exercise (phone\_book)

Day 3: 0900-1500

Exercise (phone\_book)

Preprocessor

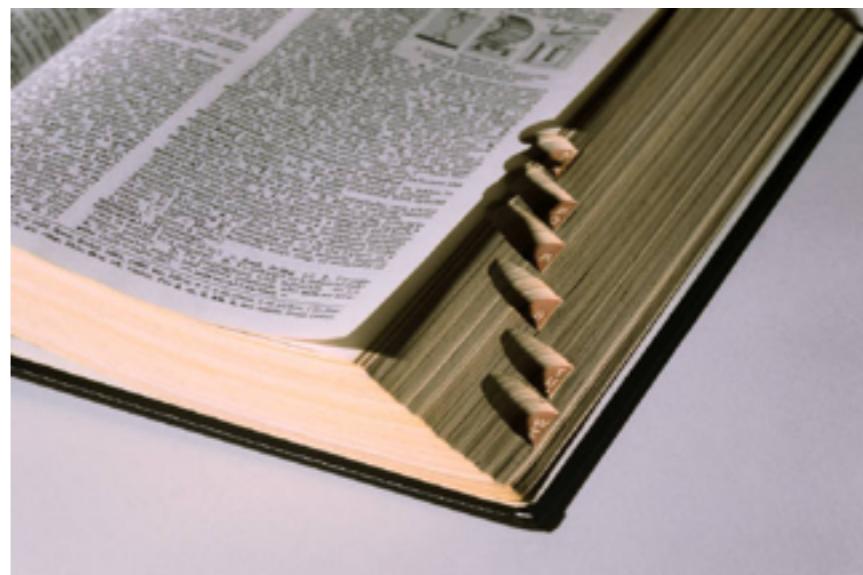
Lunch

Advanced Techniques

Summary

# A Brief Tour

## tools and vocabulary

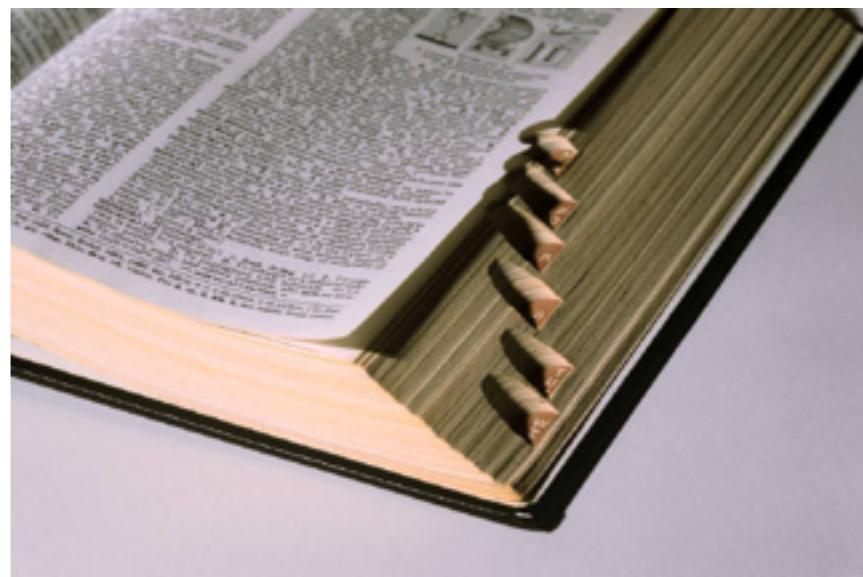


Deep C - a 3 day course  
Jon Jagger & Olve Maudal  
(March 27, 2012)

*not so*

# A Brief Tour

## tools and vocabulary



Deep C - a 3 day course  
Jon Jagger & Olve Maudal  
(March 27, 2012)

# Exercise: Hello World!

Type in this code. Compile and execute the program. What do you get?

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);
}
```

# Exercise: Hello World!

Type in this code. Compile and execute the program. What do you get?

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
```

# Exercise: Hello World!

Type in this code. Compile and execute the program. What do you get?

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
```

# Exercise: Hello World!

Type in this code. Compile and execute the program. What do you get?

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
```

# Exercise: Hello World!

Type in this code. Compile and execute the program. What do you get?

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 $?
```

# Exercise: Hello World!

Type in this code. Compile and execute the program. What do you get?

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
```

# Exercise: Hello World!

Type in this code. Compile and execute the program. What do you get?

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);
}
```

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(    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);
}
```

guess which argument is pushed first?

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);
}
```

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...

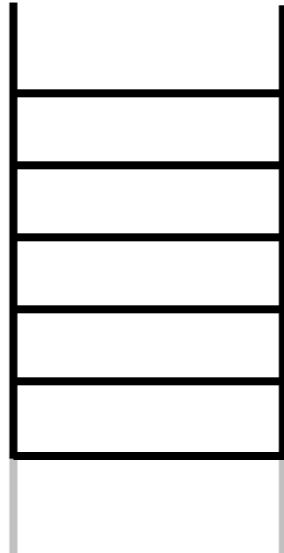
```
#include <stdio.h>

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...



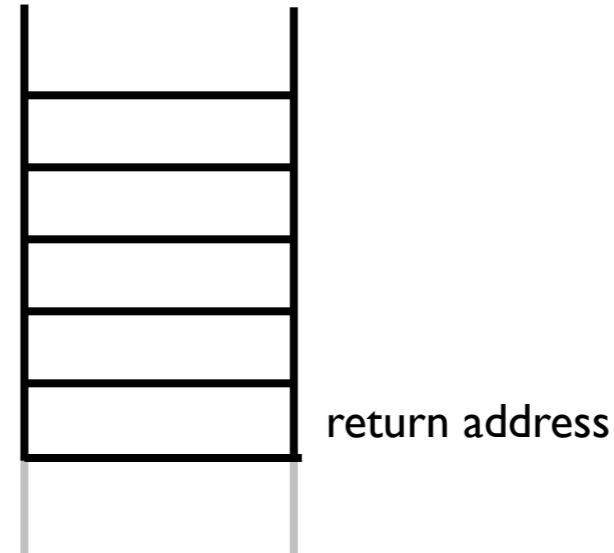
```
#include <stdio.h>

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...



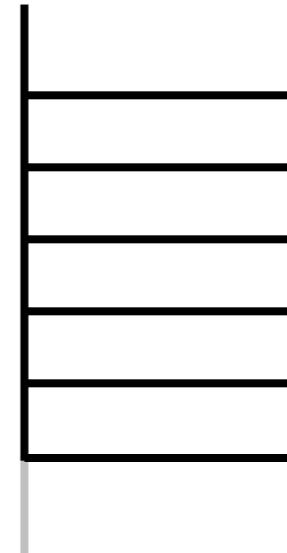
```
#include <stdio.h>

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...



space for return value  
return address

```

#include <stdio.h>

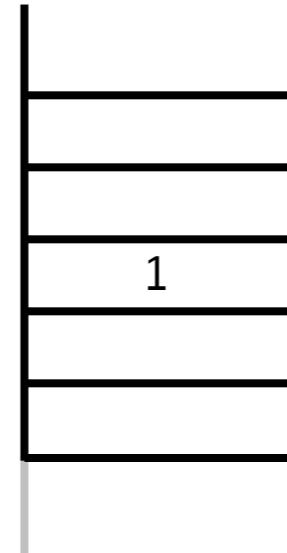
static int calc(int a, int b, int c)
{
    return a * b / c;
}

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...



first argument - c  
space for return value  
return address

```

#include <stdio.h>

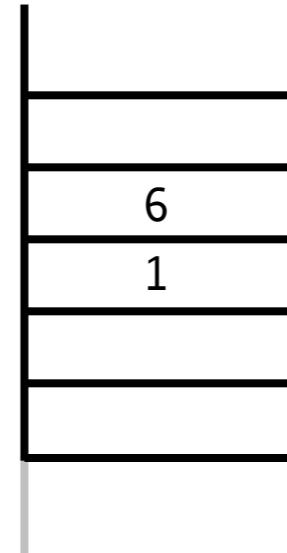
static int calc(int a, int b, int c)
{
    return a * b / c;
}

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...



second argument - b  
 first argument - c  
 space for return value  
 return address

```

#include <stdio.h>

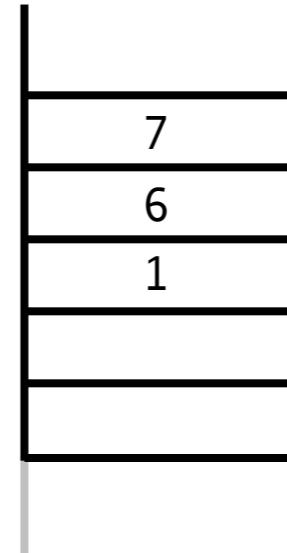
static int calc(int a, int b, int c)
{
    return a * b / c;
}

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...



third argument - c  
 second argument - b  
 first argument - c  
 space for return value  
 return address

```

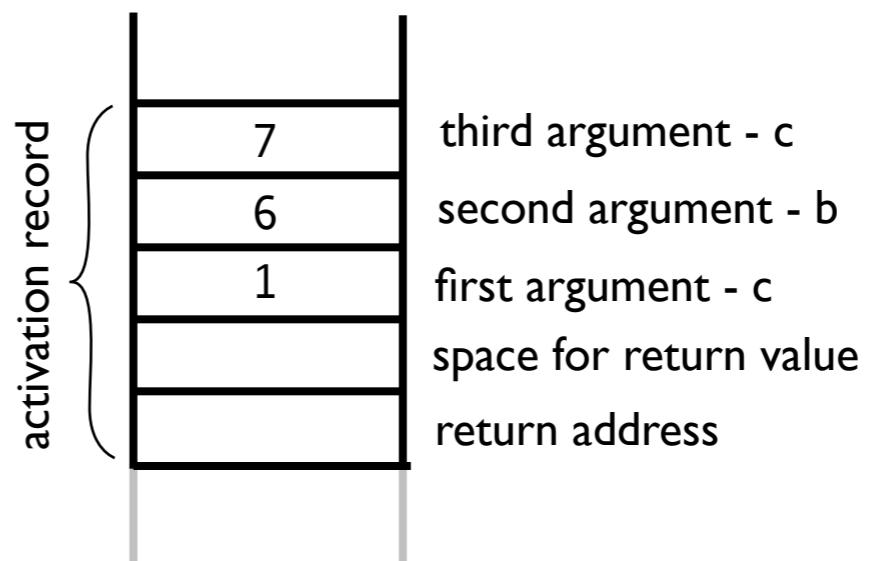
#include <stdio.h>

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>

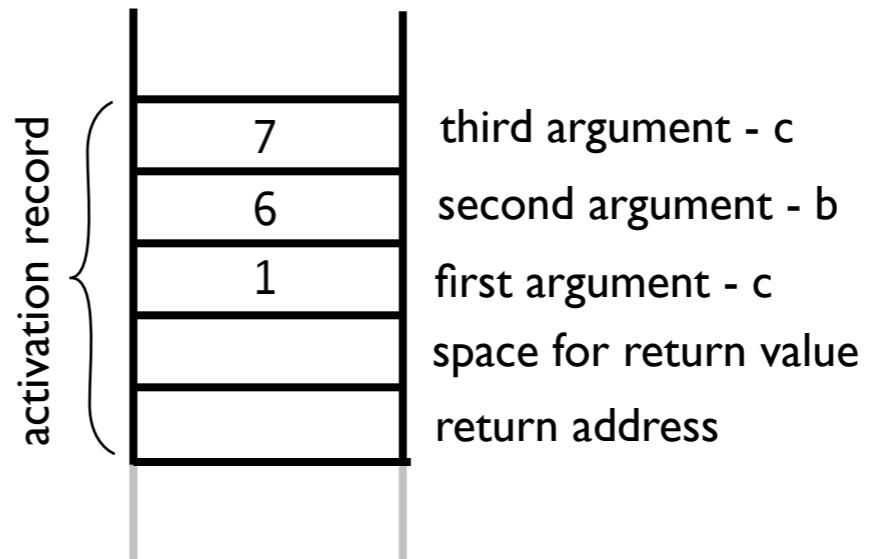
static int calc(int a, int b, int c)
{
    return a * b / c;
}

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.



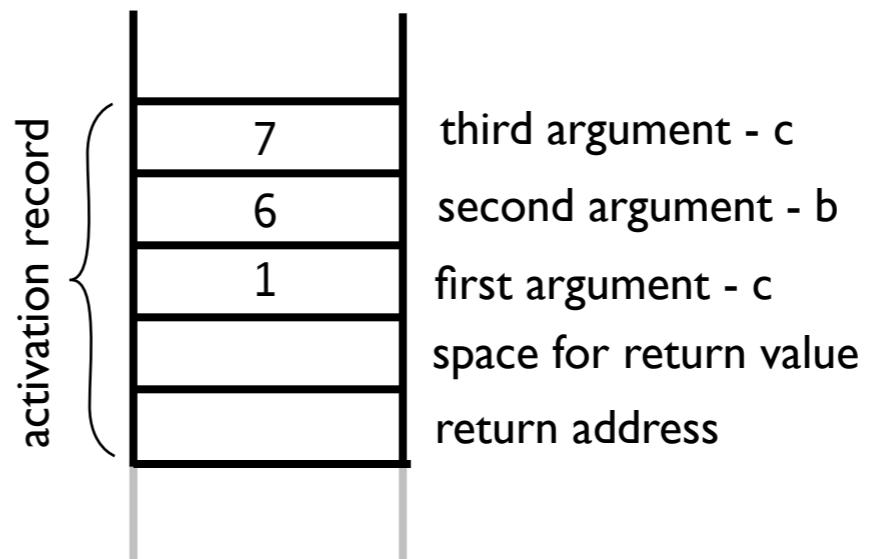
```
#include <stdio.h>

→ 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);
}
```

```

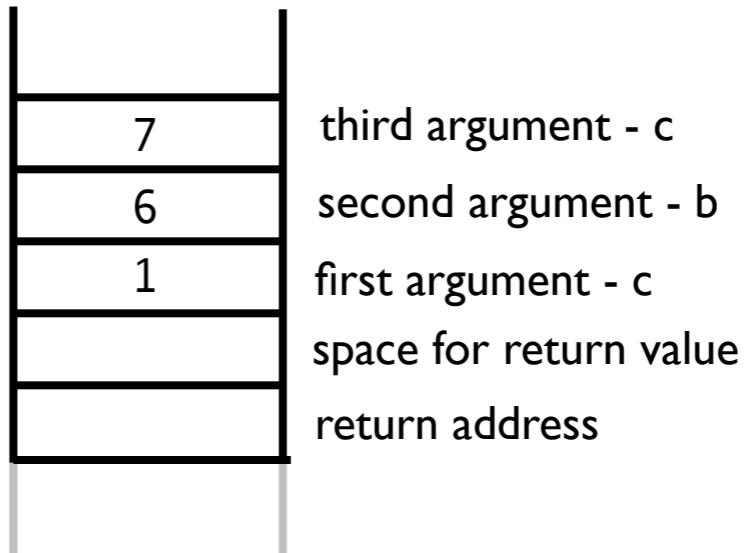
#include <stdio.h>

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);
}

```



```

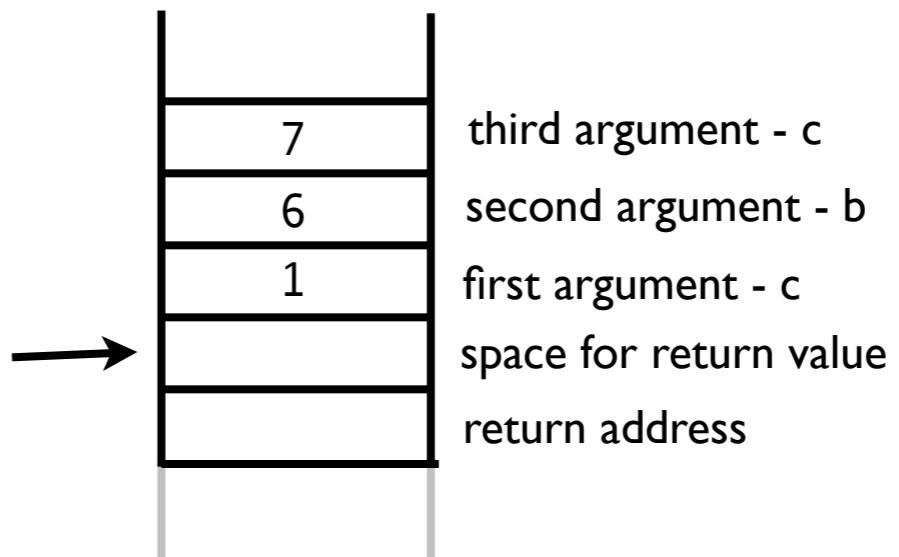
#include <stdio.h>

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);
}

```



```

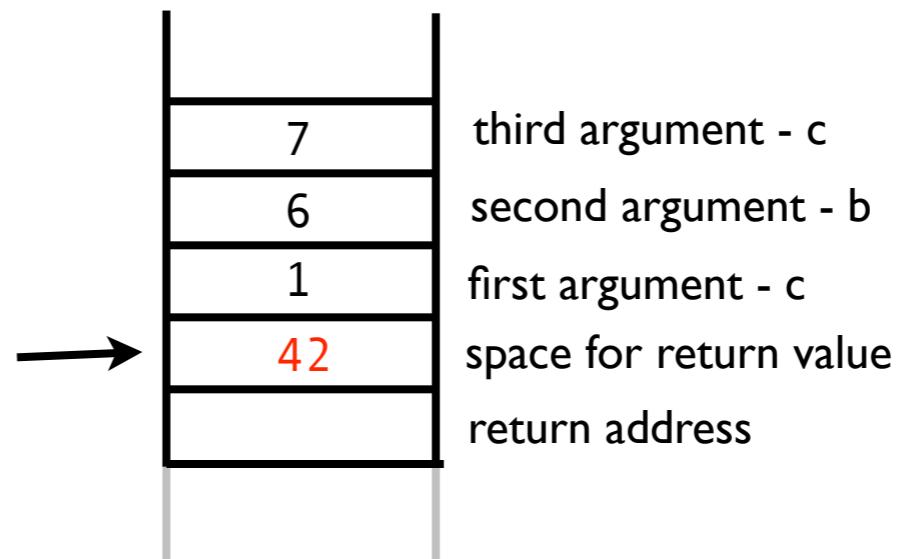
#include <stdio.h>

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);
}

```



```

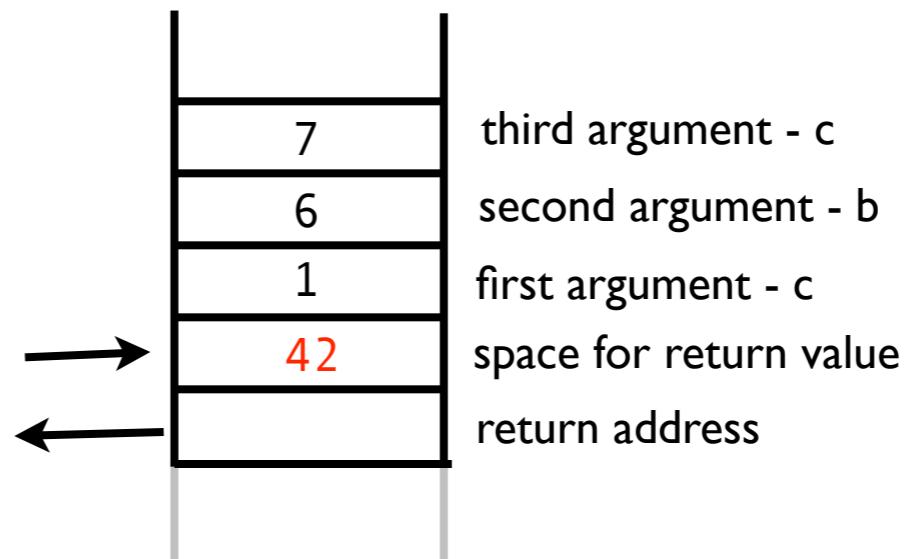
#include <stdio.h>

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);
}

```



```

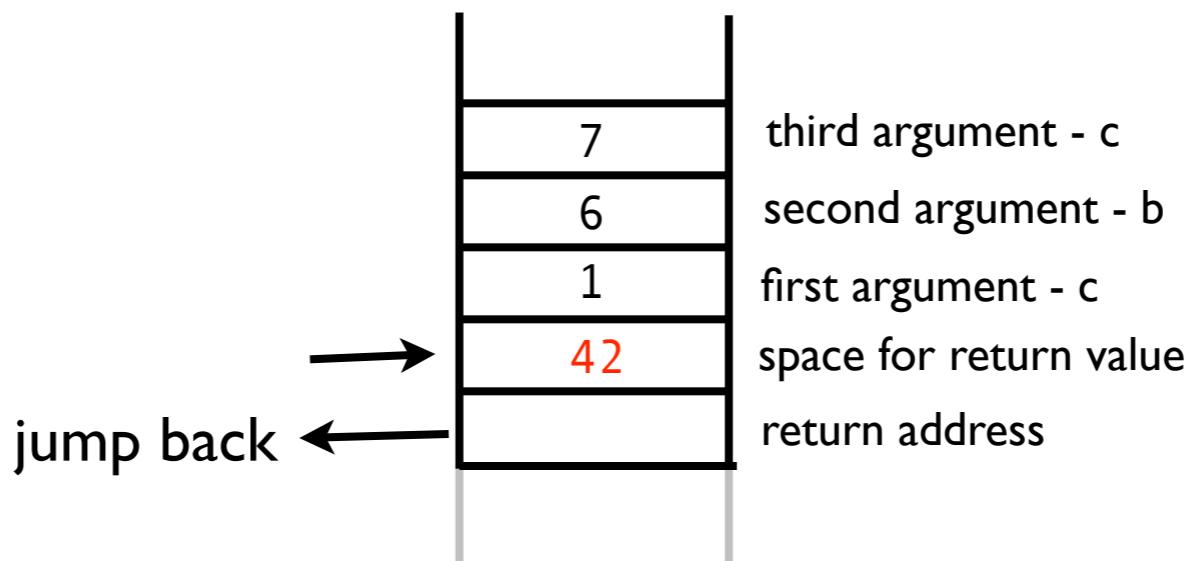
#include <stdio.h>

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);
}

```



```

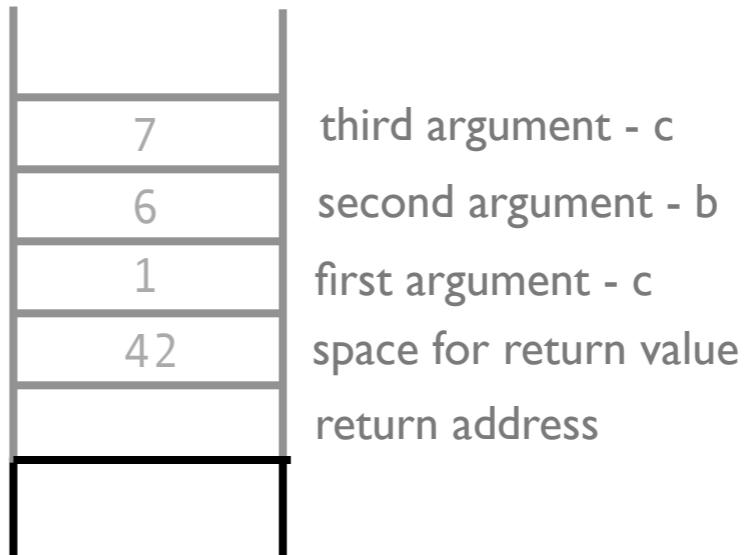
#include <stdio.h>

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);
}
```

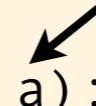


```
#include <stdio.h>

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 * life() / everything();
    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.

the C standard defines the expected behaviour, but says very little about **how** it should be implemented.

the C standard defines the expected behaviour, but says very little about **how** it should be implemented.

**this is a key feature of C, and one of the  
reason why C is such a successful  
programming language on such a wide  
range of hardware!**

# Behavior

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

int main(void)
{
    // implementation-defined
    int i = ~0;
    i >>= 1;
    printf("%d\n", i);

    // unspecified output
    printf("4") + printf("2");
    printf("\n");

    // undefined
    int k = INT_MAX;
    k += 1;
    printf("%d\n", k);
}
```

**implementation-defined behavior:**  
the construct is not incorrect; the code must compile; the compiler must document the behavior

**unspecified behavior:** the same as implementation-defined except the behavior need not be documented

**undefined behavior:** the standard imposes no requirements ; anything at all can happen, all bets are off, nasal demons might fly out of your nose.

*Note that many compilers will not give you any warnings when compiling this code, and due to the undefined behavior caused by signed integer overflow above, the whole program is in theory undefined.*

just to illustrate the point. What do you think will happen when we run this program?

```
#include <stdio.h>

int a(void) { printf("a"); return 3; }
int b(void) { printf("b"); return 4; }

int main(void)
{
    int c = a() + b();
    printf("%d\n", c);
}
```

just to illustrate the point. What do you think will happen when we run this program?

```
#include <stdio.h>

int a(void) { printf("a"); return 3; }
int b(void) { printf("b"); return 4; }

int main(void)
{
    int c = a() + b();
    printf("%d\n", c);
}
```

According to the C standard, this program will print:

just to illustrate the point. What do you think will happen when we run this program?

```
#include <stdio.h>

int a(void) { printf("a"); return 3; }
int b(void) { printf("b"); return 4; }

int main(void)
{
    int c = a() + b();
    printf("%d\n", c);
}
```

According to the C standard, this program will print:

ba7

just to illustrate the point. What do you think will happen when we run this program?

```
#include <stdio.h>

int a(void) { printf("a"); return 3; }
int b(void) { printf("b"); return 4; }

int main(void)
{
    int c = a() + b();
    printf("%d\n", c);
}
```

According to the C standard, this program will print:

ba7

or

ab7

just to illustrate the point. What do you think will happen when we run this program?

```
#include <stdio.h>

int a(void) { printf("a"); return 3; }
int b(void) { printf("b"); return 4; }

int main(void)
{
    int c = a() + b();
    printf("%d\n", c);
}
```

According to the C standard, this program will print:

ba7

or

ab7

Unlike most modern programming languages, the evaluation order of most expressions are **not** specified in C

And while we are on a roll...What do you think  
might happen when we run this program?

And while we are on a roll...What do you think might happen when we run this program?

```
#include <stdio.h>

int main(void)
{
    int v[] = {9,7,5,3,1};
    int i = 2;
    int n = v[i++] - v[i++];
    printf("%d\n", n);
    printf("%d\n", i);
    return 0;
}
```

And while we are on a roll...What do you think might happen when we run this program?

```
#include <stdio.h>

int main(void)
{
    int v[] = {9,7,5,3,1};
    int i = 2;
    int n = v[i++] - v[i++];
    printf("%d\n", n);
    printf("%d\n", i);
    return 0;
}
```

you might get:

And while we are on a roll...What do you think might happen when we run this program?

```
#include <stdio.h>

int main(void)
{
    int v[] = {9,7,5,3,1};
    int i = 2;
    int n = v[i++] - v[i++];
    printf("%d\n", n);
    printf("%d\n", i);
    return 0;
}
```

you might get:

42  
0

And while we are on a roll...What do you think might happen when we run this program?

```
#include <stdio.h>

int main(void)
{
    int v[] = {9,7,5,3,1};
    int i = 2;
    int n = v[i++] - v[i++];
    printf("%d\n", n);
    printf("%d\n", i);
    return 0;
}
```

you might get:

42  
0

If you break the rules of the language, the behaviour of the whole program is undefined. Anything can happen! And the compiler will often not be able to give you a warning.

And while we are on a roll...What do you think might happen when we run this program?

```
#include <stdio.h>

int main(void)
{
    int v[] = {9,7,5,3,1};
    int i = 2;
    int n = v[i++] - v[i++];
    printf("%d\n", n);
    printf("%d\n", i);
    return 0;
}
```

you might get:

42  
0

Try it!

If you break the rules of the language, the behaviour of the whole program is undefined. Anything can happen! And the compiler will often not be able to give you a warning.

```
#include <stdio.h>

int main(void)
{
    int v[] = {9,7,5,3,1};
    int i = 2;
    int n = v[i++] - v[i++];
    printf("%d\n", n);
    printf("%d\n", i);
    return 0;
}
```

```
#include <stdio.h>

int main(void)
{
    int v[] = {9,7,5,3,1};
    int i = 2;
    int n = v[i++] - v[i++];
    printf("%d\n", n);
    printf("%d\n", i);
    return 0;
}
```

The violation here is that we are violating the rules of sequencing, which, among other things says that a variable can not be updated twice between two sequence points.

```
#include <stdio.h>

int main(void)
{
    int v[] = {9,7,5,3,1};
    int i = 2;
    int n = v[i++] - v[i++];
    printf("%d\n", n);
    printf("%d\n", i);
    return 0;
}
```

Here is what I get on my machine:

```
$ gcc -O -Wall -Wextra -pedantic foo.c
$ ./a.out
0
4
$
```

The violation here is that we are violating the rules of sequencing, which, among other things says that a variable can not be updated twice between two sequence points.

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);  
}
```

how is a for loop  
really working?

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);  
}
```

how is a for loop  
really working?

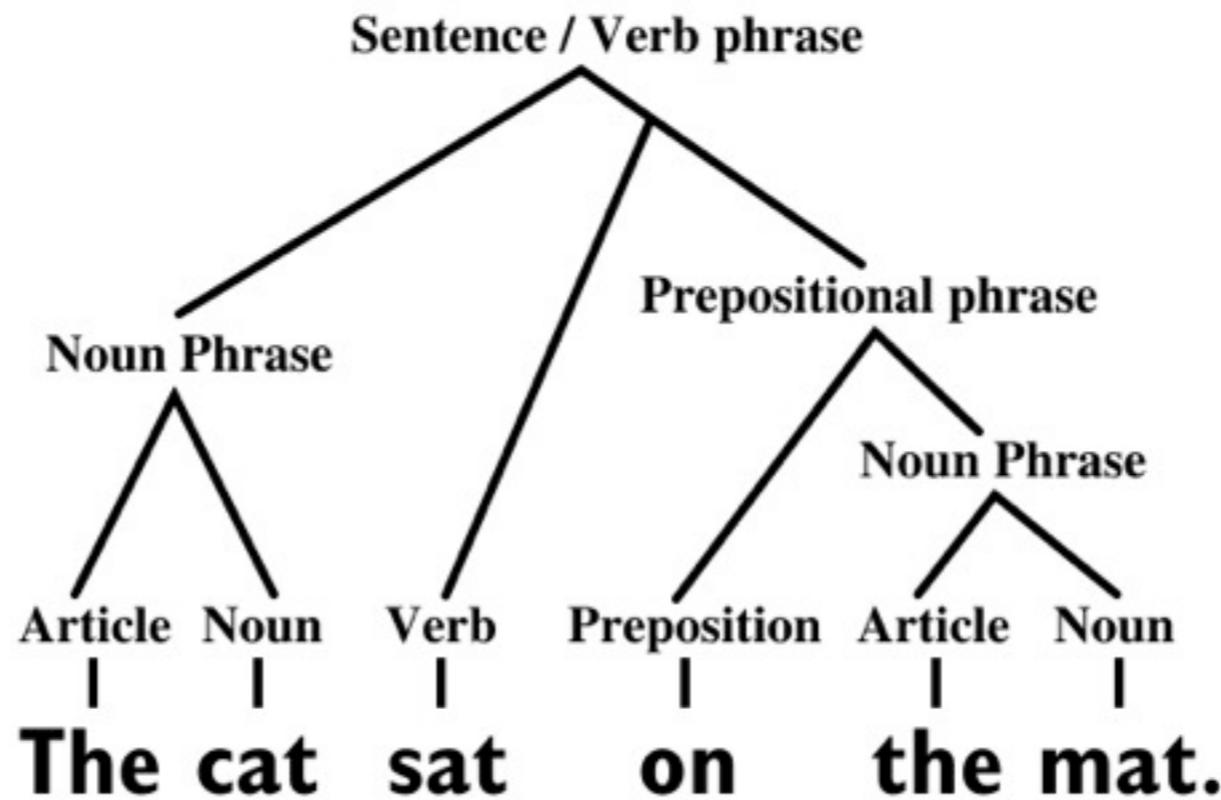
and what does  
this mean?

how many  
arguments does  
this function  
accept?

```
$ 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);
}
```

```
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;
keyword → 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");
    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;
    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



```
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



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

function definition



```
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);
}
```

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);
}
```

**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 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

declaration without  
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);  
}
```

```
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 with initialization

assignment expression

declaration without 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 with initialization

assignment expression

expression statement

declaration without 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 with initialization

assignment expression

expression statement

declaration without initialization

expression

statement

```
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

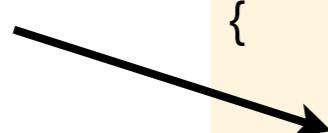
static void say_hello(const char * who)
{
    printf("Hello %s\n", who);
}

// 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);
}
```

type 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);
}
```

type qualifier

type specifier

**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);
}
```

**type qualifier**

**type 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);
}
```

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);  
}
```

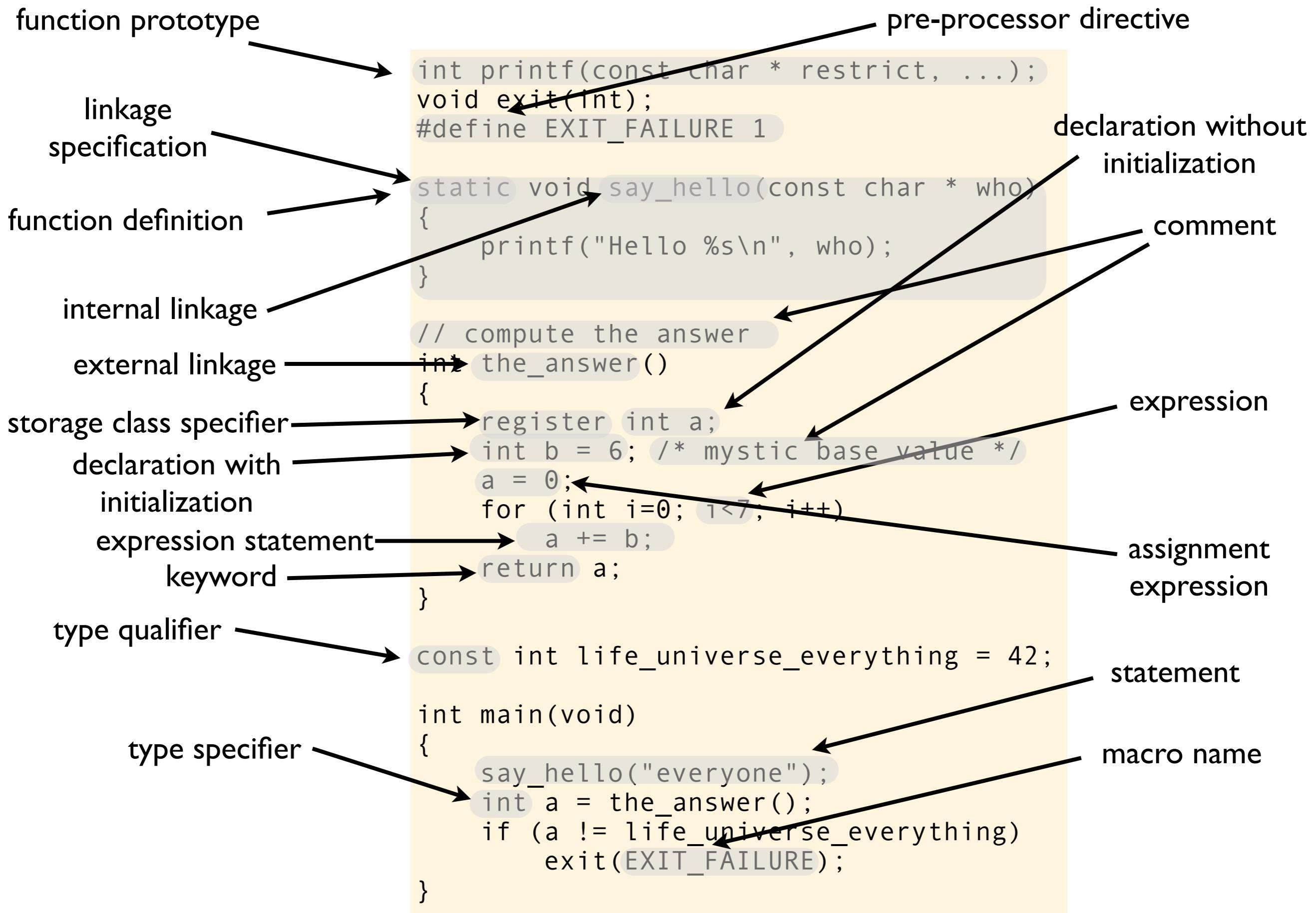
// compute the answer

```
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;
```

macro name

```
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

# Exercise: Deep thought, Part I

```
#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.c

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

dt.h

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);
}
```

# Exercise: 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
```

# Exercise: Deep thought, Part I

```
#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.c

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

dt.h

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
```

# Exercise: Deep thought, Part I

```
#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.c

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

dt.h

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
```

# Exercise: Deep thought, Part I

```
#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.c

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

dt.h

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
```

# Exercise: Deep thought, Part I

```
#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.c

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

dt.h

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
```

# Exercise: Deep thought, Part I

```
#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.c

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

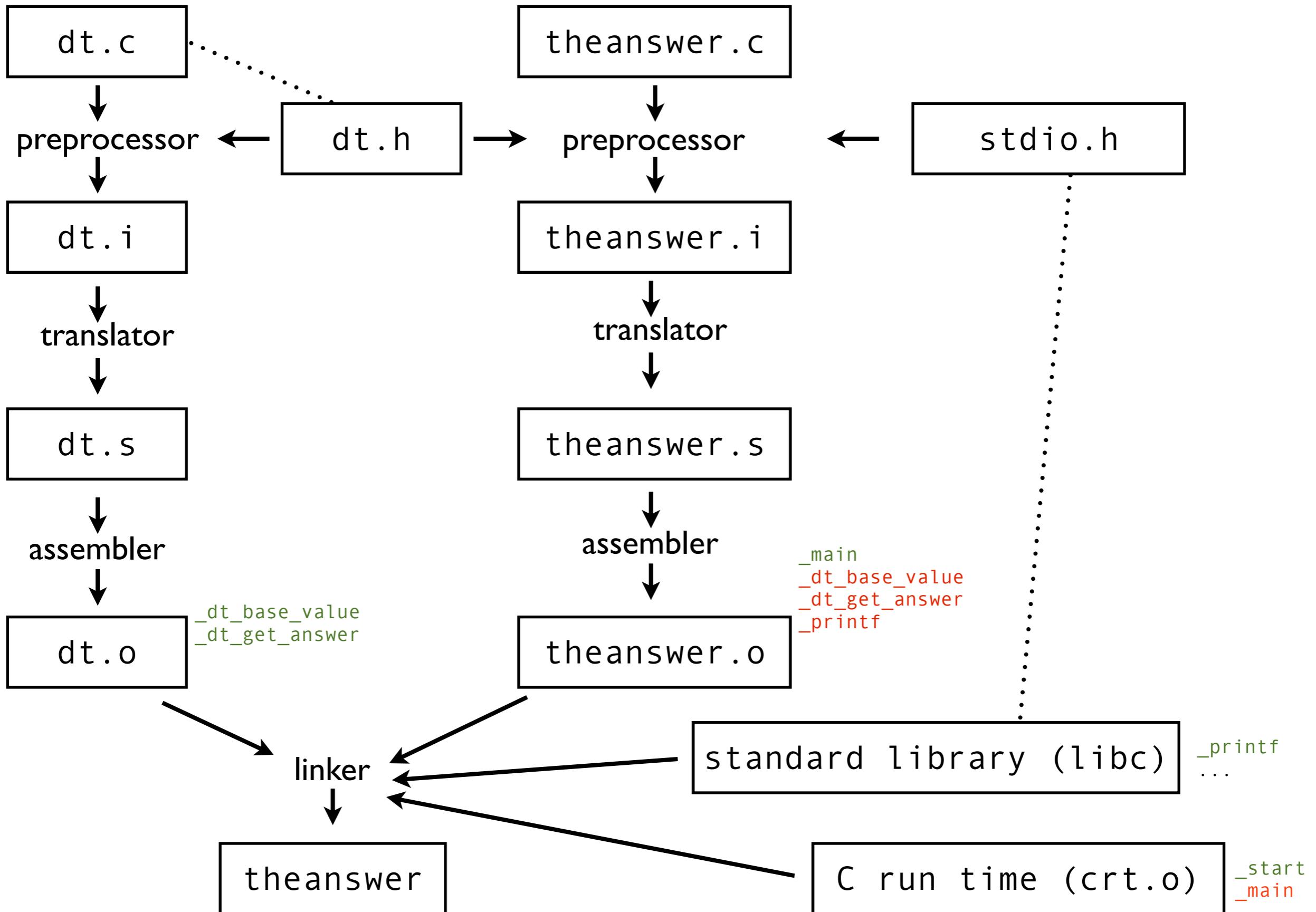
dt.h

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
$
```



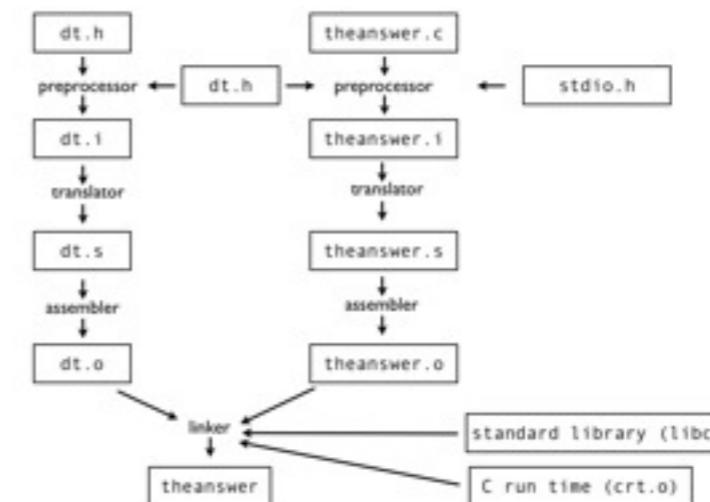
# Exercise: Deep thought, Part 2

```
dt.c                                     dt.h                                     theanswer.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;  
}

void dt\_init(void);  
int dt\_compute\_answer(void);

#include <stdio.h>  
#include "dt.h"  
  
int main(void)  
{  
 dt\_init();  
 int answer = dt\_compute\_answer();  
 printf("The answer is %d\n",  
 answer);  
}



# Exercise: Deep thought, Part 2

```
dt.c                                     dt.h
#include "dt.h"
int dt_base_value;
static int dt_answer;

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

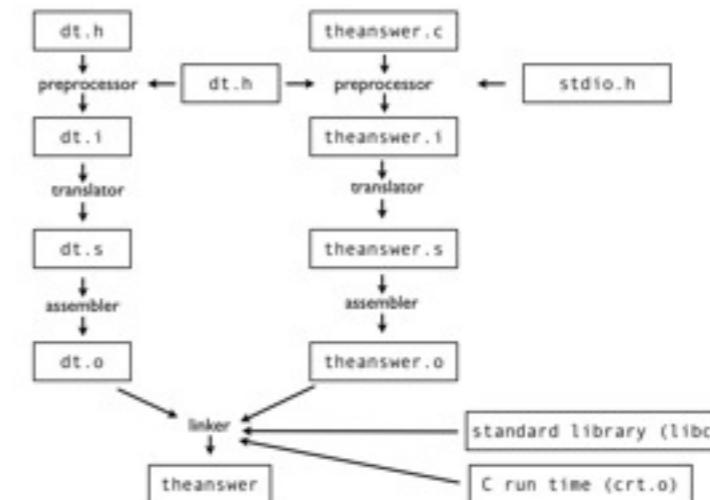
void dt_init(void)
{
    dt_base_value = 6;
}

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
```

# Exercise: Deep thought, Part 2

```
dt.c                                     dt.h
#include "dt.h"
int dt_base_value;
static int dt_answer;

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

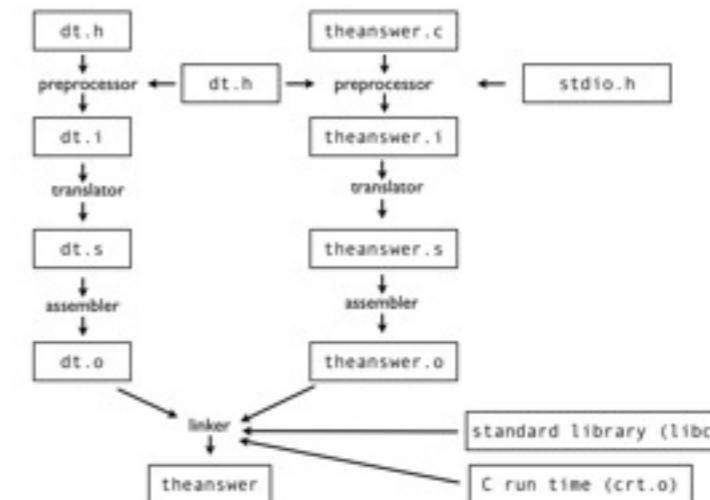
void dt_init(void)
{
    dt_base_value = 6;
}

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

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
```

# Exercise: Deep thought, Part 2

```
dt.c                                     dt.h
#include "dt.h"
int dt_base_value;
static int dt_answer;

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

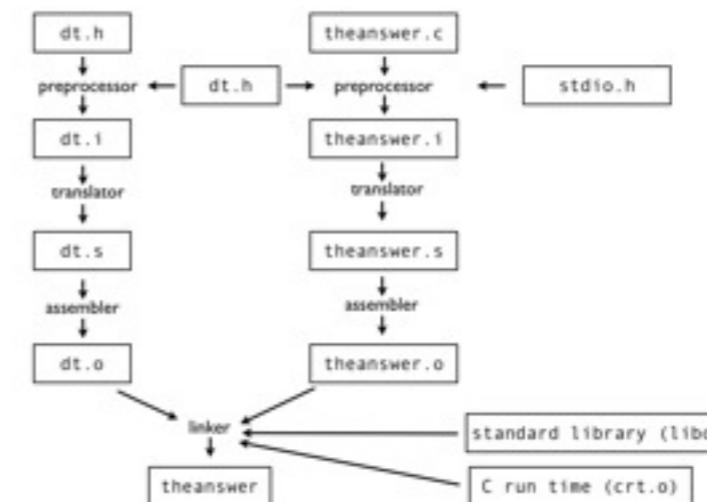
void dt_init(void)
{
    dt_base_value = 6;
}

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

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

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

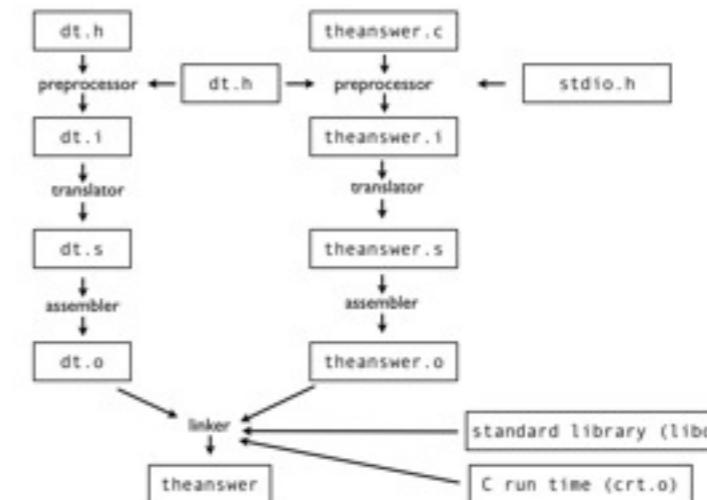
int main(void)
{
    dt_init();
    int answer = dt_compute_answer();
    printf("The answer is %d\n",
           answer);
}
```



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

# Exercise: Deep thought, Part 2

```
dt.c                                     dt.h  
-----  
#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;  
}  
  
-----  
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
```

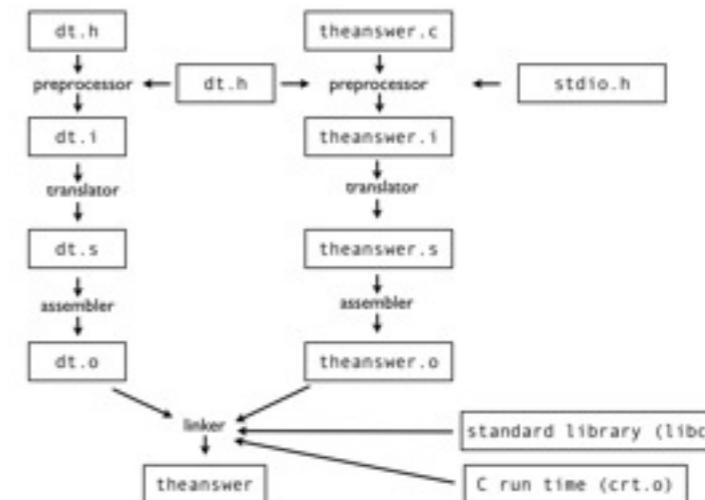
# Exercise: Deep thought, Part 2

```
dt.c                                     dt.h                                     theanswer.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;  
}

void dt\_init(void);  
int dt\_compute\_answer(void);

#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
```

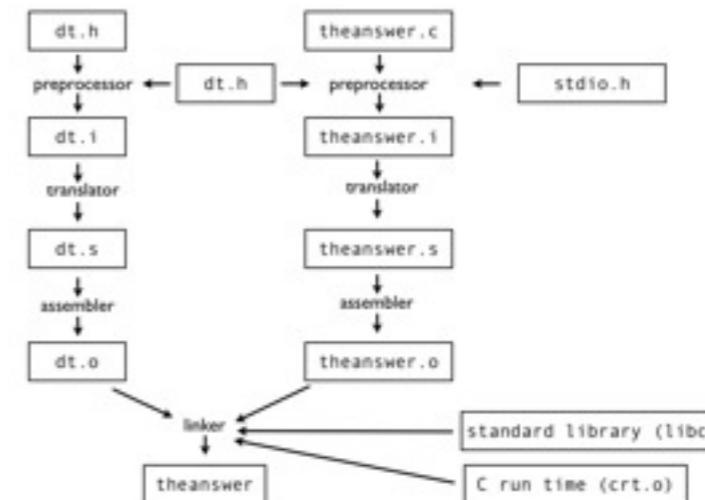
# Exercise: Deep thought, Part 2

```
dt.c                                     dt.h                                     theanswer.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;  
}

void dt\_init(void);  
int dt\_compute\_answer(void);

#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
```

# Exercise: Deep thought, Part 2

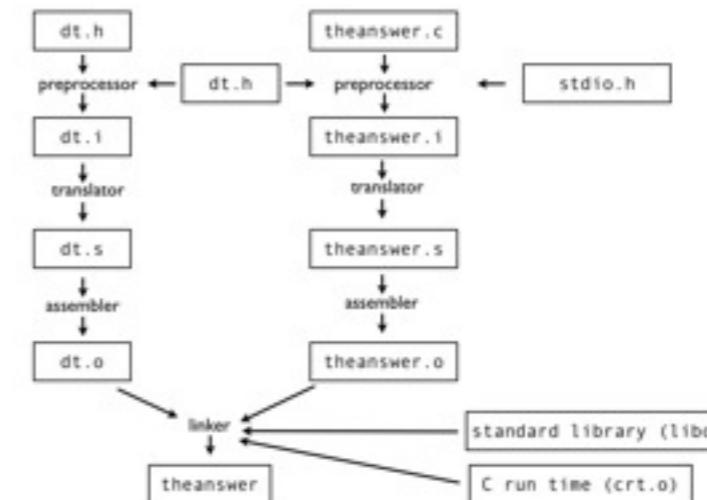
```
dt.c                                     dt.h                                     theanswer.c
#include "dt.h"
int dt_base_value;
static int dt_answer;

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;
}

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



```
$ 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-temp theanswer.c
```

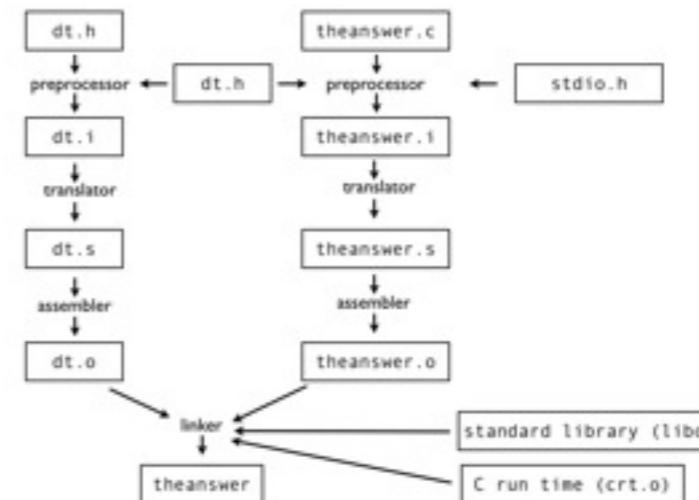
# Exercise: Deep thought, Part 2

```
dt.c                                     dt.h                                     theanswer.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;  
}

void dt\_init(void);  
int dt\_compute\_answer(void);

#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-temp theanswer.c  
$ ls theanswer.*
```

# Exercise: Deep thought, Part 2

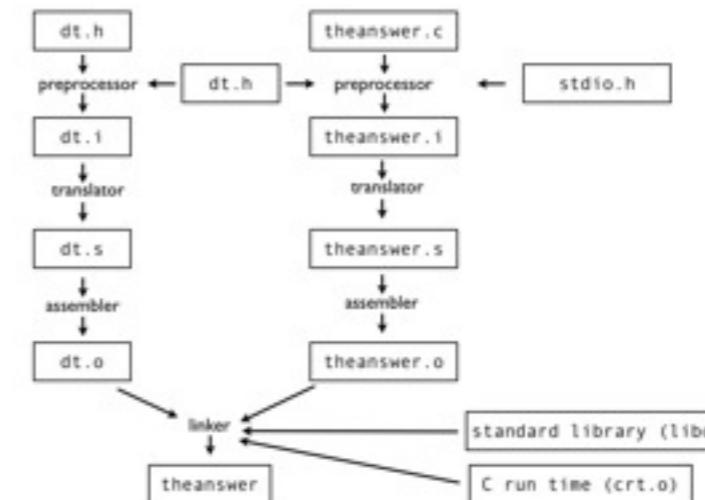
```
dt.c                                     dt.h                                     theanswer.c
#include "dt.h"
int dt_base_value;
static int dt_answer;

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;
}

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



```
$ 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-temp theanswer.c
$ ls theanswer.*
$ nm theanswer.o
```

# Exercise: Deep thought, Part 2

```
dt.c                                     dt.h
#include "dt.h"
int dt_base_value;
static int dt_answer;

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

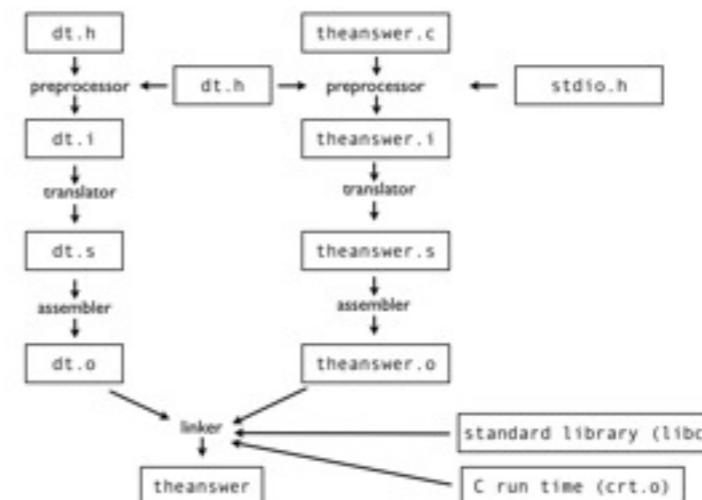
void dt_init(void)
{
    dt_base_value = 6;
}

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

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-temp theanswer.c
$ ls theanswer.*
$ nm theanswer.o

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

# Exercise: Deep thought, Part 2

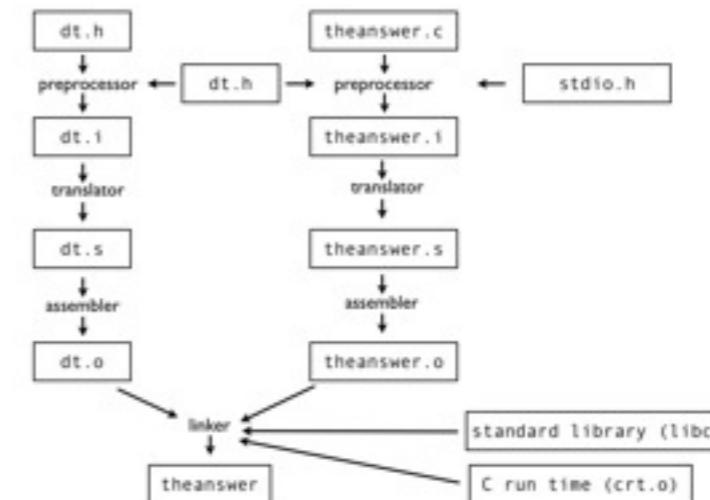
```
dt.c                                     dt.h                                     theanswer.c
#include "dt.h"
int dt_base_value;
static int dt_answer;

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;
}

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



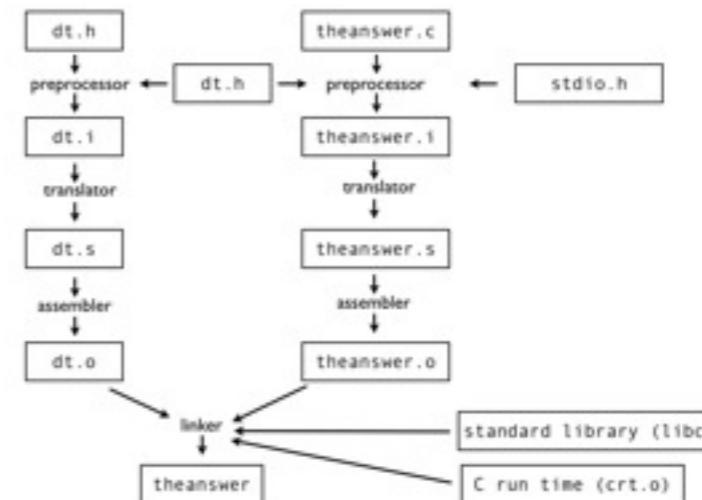
```
$ cc -E dt.c >dt.i
$ cat dt.i
$ cc -S dt.i
$ cat dt.s
$ cc -c dt.s
$ nm dt.o

$ cc -c -save-temp theanswer.c
$ ls theanswer.*
$ nm theanswer.o

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

# Exercise: Deep thought, Part 2

```
dt.c                                     dt.h  
-----  
#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;  
}  
  
-----  
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-temp theanswer.c  
$ ls theanswer.*  
$ nm theanswer.o  
  
$ ld -lc -o theanswer dt.o theanswer.o /usr/lib/crt1.o  
$ ./theanswer  
The answer is 42
```

# Exercise: 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);  #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> |

# Exercise: 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
```

# Exercise: 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
```

# Exercise: 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
```

# Exercise: 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 run_computer
```

# Exercise: Deep thought, Part 3

```
dt.c                               dt.h
#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;
}
```

```
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 -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break run_computer
(gdb) set dt_base_value = 8
```

# Exercise: 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 run_computer
(gdb) set dt_base_value = 8
(gdb) cont
```

# Exercise: Deep thought, Part 3

```
dt.c                               dt.h
#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;
}

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 -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break run_computer
(gdb) set dt_base_value = 8
(gdb) cont
(gdb) disassemble run_computer
```

# Exercise: Deep thought, Part 3

```
dt.c                               dt.h
#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;
}

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 -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break run_computer
(gdb) set dt_base_value = 8
(gdb) cont
(gdb) disassemble run_computer
(gdb) set disassembly-flavor intel
```

# Exercise: Deep thought, Part 3

```
dt.c                               dt.h
#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;
}

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 -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break run_computer
(gdb) set dt_base_value = 8
(gdb) cont
(gdb) disassemble run_computer
(gdb) set disassembly-flavor intel
(gdb) disassemble run_computer
```

# Exercise: Deep thought, Part 3

```
dt.c                               dt.h
#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;
}

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 -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break run_computer
(gdb) set dt_base_value = 8
(gdb) cont
(gdb) disassemble run_computer
(gdb) set disassembly-flavor intel
(gdb) disassemble run_computer
(gdb) help
```

# Exercise: Deep thought, Part 3

```
dt.c                               dt.h
#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;
}

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 -g -o theanswer dt.c theanswer.c
$ gdb theanswer
(gdb) run
(gdb) break run_computer
(gdb) set dt_base_value = 8
(gdb) cont
(gdb) disassemble run_computer
(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

*(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)*

Under the hood



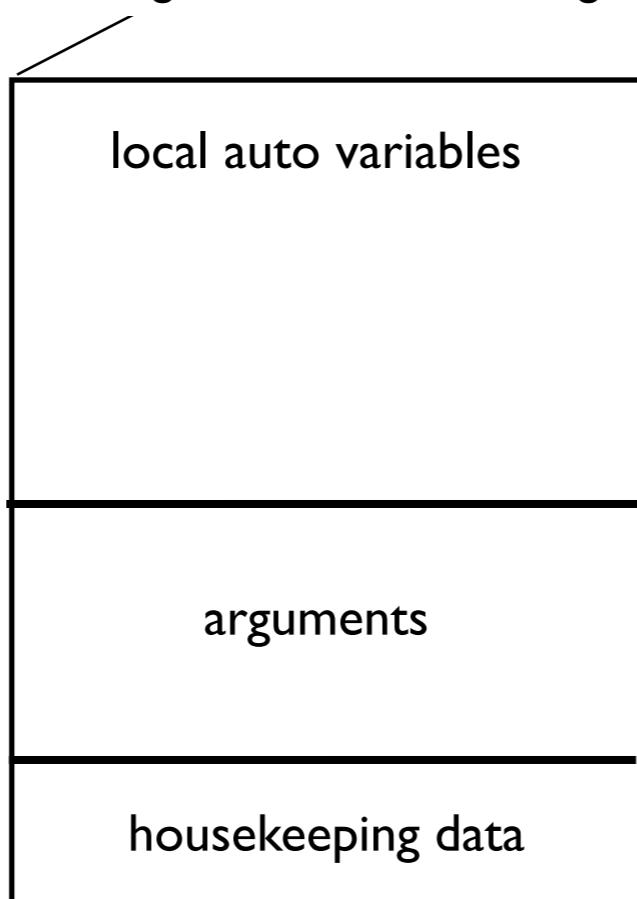
# Memory Layout \*

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.

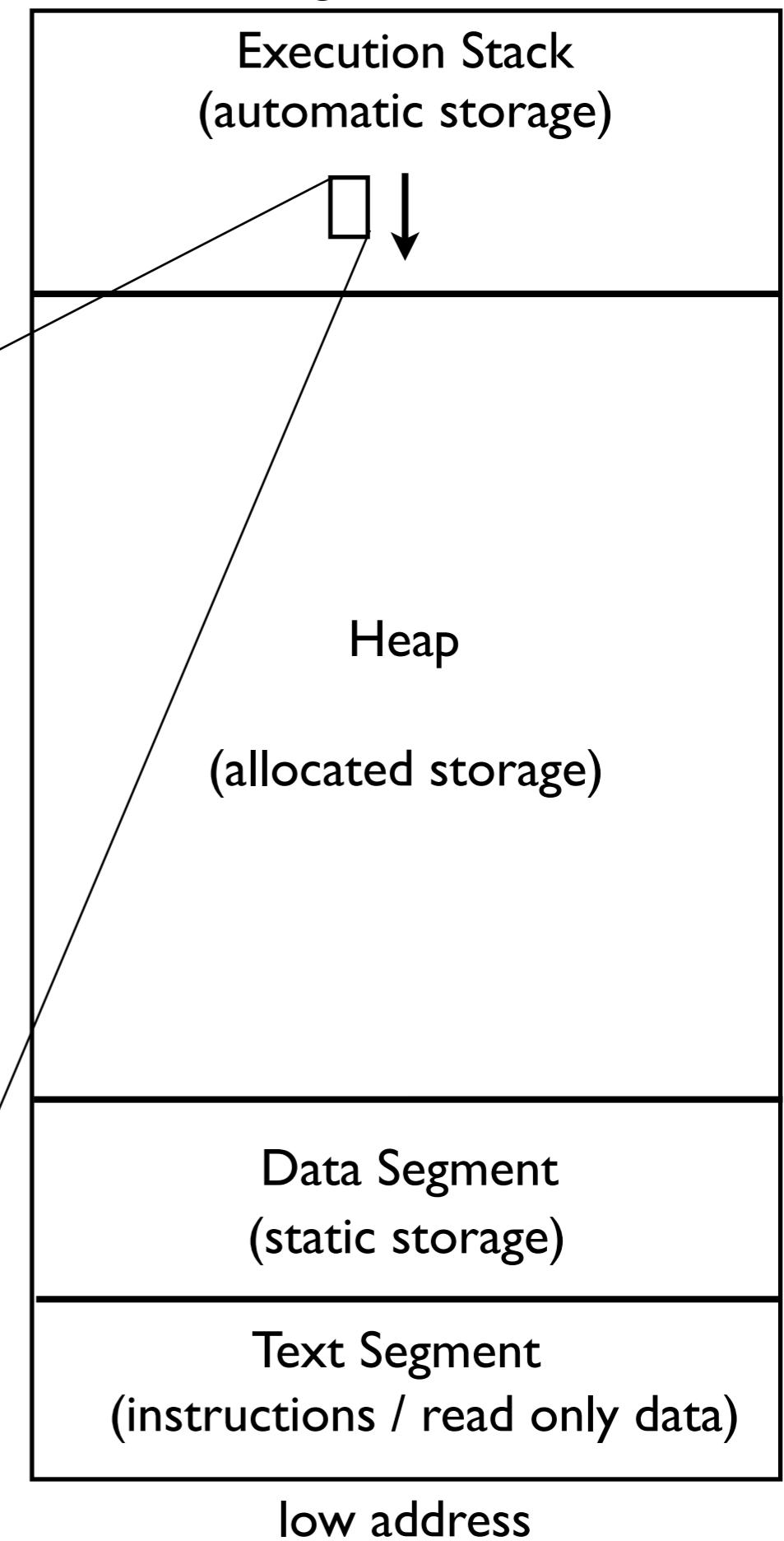
The stack and the heap is typically not cleaned up in any way at startup, or during execution, so before objects are explicitly initialized they typically get garbage values based on whatever is left in memory from discarded objects and previous executions. In other words, the programmer must do all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

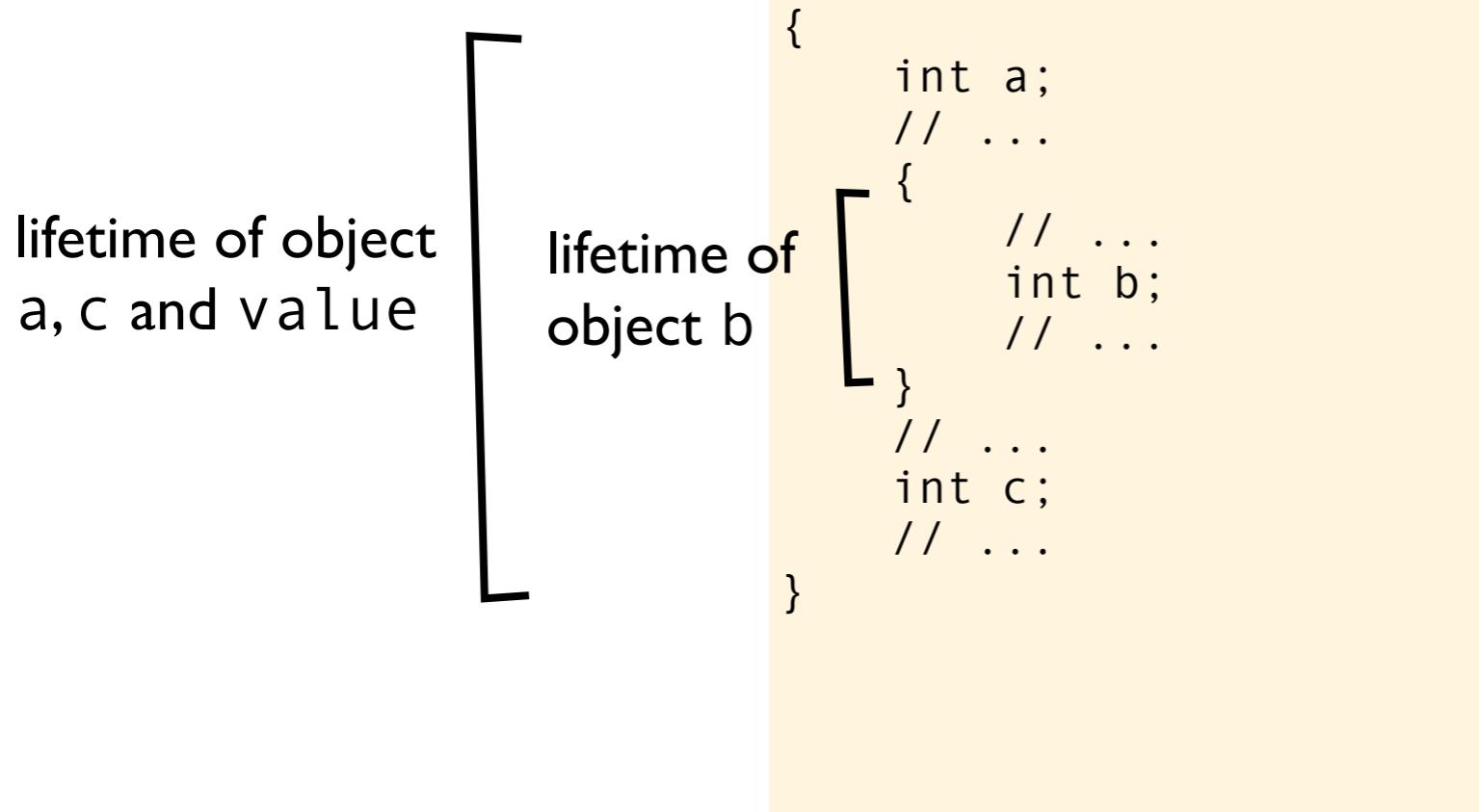


(\*) The C standard does not dictate any particular memory layout, so what is presented here is just a useful conceptual example model that is similar to what some architecture and run-time environments look like



# 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

# Building blocks



Deep C - a 3 day course  
Jon Jagger & Olve Maudal  
(March 27, 2012)

so lets get started with the basics

this is what the compiler see

```
if (answer < 0)  
    answer = 42;
```

this is what you see

lexical analyser

```
if  
(  
answer  
<  
0  
)  
answer  
=  
42  
;
```

this is what the compiler see

There are 5 classes of tokens in a C program

| token          | examples               |
|----------------|------------------------|
| keyword        | if struct restrict ... |
| identifier     | the_answer main i      |
| constant       | 9 6f 013 '9' 9.6       |
| string literal | “Hello”<br>...         |
| punctuator     | + - * / % = += ; { } , |

```
if (answer < 0)
    answer = 42;
```

lexical analyser

```
if
(
answer
<
0
)
answer
=
42
;
```

There are 5 classes of tokens in a C program

| token          | examples               |
|----------------|------------------------|
| keyword        | if struct restrict ... |
| identifier     | the_answer main i      |
| constant       | 9 6f 013 '9' 9.6       |
| string literal | "Hello" ...            |
| punctuator     | + - * / % = += ; { } , |

```
if (answer < 0)
    answer = 42;
```

lexical analyser

punctuators

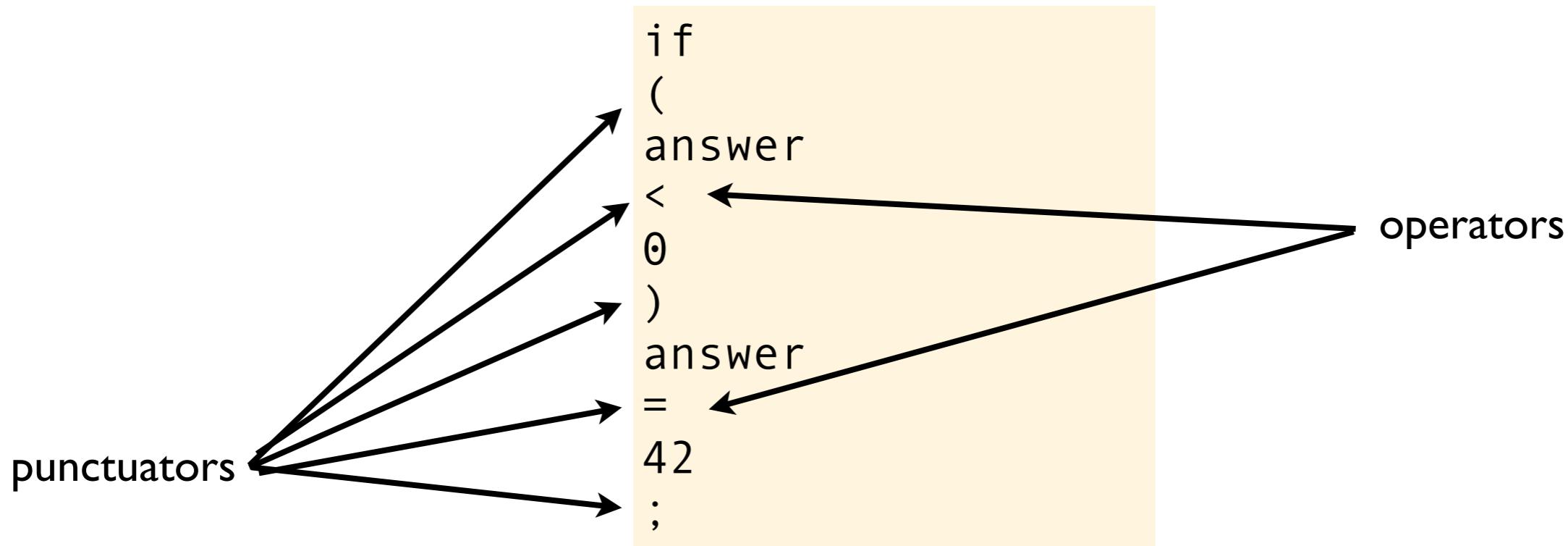
```
if
(
answer
<
0
)
answer
=
42
;
```

There are 5 classes of tokens in a C program

| token          | examples               |
|----------------|------------------------|
| keyword        | if struct restrict ... |
| identifier     | the_answer main i      |
| constant       | 9 6f 013 '9' 9.6       |
| string literal | "Hello" ...            |
| punctuator     | + - * / % = += ; { } , |

```
if (answer < 0)  
    answer = 42;
```

lexical analyser

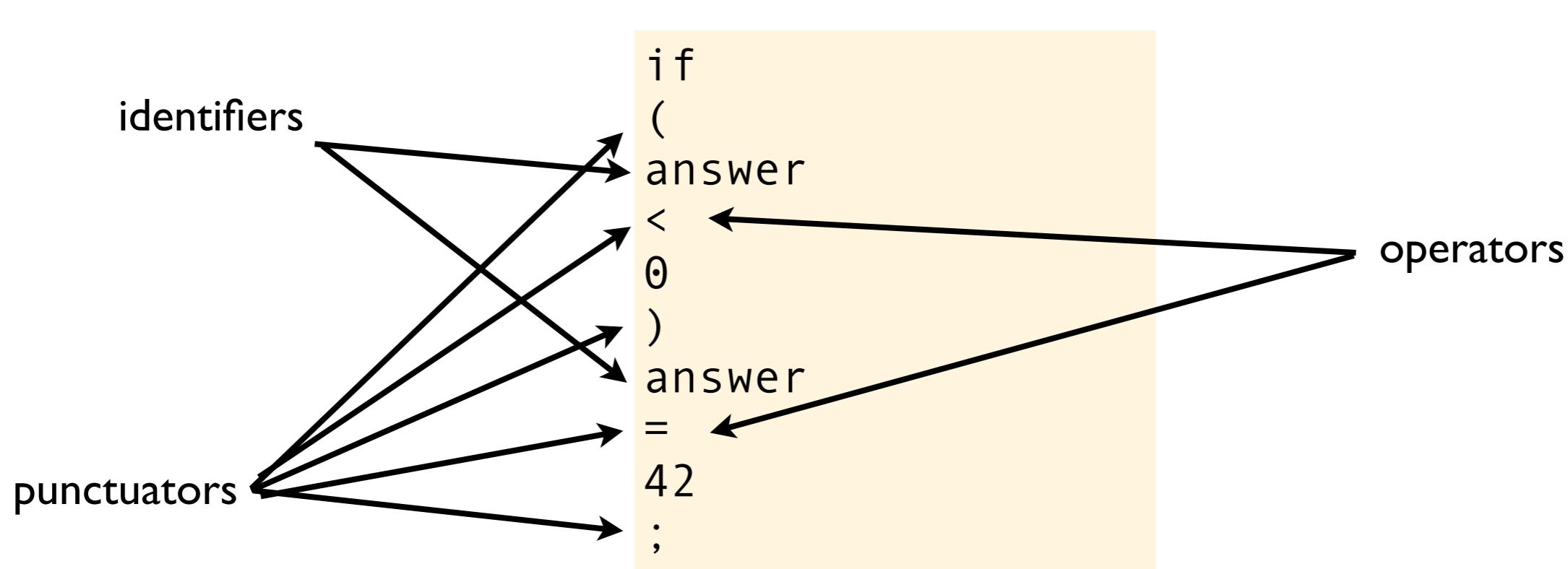


There are 5 classes of tokens in a C program

| token          | examples               |
|----------------|------------------------|
| keyword        | if struct restrict ... |
| identifier     | the_answer main i      |
| constant       | 9 6f 013 '9' 9.6       |
| string literal | "Hello" ...            |
| punctuator     | + - * / % = += ; { } , |

```
if (answer < 0)  
    answer = 42;
```

lexical analyser

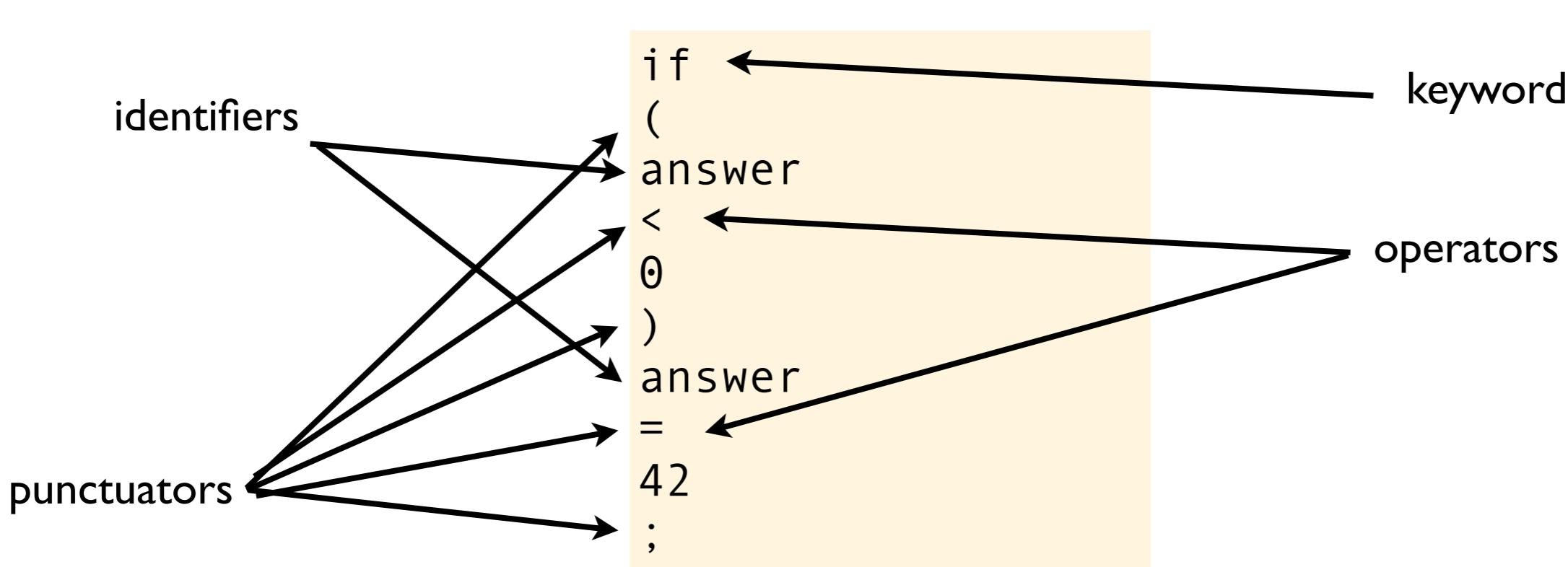


There are 5 classes of tokens in a C program

| token          | examples               |
|----------------|------------------------|
| keyword        | if struct restrict ... |
| identifier     | the_answer main i      |
| constant       | 9 6f 013 '9' 9.6       |
| string literal | "Hello" ...            |
| punctuator     | + - * / % = += ; { } , |

```
if (answer < 0)
    answer = 42;
```

lexical analyser

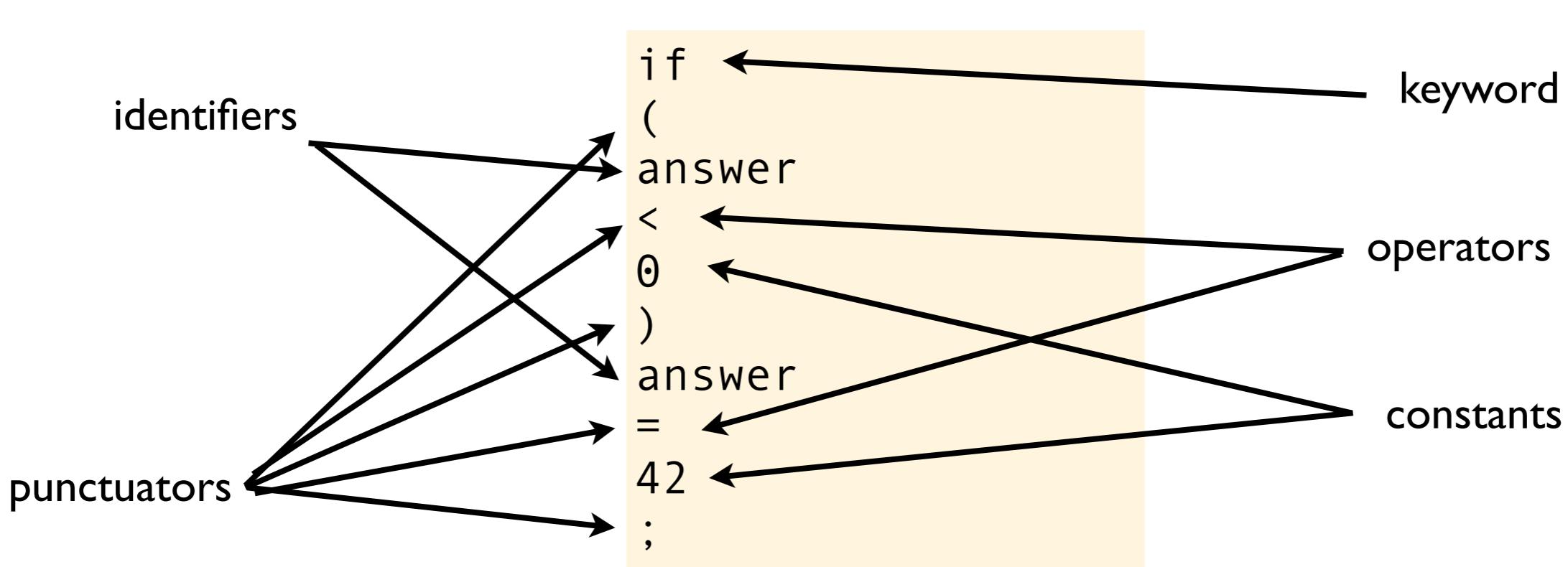


There are 5 classes of tokens in a C program

| token          | examples               |
|----------------|------------------------|
| keyword        | if struct restrict ... |
| identifier     | the_answer main i      |
| constant       | 9 6f 013 '9' 9.6       |
| string literal | "Hello" ...            |
| punctuator     | + - * / % = += ; { } , |

```
if (answer < 0)  
    answer = 42;
```

lexical analyser



There are 5 classes of tokens in a C program

| token          | examples               |
|----------------|------------------------|
| keyword        | if struct restrict ... |
| identifier     | the_answer main i      |
| constant       | 9 6f 013 '9' 9.6       |
| string literal | "Hello" ...            |
| punctuator     | + - * / % = += ; { } , |

```
#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);
}
```



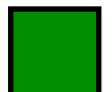
keywords

```
#include <stdio.h>

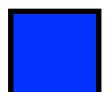
static int calc(int a, int b, int c)
{
    return a * b / c;
}

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);
}
```



keywords



identifiers

```
#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);
}
```

 keywords

 constants

 identifiers

```
#include <stdio.h>

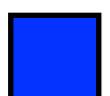
static int calc(int a, int b, int c)
{
    return a * b / c;
}

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);
}
```

 keywords

 constants

 identifiers

 string literals

```
#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);
}
```

 keywords

 constants

 punctuators

 identifiers

 string literals

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

but what about  
this one?

```
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; }
```

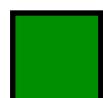
```
int main(void)
```

```
{
```

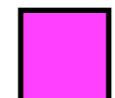
```
    int a = calc(universe, life(), everything());
```

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

```
}
```



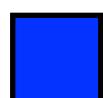
keywords



constants



punctuators



identifiers



string literals

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

but what about  
this one?

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

this is a preprocessor directive, and is  
not really part of the language. We will  
deal with this later.

```
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);  
}
```

  keywords

  constants

  punctuators

  identifiers

  string literals

## token

## examples

|                |                        |
|----------------|------------------------|
| keyword        | if struct restrict ... |
| identifier     | the_answer main i      |
| constant       | 9 6f 013 '9' 9.6       |
| string literal | “Hello”<br>...         |
| punctuator     | + - * / % = += ; { } , |

# Punctuators

- tend to change meaning in depending on context
- operators are examples of punctuators
- eg, sometimes & means bitwise AND, other times address-of
- can be used to change presedence
- can be used to introduce sequence points
- create new blocks of expressions and statements
- ...

# String literals

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

int main(void)
{
    char * str2 = "bar";
    const char * str3 = "gaz";
    char * str4 = "Hello" " World!";
    char str5[] = {'a','b','c'};
    assert( sizeof(str5) == 3 );
    char str6[] = "abc";
    assert( sizeof(str6) == 4 );
    assert( strlen(str6) == 3 );

    const size_t msg_size = 13;
    char str7[msg_size];
    strcpy(str7, "Hello World!");
    assert( strlen(str7)+1 == msg_size );

    const size_t msg_len = 12;
    char str8[msg_len+1];
    strcpy(str8, "Hello World!");
    assert( strlen(str8) == msg_len );
}
```

# Constants

- default is int
- default is double
- avoid magic numbers, replace with named intention
- beware of comparison with floating point constants

# Identifiers

## Rules

- made of letters, digits and underscores
- can't start with a digit
- case sensitive

variable1  
802id

## Recommendations

- don't start with underscore (reserved)
- use case consistently
- avoid abrs
- don't use clever spelling
- don't use hungarian

\_variable  
vrbls  
iVarible  
  
CamelCase  
snake\_case  
mixedCase

```
int year = 1992;  
if (is_leap_year(year))  
    do_extra_maintainance();
```

```
int year = 1992;  
if (IsLeapYear(year))  
    DoExtraMaintenance();
```

```
int Year = 1992;  
if (IsLeapYear(Year))  
    do_extra_maintainance();
```



Follow the local de facto standards - but when in doubt, follow K&R

# Keywords

|          |          |          |            |
|----------|----------|----------|------------|
| auto     | extern   | short    | while      |
| break    | float    | signed   | _Bool      |
| case     | for      | sizeof   | _Complex   |
| char     | goto     | static   | _Imaginary |
| const    | if       | struct   |            |
| continue | inline   | switch   |            |
| default  | int      | typedef  |            |
| do       | long     | union    |            |
| double   | register | unsigned |            |
| else     | restrict | void     |            |
| enum     | return   | volatile |            |

**Quick...What does the following code print?**

# Quick...What does the following code print?

```
#include <stdio.h>

int main(void)
{
    int a = 44;
    a -= 2;
    printf("%d\n", a);
}
```

# Quick...What does the following code print?

```
#include <stdio.h>

int main(void)
{
    int a = 44;
    a -= 2;
    printf("%d\n", a);
}
```

-2

# Quick...What does the following code print?

```
#include <stdio.h>

int main(void)
{
    int a = 44;
    a -= 2;
    printf("%d\n", a);
}
```

-2

but you do get a warning from the compiler?

# Quick...What does the following code print?

```
#include <stdio.h>

int main(void)
{
    int a = 44;
    a -= 2;
    printf("%d\n", a);
}
```

-2

but you do get a warning from the compiler?

```
$ gcc foo.c && ./a.out
```

# Quick...What does the following code print?

```
#include <stdio.h>

int main(void)
{
    int a = 44;
    a -= 2;
    printf("%d\n", a);
}
```

-2

but you do get a warning from the compiler?

```
$ gcc foo.c && ./a.out
-2
```

# Quick...What does the following code print?

```
#include <stdio.h>

int main(void)
{
    int a = 44;
    a -= 2;
    printf("%d\n", a);
}
```

-2

but you do get a warning from the compiler?

```
$ gcc foo.c && ./a.out
-2
$ gcc -Wall -Wextra -pedantic foo.c && ./a.out
```

# Quick...What does the following code print?

```
#include <stdio.h>

int main(void)
{
    int a = 44;
    a -= 2;
    printf("%d\n", a);
}
```

-2

but you do get a warning from the compiler?

```
$ gcc foo.c && ./a.out
-2
$ gcc -Wall -Wextra -pedantic foo.c && ./a.out
-2
```

# Quick...What does the following code print?

```
#include <stdio.h>

int main(void)
{
    int a = 44;
    a -= 2;
    printf("%d\n", a);
}
```

-2

but you do get a warning from the compiler?

```
$ gcc foo.c && ./a.out
-2
$ gcc -Wall -Wextra -pedantic foo.c && ./a.out
-2
```

lexical analyser

this is what the compiler sees

```
...
int
a
=
44
;
a
=
-
2
;
printf
(
"%d\n"
,
a
...

```

## Summary

- keywords
- identifiers
- punctuators (operators and separators)
- string literals
- constants
- compiler warnings

Building blocks



# Functions and Program Structure

```
#ifndef DATE_INCLUDED
#define DATE_INCLUDED

struct date
{
    int year,month,day;
};

struct date_string
{
    char buffer[42];
};

struct date_string date_to_string(const struct date *);
```

```
#include "date.hpp"
#include <stdio.h>

int main(void)
{
    struct date today = { 2012, 3, 28 };
    puts(date_to_string(&today).buffer);
    return 0;
}
```

2012:03:28

by  
Jon Jagger & Olve Maudal  
(March 28, 2012)

# header files

can contain

- #includes
- macro guards (idempotent)
- macros (e.g. EOF)
- type declarations (e.g. FILE)
- external data declarations (e.g. stdin)
- external function declarations (e.g. printf)
- inline function definitions

should not contain

- function definitions (unless inline)
- data definitions

# function declarations

a function declaration promises that a function exists

- typically written in header files
- return type must be explicit

stdio.h

```
...
FILE * fopen(const char * path, const char * mode);
int fflush(FILE * stream);
...
void perror(const char * diagnostic);
FILE * tmpfile(void);
...
```



parameter names are optional but help readability



avoid bool parameters as they can be confusing

# header example

#IF Not DEFined

ALL UPPERCASE  
is a very strong preprocessor  
convention

```
stdio.h
#ifndef STDIO_INCLUDED
#define STDIO_INCLUDED

#define EOF ...

typedef struct FILE FILE;

extern FILE * stdin;

FILE * fopen(const char * path, const char * mode);
int fflush(FILE * stream);
...
void perror(const char * diagnostic);
FILE * tmpfile(void);
...

#endif
```

# exercise

## I.What does this say?

```
int get_value();
```

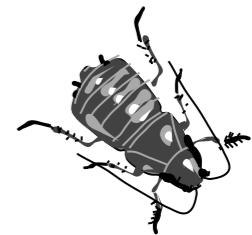
## 2.Try the following in your environment.

```
#include <stdio.h>

int get_value();

int main()
{
    printf("%d\n", get_value());
    printf("%d\n", get_value(42));
    printf("%d\n", get_value(42,24));
    return 0;
}

int get_value()
{
    return 42;
}
```



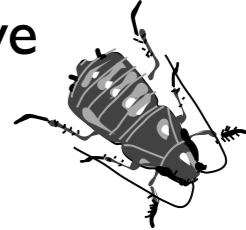
## f() vs f(void)

if a function has no parameters say so explicitly with void

```
int rand();
```



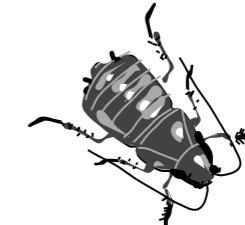
provides no parameter information;  
the definition of rand can have  
any number of parameters!



```
int rand(void);
```



rand has no parameters;  
the definition of rand must have  
no parameters



Parameter-argument number and type mismatches are **not** caught with the `-Wall` option. Read `-Wall` as `-Wmost`

### call.c

```
int func();  
int main(void)  
{  
    return func(42);  
}
```

```
$ gcc -Wall call.c  
$
```

You need to use `-Wstrict-prototypes`

### call.c

```
int func(void);  
int main(void)  
{  
    return func(42);  
}
```

```
$ gcc -Wstrict-prototypes call.c  
error: too many arguments to function 'func'  
$
```

# definitions

- a definition honors the declaration promise
- definitions are written in source files

stdio.h

```
#include <stdio.h>
...
FILE * fopen(const char * path, const char * mode)
{
    ...
}

int fflush(FILE * stream)
{
    ...
}

void perror(const char * message)
{
    ...
}
...
```

parameter names are required



parameter name can be different to that used in the function declaration

# pass by pointer

use a pointer to a non-const

- if the definition needs to change the target

delay.h

```
void delay( struct date * when);
```

the lack of a const here means  
delay might change \*when

```
#include "delay.h"

int main(void)
{
    struct date due = { 2012, march, 28 };
    delay(&due);
    ...
}
```

struct date \* when = &due;

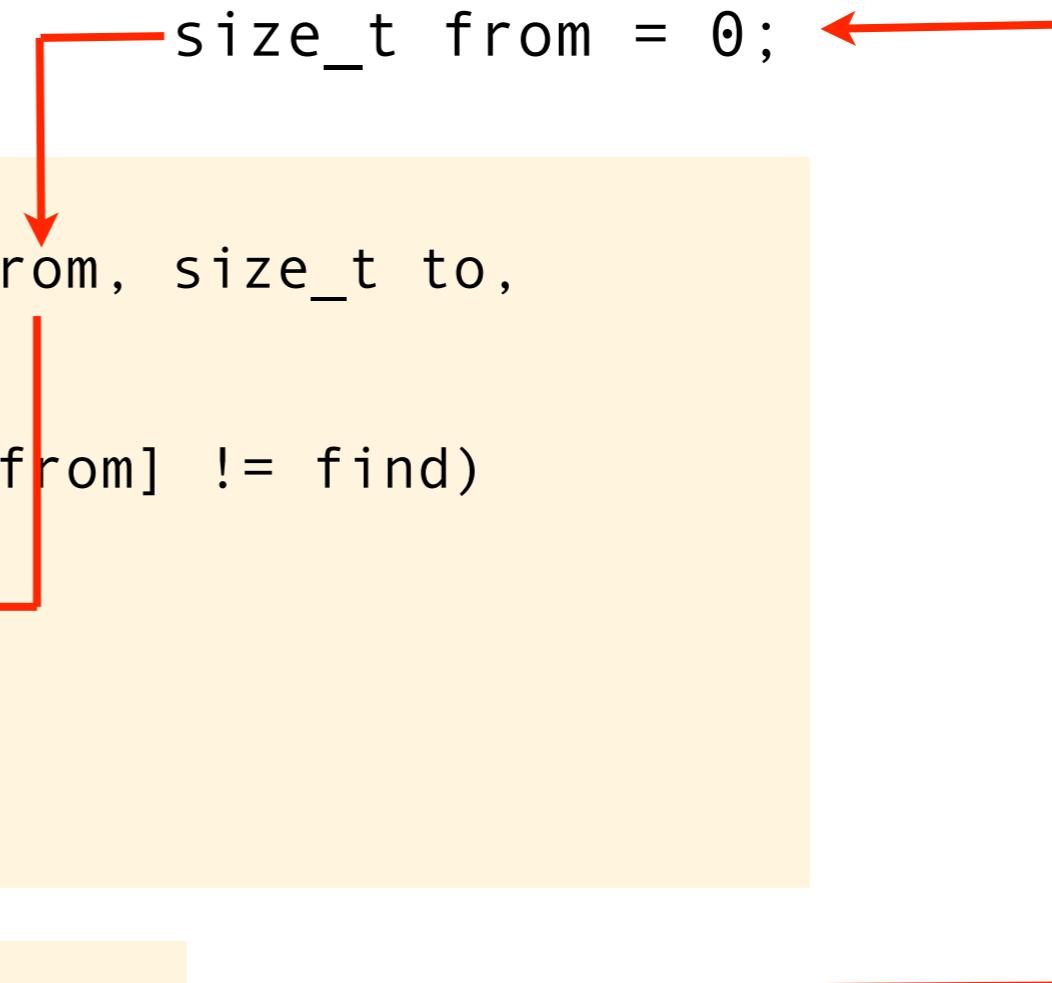


# pass by value

changing the parameter does not change the argument

```
bool search(  
    const int values[], size_t from, size_t to,  
    int find)  
{  
    while (from != to && values[from] != find)  
    {  
        from++;  
    }  
    return from != to;  
}
```

```
int main(void)  
{  
    ...  
    ... search(array, 0, size, 42);  
    ...  
}
```



# pass by value

works for enums and structs too

- but not for arrays

date.h

```
struct date
{
    int year; int month; int day;
};

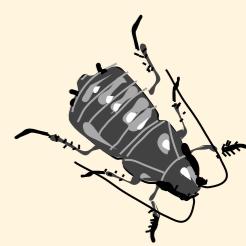
const char * day_name(struct date when);
```

```
#include "date.h"
```

```
int main(void)
{
    struct date today = { 2012, march, 28 };
    ...
    puts(day_name(today));

    assert(today.year == 2012);
    assert(today.month == march);
    assert(today.day == 28);
}
```

Wednesday



Whole arrays **can** be passed-to and returned-from functions if you put them inside a struct. For example, instead of this...

```
void date_print(const struct date *, int, char * buffer);

int main(void)
{
    struct date today = { 2012, 3, 28 };
    char buffer[42];
    date_print(&today, 42, buffer);
    puts(buffer);
}
```

2012:03:28

You can do this...

```
struct date_string { char buffer[42]; };
struct date_string date_to_string(const struct date *);

int main(void)
{
    struct date today = { 2012, 3, 28 };
    puts(date_to_string(&today).buffer);
}
```

2012:03:28

# parameter order

- list output parameters first
  - loosely mimics assignment

```
char * strcpy(char * dst, const char * src);
```

```
#include <string.h>
...
{
    const char * from = "Hello";
    char to[128];
    ...
    //      to = from ←
    strcpy(to , from);
}
```

# pass by pointer to const

an efficient alternative to pass by copy

- except that the parameter can be null

date.h

```
const char * day_name(const struct date * when);
```



the const here promises that day\_name wont change \*when

```
#include "date.h"

int main(void)
{
    struct date today = { 2012, march, 28 };
    ...
    puts(day_name(&today));

    assert(today.year == 2012);
    assert(today.month == march);
    assert(today.day == 28);
}
```

Wednesday



you can list the type and its qualifiers in either order

```
const char * day_name(const struct date * at);
```

```
const char * day_name(struct date const * at);
```

The second style (`const` last) is common in C++ but not C. The rationale for preferring the second style is that it allows you to read a declaration from right to left:

"`at` is a pointer to a `const date`"

# register variables

a speed optimization hint to the compiler

- compiler will use registers as best it can anyway
- effect is implementation defined
- register variables can't have their address taken
- don't use!

```
void send(register short * to,
          register short * from,
          register int count)
{
    register int n = (count + 7) / 8;
    switch (count % 8)
    {
        case 0 : do { *to++ = *from++; }
        case 7 :      *to++ = *from++;
        case 6 :      *to++ = *from++;
        case 5 :      *to++ = *from++;
        case 4 :      *to++ = *from++;
        case 3 :      *to++ = *from++;
        case 2 :      *to++ = *from++;
        case 1 :      *to++ = *from++;
                      } while (--n > 0);
    }
}
```

??

# statics

a local variable can have static storage class

- a local variable with 'infinite' lifetime
- best avoided – subtle and hurts thread safety
- but ok for naming magic numbers (as are enums)

```
int remembers(void)
{
    static int count = 0;
    return ++count;
}
```

?

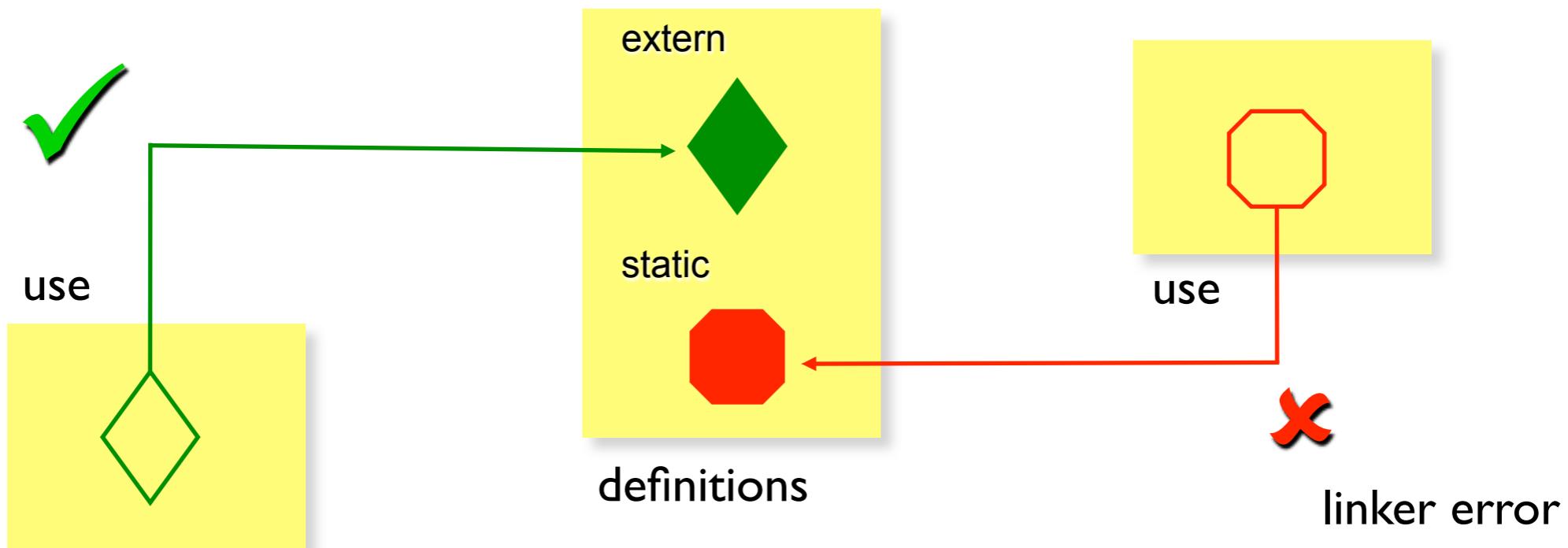
```
void send(short * to, short * from, int count)
{
    static const int unrolled = 8;

    int n = (count + unrolled - 1) / unrolled;
    switch (count % unrolled)
    {
        ...
    }
}
```

# linking

a linker links the use of an identifier in one file with its definition in another file

- an identifier is made available to the linker by giving it **external linkage** (the default) using the `extern` keyword
- an identifier is hidden from the linker by giving it **internal linkage** using the `static` keyword



# function declaration linkage

- default to external linkage
- `extern` keyword makes default explicit

time.h

```
✓ struct tm * localtime(const time_t * when);  
time_t time(time_t * when);
```

equivalent

time.h

```
? extern struct tm * localtime(const time_t * when);  
extern time_t time(time_t * when);
```



# function definition linkage

- default to external linkage
- use static keyword for internal linkage

time.c

```
✓ time_t time(time_t * when)
{   ...
}
```

time.c

```
? extern time_t time(time_t * when)
{   ...
}
```

equivalent

source.c

```
✓ static void hidden(time_t * when);

static void hidden(time_t * when)
{
    ...
}
```

# data linkage

without a storage class or an initializer

- the definition is tentative – and can be repeated
- this is confusing and not compatible with C++

ok in C, duplicate definition errors in C++

?

```
int v; // external, tentative definition
...
int v; // external, tentative definition
```

recommendation: extern data declarations

- use explicit `extern` keyword, do not initialize

recommendation: extern data definitions

- do not use `extern` keyword, do initialize

multiple declarations ok

```
extern int v;
extern int v;
```



single definition with initializer

```
int v = 42;
```



# type linkage

static can be used on a type definition

- it has no affect - in C type names do not have linkage
- don't do it

?

```
static struct date
{
    int year, month, day;
};
```

?

```
static enum month { january, ... };
```

# inline functions

## inline function rules

- all declarations must be declared inline
- there must be a definition in the translation unit – a file-scope declaration with `extern` is a definition
- does not affect sequence point model – there is still a sequence point before a call to an inline function
- prefer inlining over macros

`is_even.h`

```
#include <stdbool.h>

static inline bool is_even(int value)
{
    return value % 2 == 0;
}
```

## \_\_func\_\_

the name of the current function is available

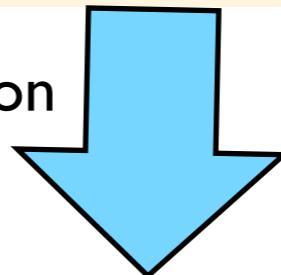
- via the reserved identifier \_\_func\_\_

```
void some_function(void)
{
    puts(__func__);
}
```

c99



as-if compiler translation



```
void some_function(void)
{
    static const char __func__[] =
        "some_function";
    puts(__func__);
}
```



## ... variadic functions

functions with a variable no. of arguments

- helpers in <stdarg.h> provide type-unsafe access

```
#include <stdarg.h>

int my_printf(const char * format, ...)
{
    va_list args;
    va_start(args, format);
    for (size_t at = 0; format[at] != '\0'; at++)
    {
        switch (format[at])
        {
            case 'd': case 'i':
                print_int(va_arg(args, int)); break;
            case 'f': case 'F':
                print_double(va_arg(args, double)); break;
            ...
        }
    }
    va_end(args);
}
```

# function pointers

in an expression the name of function "decays" into a pointer to the function

- ( ) is a binary operator with very high precedence
- $f(a, b)$  is like an infix version of  $(a) f, b$
- you can name a function without calling it!
- the result is a strongly typed function pointer

```
#include <stdio.h>

* is needed
here ↗

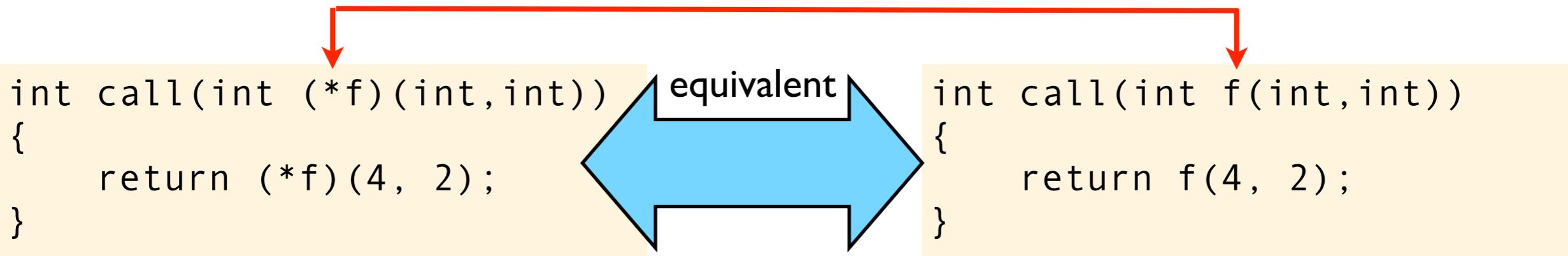
int add(int a, int b) { return a + b; }
int sub(int a, int b) { return a - b; }

int main(int argc, char * argv[])
{
    int (*f)(int,int) =
        argc % 2 == 0 ? add : sub;
    printf("%d\n", f(4, 2));
}
```

# function pointer arguments

function pointers can be function parameters!

- \* is optional on the parameter



```
#include <stdio.h>

int add(int a, int b) { return a + b; }
int sub(int a, int b) { return a - b; }

int main(int argc, char * argv[])
{
    int (*f)(int,int) =
        argc % 2 == 0 ? add : sub;

    printf("%d\n", call(f));
}
```

# function pointer arguments

- `typedef` can sometimes help

```
typedef int func(int, int);
```

```
int call(func * f)
{
    return (*f)(4, 2);
}
```

equivalent

```
int call(func f)
{
    return f(4, 2);
}
```



Function pointers provide yet another possible print-the-date design...

```
struct date
{
    int year,month,day;
};

typedef void date_printer(const char *);

void date_print(const struct date * at,
                date_printer print);
```

```
void date_print(const struct date * at,
                date_printer print)
{
    char buffer[42];
    sprintf(buffer, "%04d:%02d:%02d",
            at->year, at->month, at->day);
    print(buffer);
}
```

# summary

don't use auto keyword

don't use register keyword

don't use static local variables unless const

don't use f( ); declarations

do use f(void); in declarations

don't define data in a header file

do give static linkage to anything in a source file not declared in its header file

do ensure header files are idempotent

do pass by copy for built in types and enum...

- they are small and they will stay small
- copying is supported at a low level, very fast
- sometimes for structs as a no-alias no-indirection optimization

do pass by plain pointer...

- when the function needs to change the argument

do pass by pointer to const (mimic pass by copy)

- for most structs
- they are not small and they only get bigger!
- very fast to pass, but be aware of cost of indirection

# Exercise, day 1 (calc\_stat)

Deep C - a 3 day course  
Jon Jagger & Olve Maudal  
(March 27, 2012)

# Calculate statistics over some numbers

Write a program that calculate some simple statistics for a sequence of integers. Eg,

```
$ cat foo.dat  
4  
9  
3  
$ cat foo.dat | calcstat  
n=3, average=5.33333, min=3, max=9
```

Hint: Make sure that you test your code properly. Consider to separate the code into a library (mystatlib.c), header file (mystatlib.h) and a user of that library (calcstat.c). Use make to organize your build.

Extra: Extend the calcstat program to accept argument, -i as the input file and -o as the output file. Eg:

```
$ calcstat -i foo.dat -o /dev/stdout  
n=3, average=5.33333, min=3, max=9
```

# Test-Driven Development

## a Yahtzee game kata



by  
Jon Jagger & Olve Maudal  
(March 28, 2012)

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.



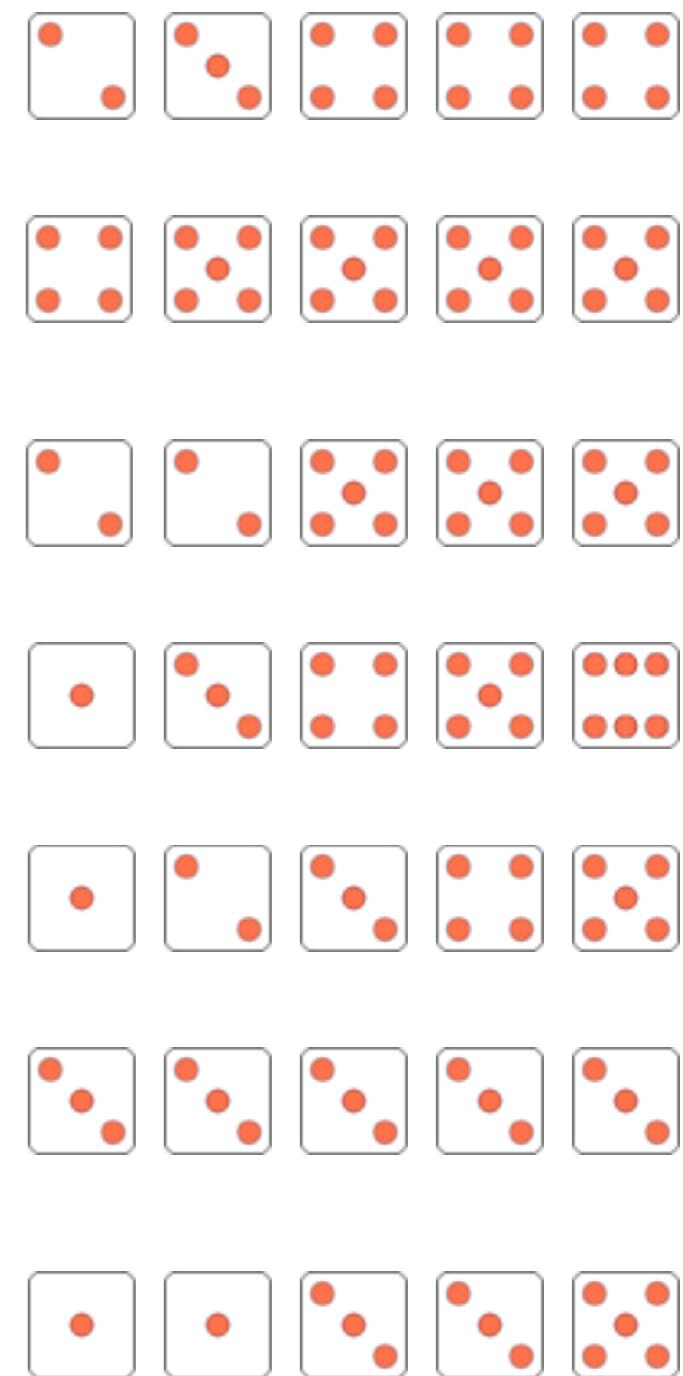
| Yahtzee                    |            | NAME _____                            |
|----------------------------|------------|---------------------------------------|
| UPPER SECTION              |            | HOW TO SCORE                          |
| Aces                       | • = 1      | Count 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                      |            | If total score is 60 or over SCORE 50 |
| TOTAL OF Upper Section     |            | →                                     |
| 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 BONUS              |            | ✓ FOR EACH BONUS SCORE 100 PER ✓      |
| TOTAL OF Lower Section     |            | →                                     |
| TOTAL OF Upper Section     |            | →                                     |
| GRAND TOTAL                |            | →                                     |

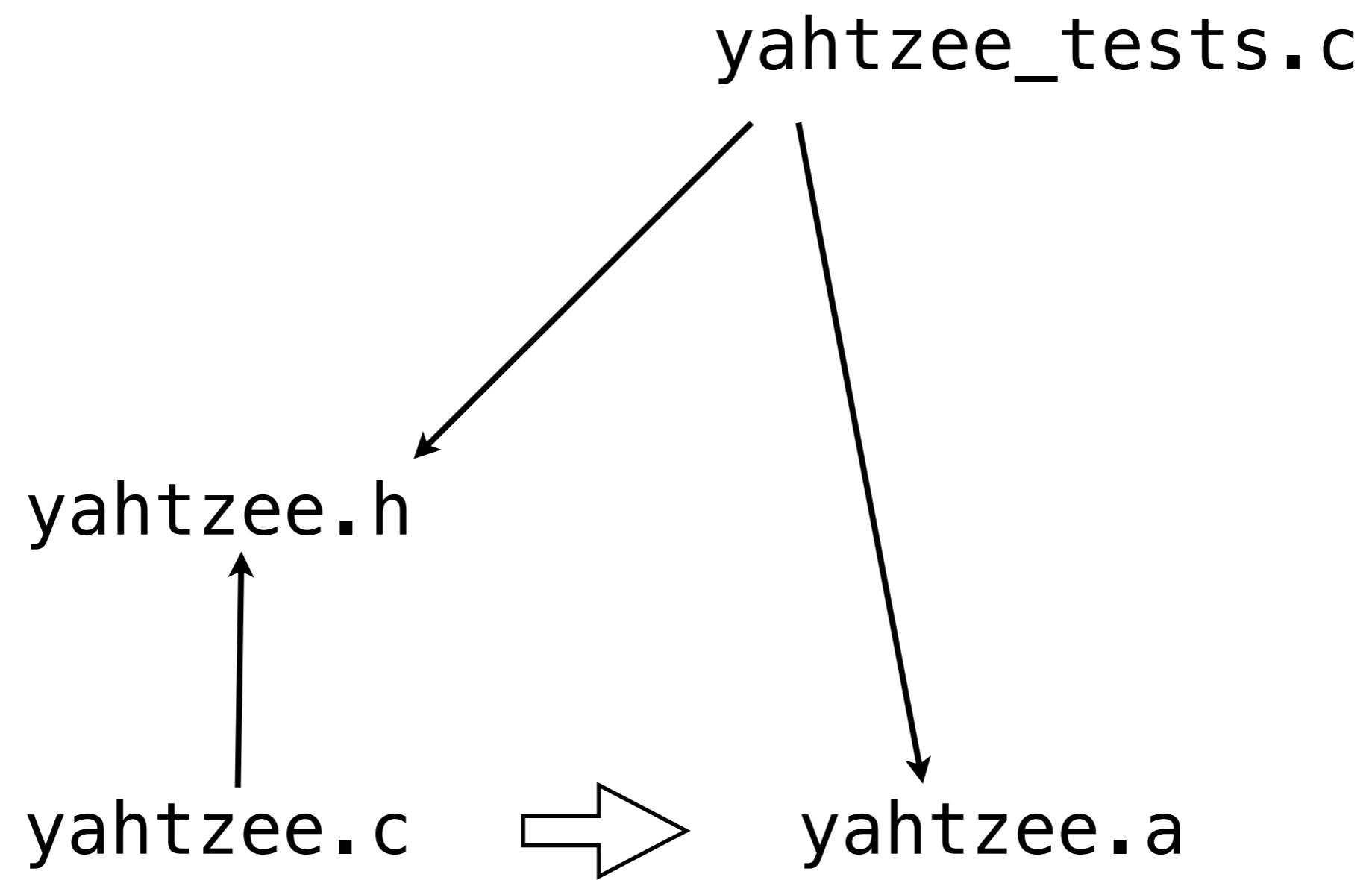
©1962, 1990, 1996 Milton Bradley Company. All Rights Reserved.  
E6100

## LOWER SECTION

|                                      |                              |
|--------------------------------------|------------------------------|
| <b>3 of a kind</b>                   | Add Total<br>Of All Dice     |
| <b>4 of a kind</b>                   | Add Total<br>Of All Dice     |
| <b>Full House</b>                    | <b>SCORE 25</b>              |
| <b>Sm. Straight</b> Sequence<br>of 4 | <b>SCORE 30</b>              |
| <b>Lg. Straight</b> Sequence<br>of 5 | <b>SCORE 40</b>              |
| <b>YAHTZEE</b> 5 of<br>a kind        | <b>SCORE 50</b>              |
| <b>Chance</b>                        | 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.c
gcc -std=c99 -O -Wall 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.
```

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 -c -fno-strict-aliasing -Wall -Wextra yahtzee.c

```

**Fail - Fix - Pass**

```
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
```

Our first unit test failed. This is good! Exactly what we want. The rhythm of TDD is : fail, fix, pass... fail, fix, pass

**Fail - Fix - Pass**

```
gcc -c yahtzee_tests.c yahtzee_tests.o yahtzee.o
```

```
./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
```

**Fail - Fix - Pass**

```
$ make check
gcc -std=c99 -O yahtzee_tests.c
gcc -o yahtzee_tests yahtzee_tests.o
./yahtzee_tests
Yahtzee tests OK
```

All tests are OK!

yahtzee\_test.c

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

int main(void)
{
    assert(6*7 == 42);

    puts("Yahtzee tests OK");
}
```

**Fail - Fix - Pass**

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");
}
```

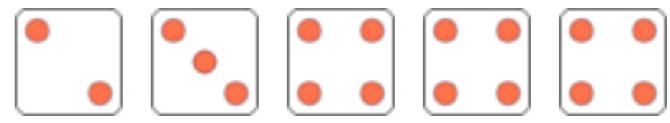
Fail - Fix - Pass

Fail - Fix - Pass

```
$ make check
gcc -std=c99 -O yahtzee_tests.c
gcc -o yahtzee_tests yahtzee_tests.o
./yahtzee_tests
Yahtzee tests OK
```

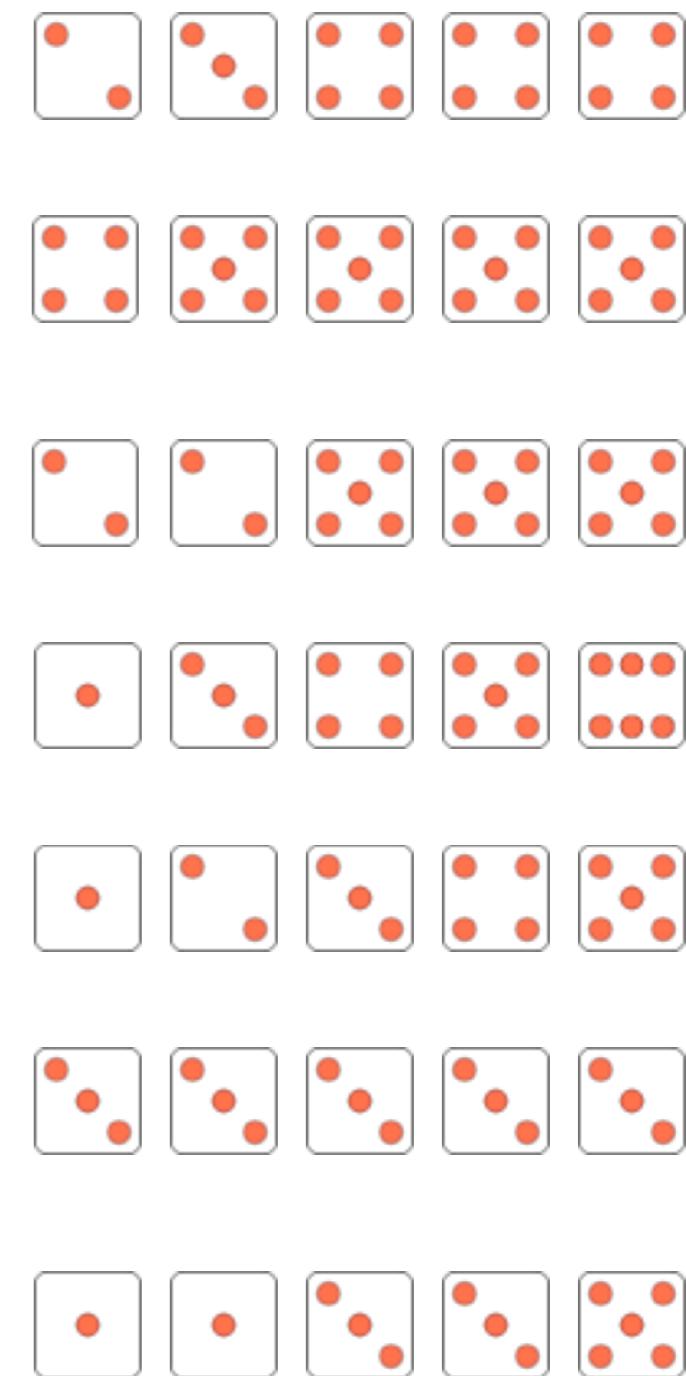
All tests are OK!

## LOWER SECTION

|                                   |                                      |   |
|-----------------------------------|--------------------------------------|---|
| <b>3 of a kind</b>                | <b>Add Total<br/>Of All Dice</b>     |    |
| <b>4 of a kind</b>                | <b>Add Total<br/>Of All Dice</b>     |    |
| <b>Full House</b>                 | <b>SCORE 25</b>                      |    |
| <b>Sm. Straight</b> Sequence of 4 | <b>SCORE 30</b>                      |  |
| <b>Lg. Straight</b> Sequence of 5 | <b>SCORE 40</b>                      |  |
| <b>YAHTZEE</b> 5 of a kind        | <b>SCORE 50</b>                      |  |
| <b>Chance</b>                     | <b>Score Total<br/>Of All 5 Dice</b> |  |

## LOWER SECTION

|                                      |                              |
|--------------------------------------|------------------------------|
| <b>3 of a kind</b>                   | Add Total<br>Of All Dice     |
| <b>4 of a kind</b>                   | Add Total<br>Of All Dice     |
| <b>Full House</b>                    | <b>SCORE 25</b>              |
| <b>Sm. Straight</b> Sequence<br>of 4 | <b>SCORE 30</b>              |
| <b>Lg. Straight</b> Sequence<br>of 5 | <b>SCORE 40</b>              |
| <b>YAHTZEE</b> 5 of<br>a kind        | <b>SCORE 50</b>              |
| <b>Chance</b>                        | 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
#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;  
}
```

Fail - **Fix** - Pass

yahtzee.h

```
int score_three_of_a_kind(const int dice[5]);
```

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?

```
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;  
}
```

Fail - **Fix** - Pass

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?

```
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;
}
```

Fail - **Fix** - Pass

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?

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_
```

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_
```

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

Yahtzee tests OK

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);
}
```

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

Yahtzee tests OK

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);
}
```

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

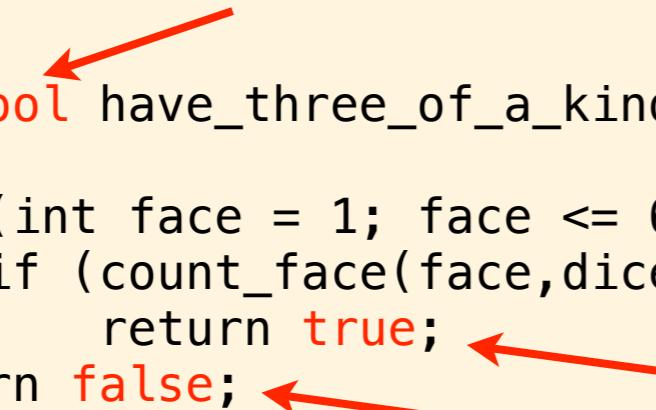
```
#include "yahtzee.h"
#include <stddef.h>

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

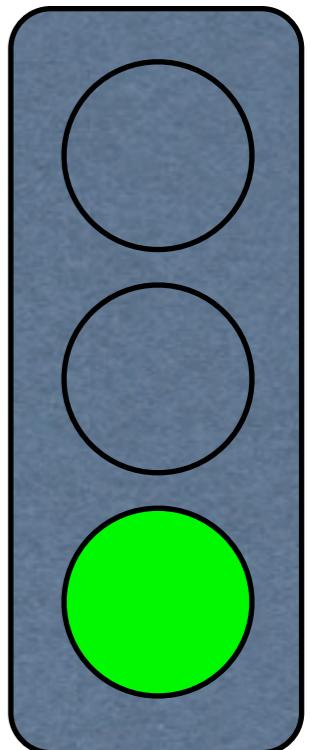
```
#include "yahtzee.h"
#include <stddef.h>
#include <stdbool.h>

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);
```



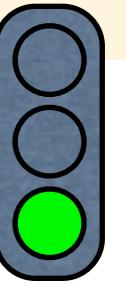
## yahtzee\_test.c

```
...
    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

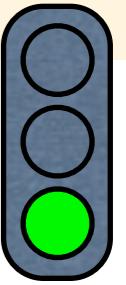


## yahtzee\_test.c

```
...
    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



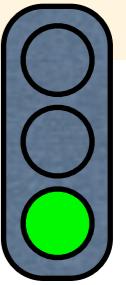
## yahtzee\_test.c

```
...
    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

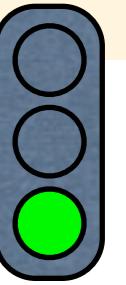


Looking good! Looking good!

### 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



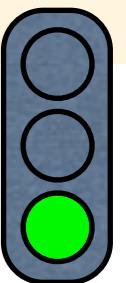
Looking good! Looking good!

### 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);
```



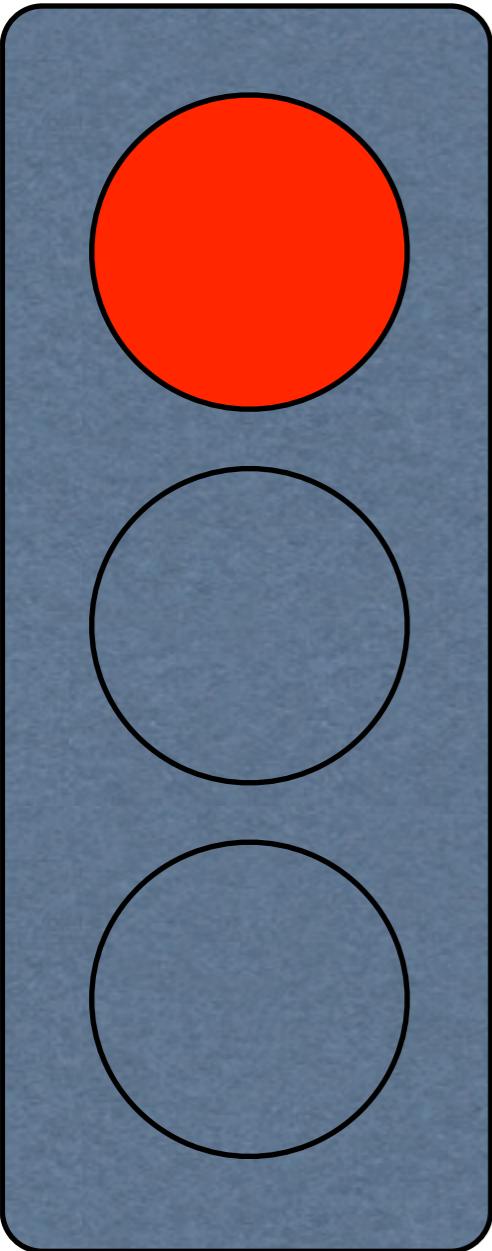
Yahtzee tests OK



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);
```

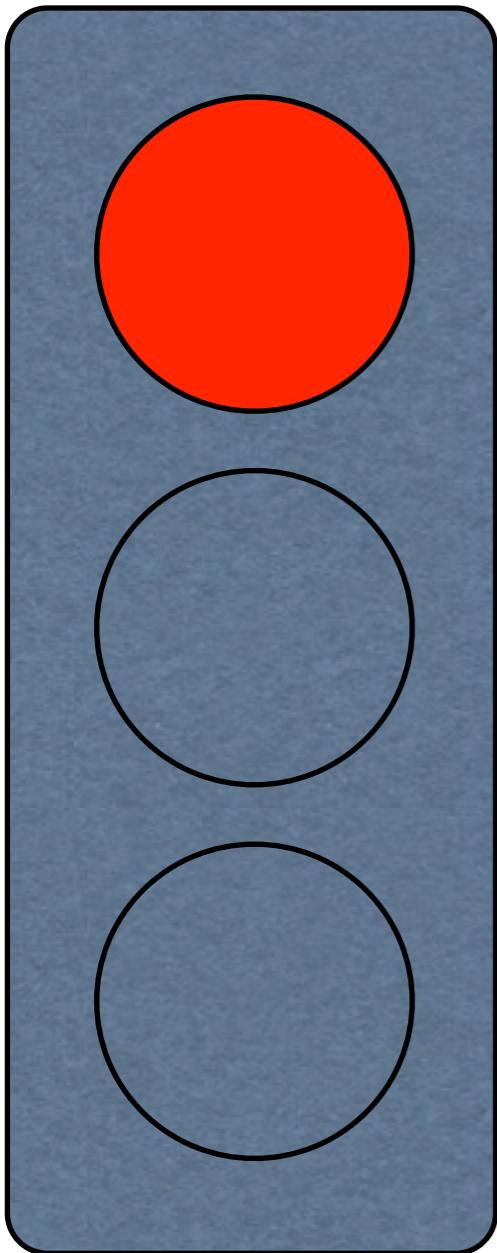


Looking good! Looking good!

### 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);
```

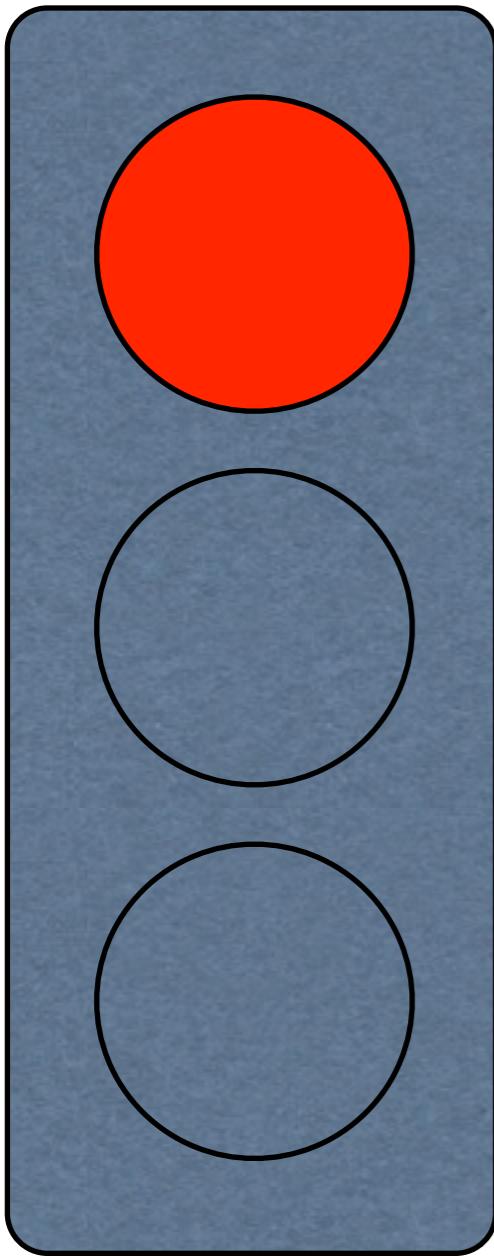
Assertion failed: (score\_three\_of\_a\_kind((int[5]){6,1,6,6,6}) == 18+7)



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)

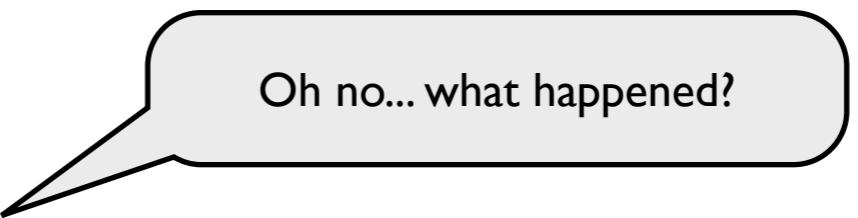


yahtzee\_test.c

```
...
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)

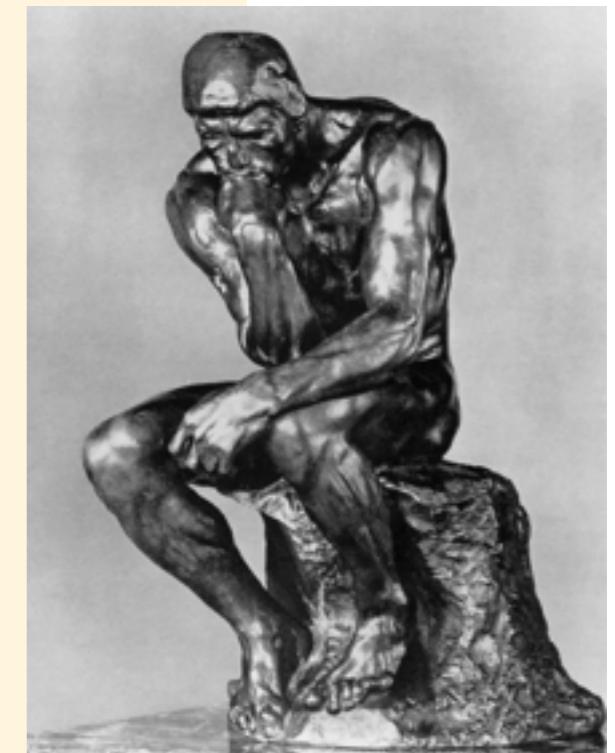
## yahtzee.c

```
...
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;
}
```

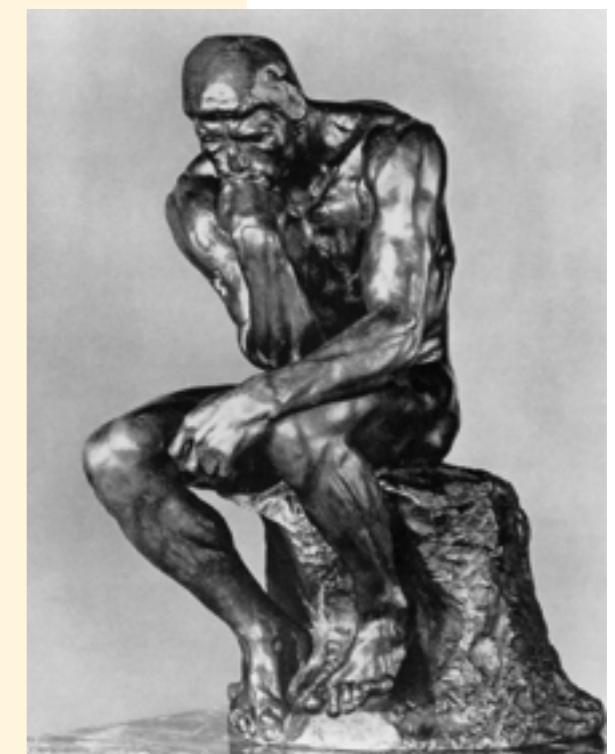
```
static bool have_three_of_a_kind(const int dice[5])
{
```

```
    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;
}
```

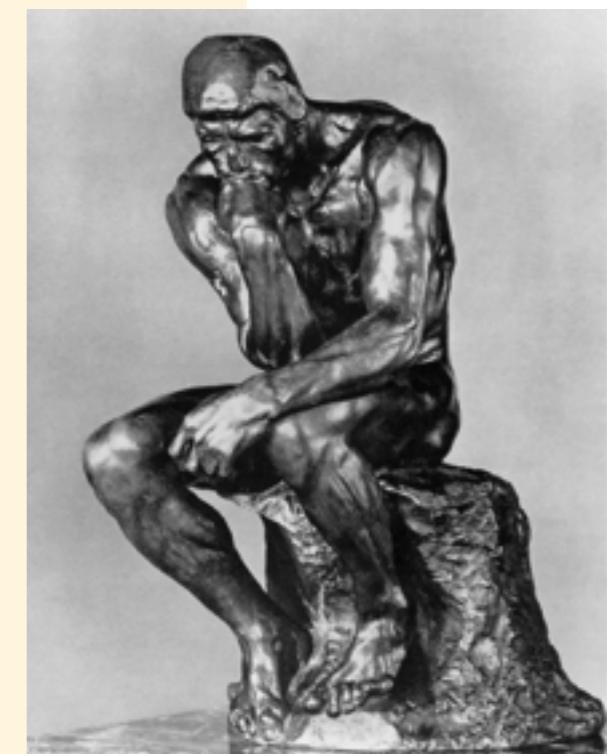
```
static bool have_three_of_a_kind(const int dice[5])
{
```

```
    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])
{
```

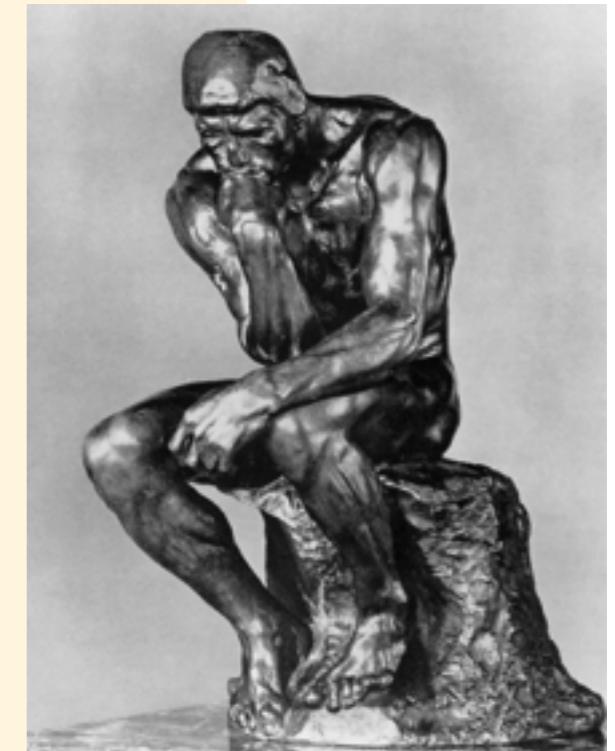
```
    for (int face = 1; face <= 6; face++)
        if (count_face(face,dice) == 3)
            return true;
    return false;
}
```

```
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

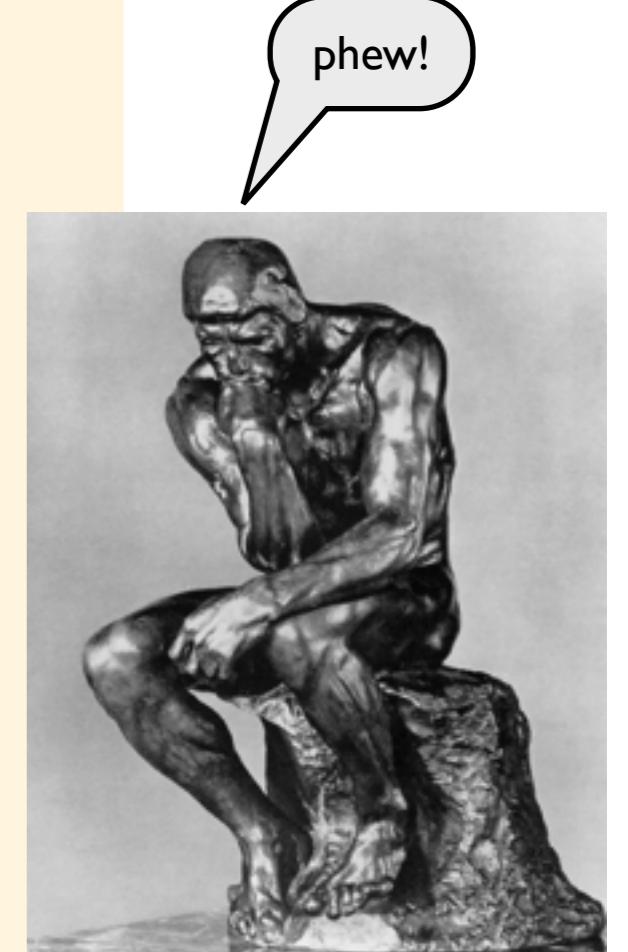
## yahtzee.c

```
...
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;
}
```



Yahtzee tests OK

## yahtzee\_test.c

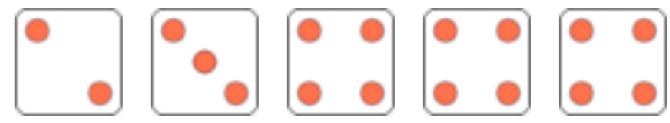
```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 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

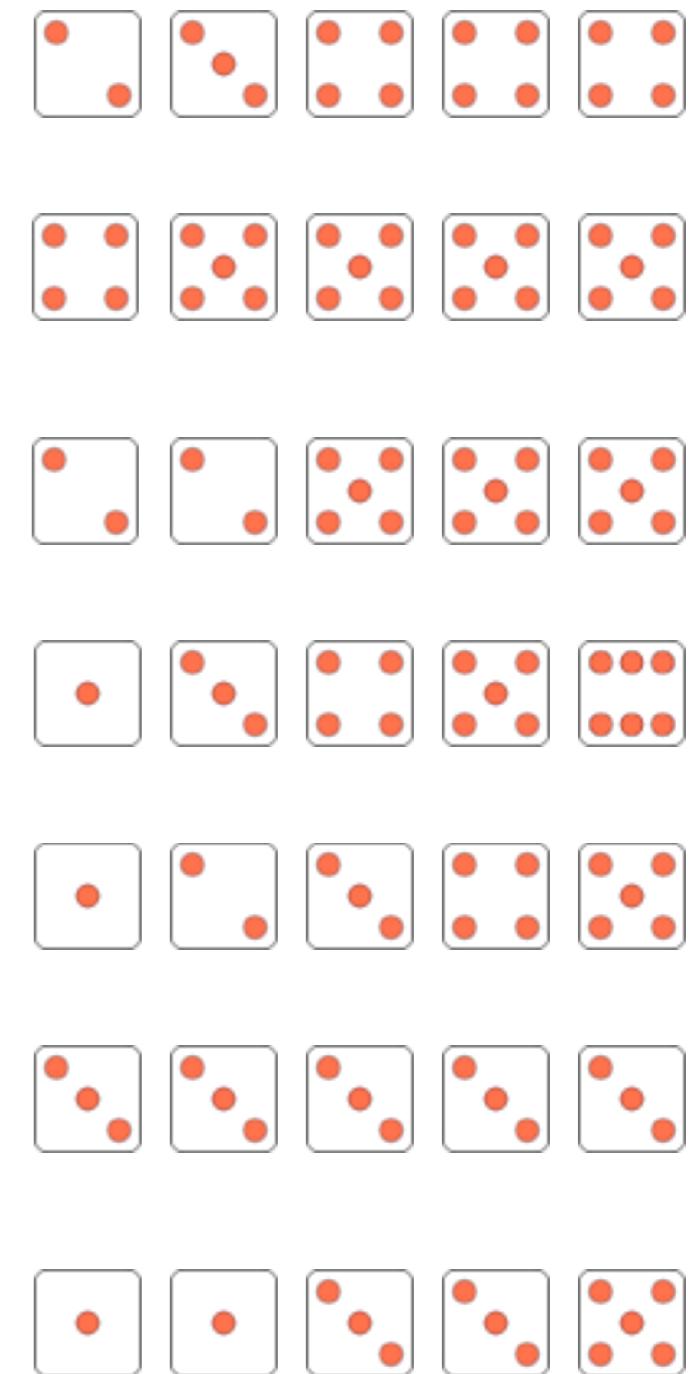
```
...
assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 18+2);
assert(score_three_of_a_kind((int[5]){6,1,6,6,6}) == 18+7);
```

## LOWER SECTION

|                                      |                                      |   |
|--------------------------------------|--------------------------------------|---|
| <b>3 of a kind</b>                   | <b>Add Total<br/>Of All Dice</b>     |    |
| <b>4 of a kind</b>                   | <b>Add Total<br/>Of All Dice</b>     |    |
| <b>Full House</b>                    | <b>SCORE 25</b>                      |    |
| <b>Sm. Straight</b> Sequence<br>of 4 | <b>SCORE 30</b>                      |  |
| <b>Lg. Straight</b> Sequence<br>of 5 | <b>SCORE 40</b>                      |  |
| <b>YAHTZEE</b> 5 of<br>a kind        | <b>SCORE 50</b>                      |  |
| <b>Chance</b>                        | <b>Score Total<br/>Of All 5 Dice</b> |  |

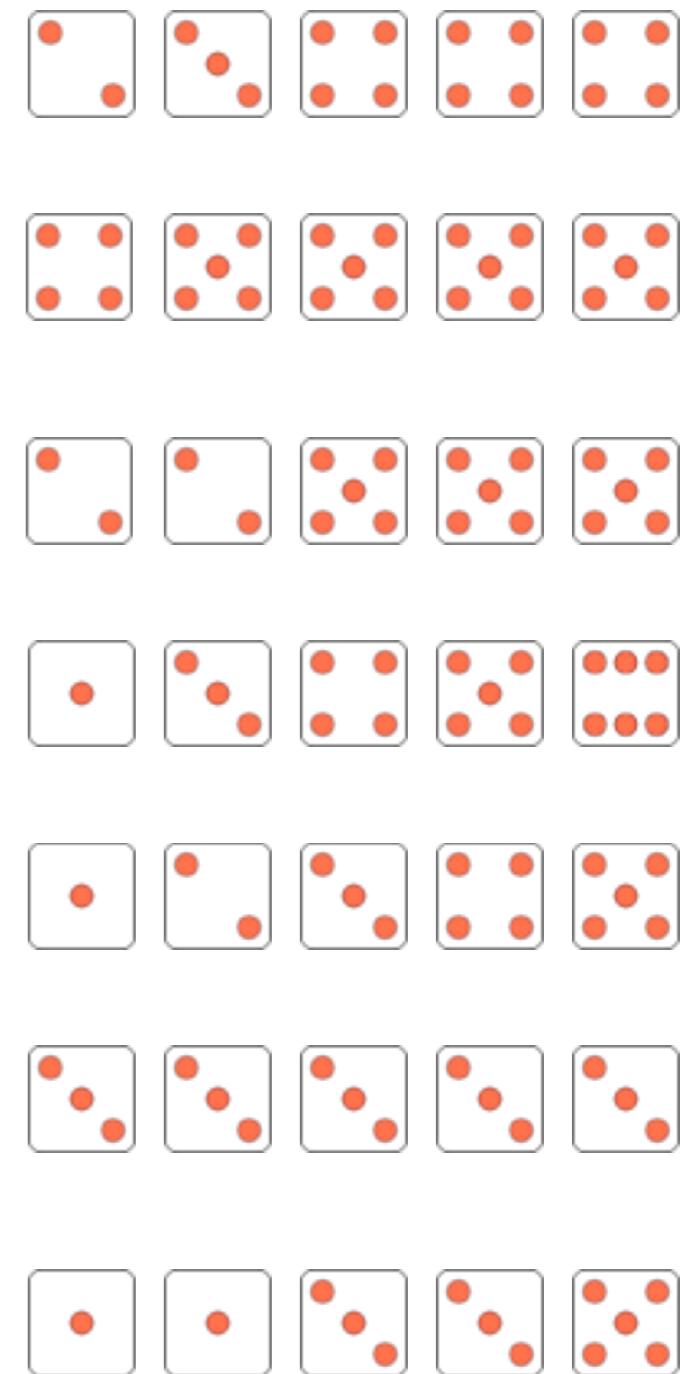
## LOWER SECTION

|                                      |                              |
|--------------------------------------|------------------------------|
| <b>3 of a kind</b> ✓                 | Add Total<br>Of All Dice     |
| <b>4 of a kind</b>                   | Add Total<br>Of All Dice     |
| <b>Full House</b>                    | <b>SCORE 25</b>              |
| <b>Sm. Straight</b> Sequence<br>of 4 | <b>SCORE 30</b>              |
| <b>Lg. Straight</b> Sequence<br>of 5 | <b>SCORE 40</b>              |
| <b>YAHTZEE</b> 5 of<br>a kind        | <b>SCORE 50</b>              |
| <b>Chance</b>                        | 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 | 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 |



## LOWER SECTION

3 of a kind ✓

4 of a kind

Full House

Sm. Straight Sequence  
of 4

Lg. Straight Sequence  
of 5

YAHTZEE 5 of  
a kind

Chance

Add Total  
Of All Dice

Add Total  
Of All Dice

SCORE 25

SCORE 30

SCORE 40

SCORE 50

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

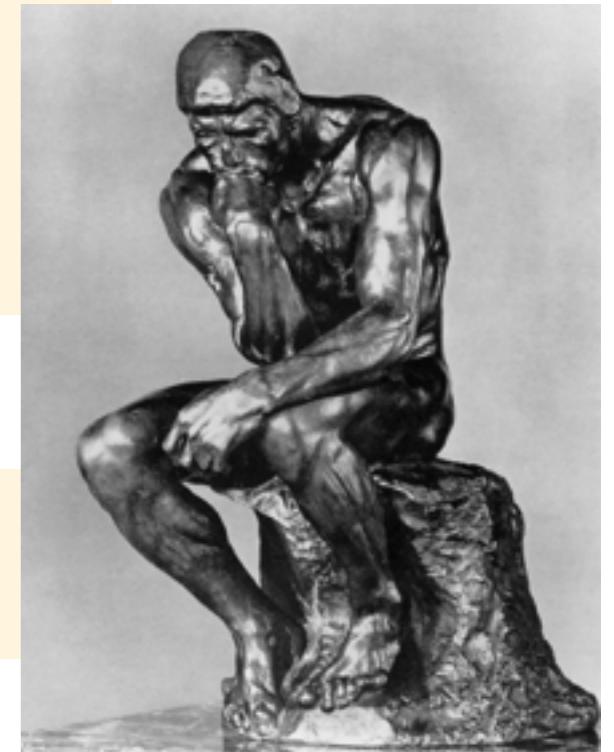
```
...
    assert(score_three_of_a_kind((int[5])\{1,1,1,2,2\}) == 3+2+2);
    assert(score_three_of_a_kind((int[5])\{1,1,1,3,4\}) == 3+3+4);
    assert(score_three_of_a_kind((int[5])\{1,2,3,4,5\}) == 0);
    assert(score_three_of_a_kind((int[5])\{5,3,5,5,2\}) == 15+5);
    assert(score_three_of_a_kind((int[5])\{1,1,6,6,6\}) == 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...

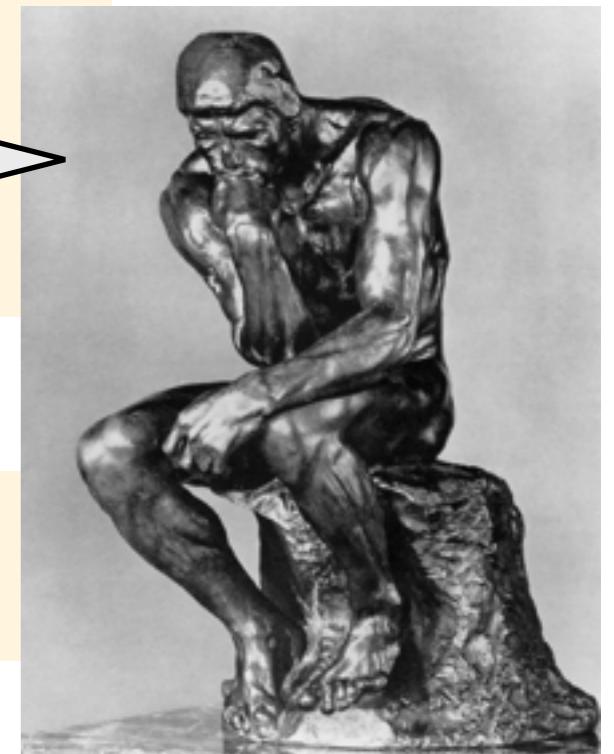
## yahtzee.h

```
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()?

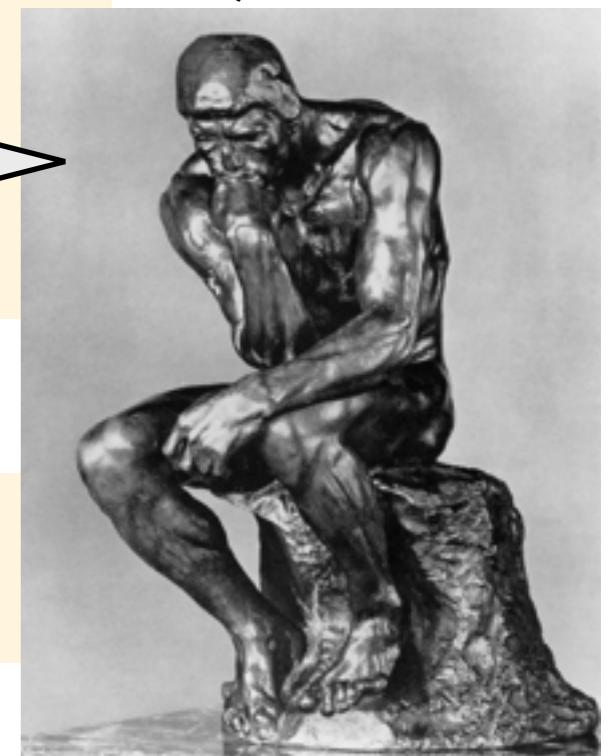
hmm...

## 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])  
{  
}  
}
```

hmm...

Perhaps we need a  
have\_at\_least\_n\_of\_a\_kind()?

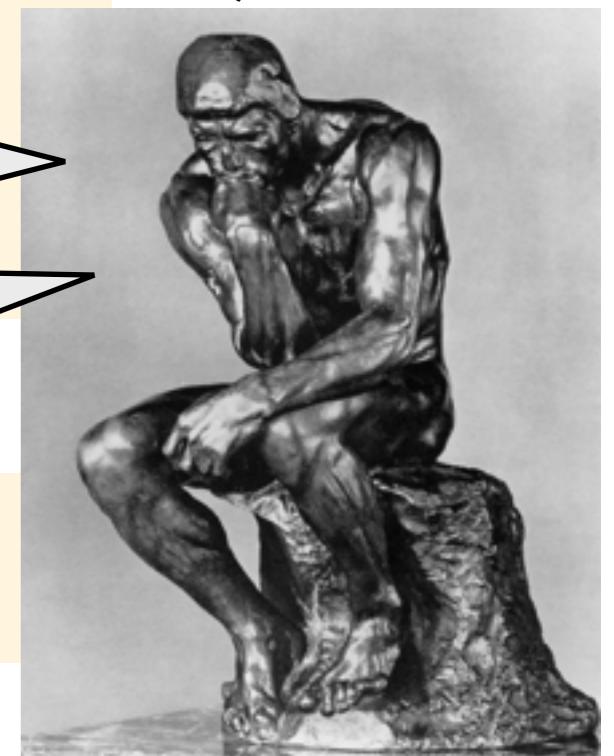
Let's go back to "green"

## 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

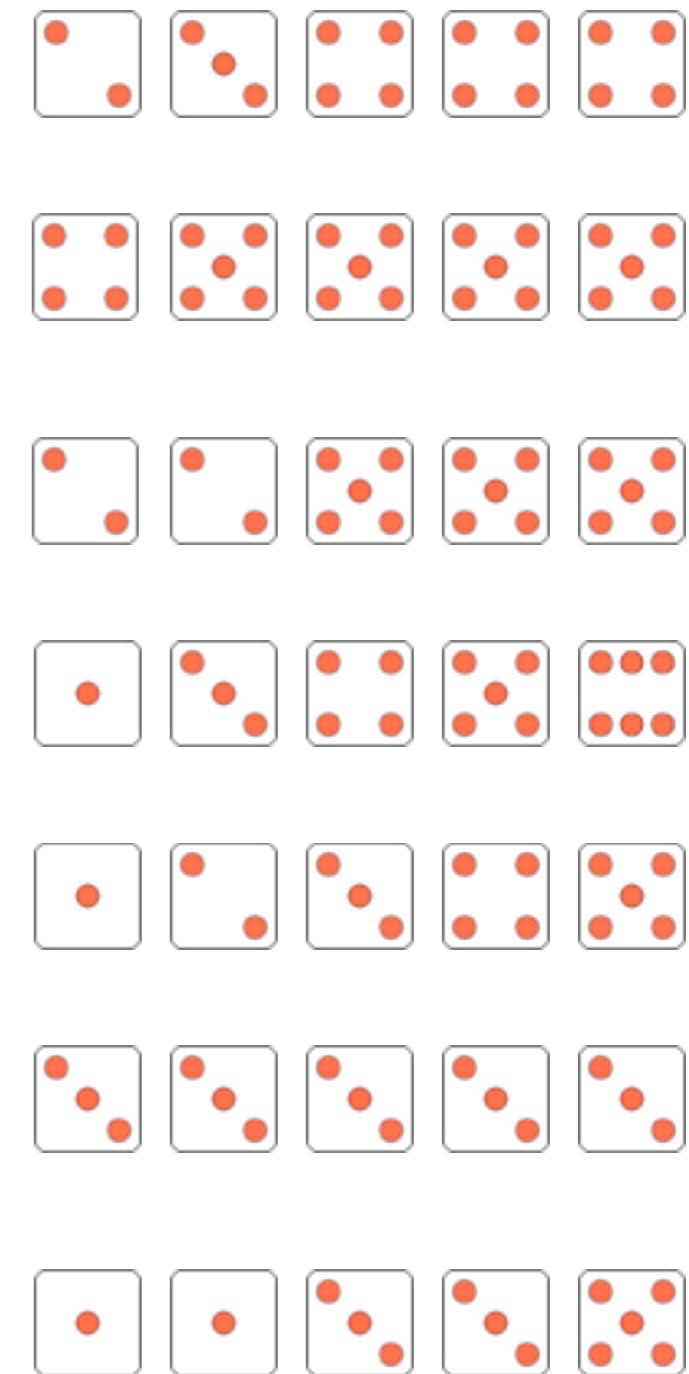
```
...
    assert(score_three_of_a_kind((int[5]){1,1,1,2,2}) == 3+2+2);
    assert(score_three_of_a_kind((int[5]){1,1,1,3,4}) == 3+3+4);
    assert(score_three_of_a_kind((int[5]){1,2,3,4,5}) == 0);
    assert(score_three_of_a_kind((int[5]){5,3,5,5,2}) == 15+5);
    assert(score_three_of_a_kind((int[5]){1,1,6,6,6}) == 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

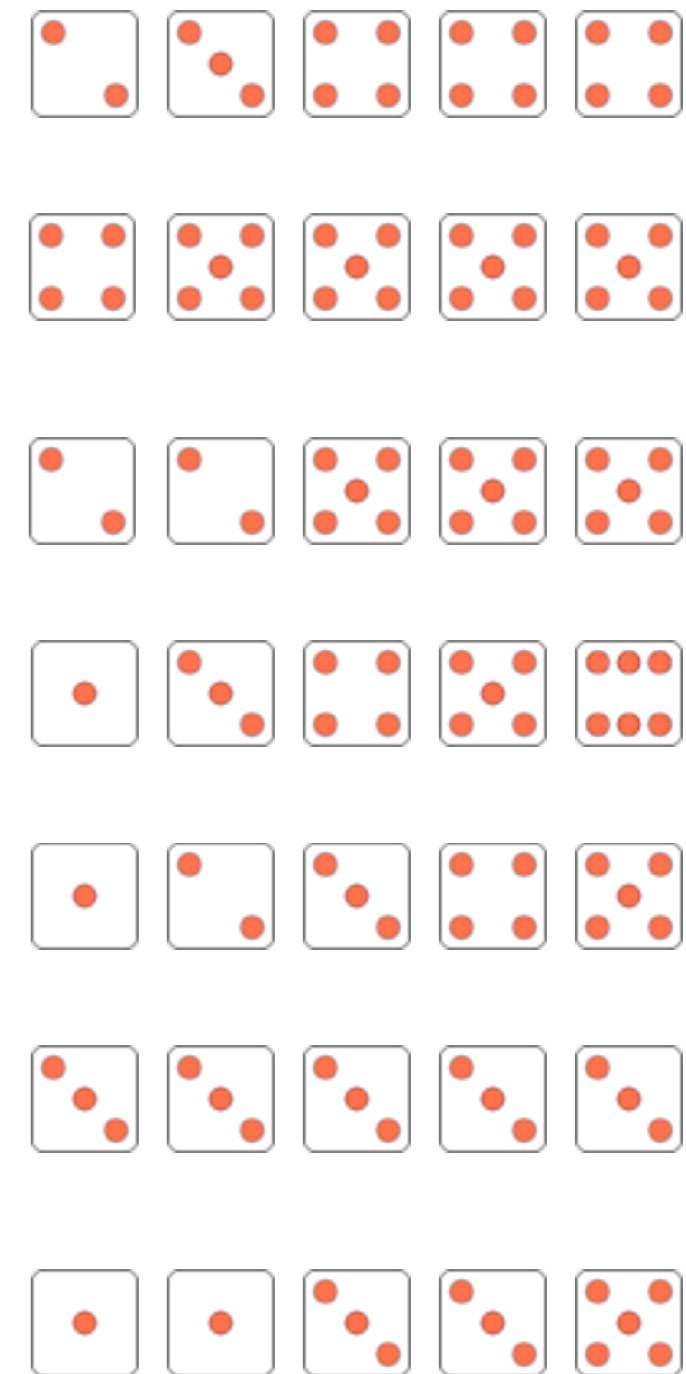
## LOWER SECTION

|                                      |                              |
|--------------------------------------|------------------------------|
| <b>3 of a kind</b> ✓                 | Add Total<br>Of All Dice     |
| <b>4 of a kind</b>                   | Add Total<br>Of All Dice     |
| <b>Full House</b>                    | <b>SCORE 25</b>              |
| <b>Sm. Straight</b> Sequence<br>of 4 | <b>SCORE 30</b>              |
| <b>Lg. Straight</b> Sequence<br>of 5 | <b>SCORE 40</b>              |
| <b>YAHTZEE</b> 5 of<br>a kind        | <b>SCORE 50</b>              |
| <b>Chance</b>                        | Score Total<br>Of All 5 Dice |



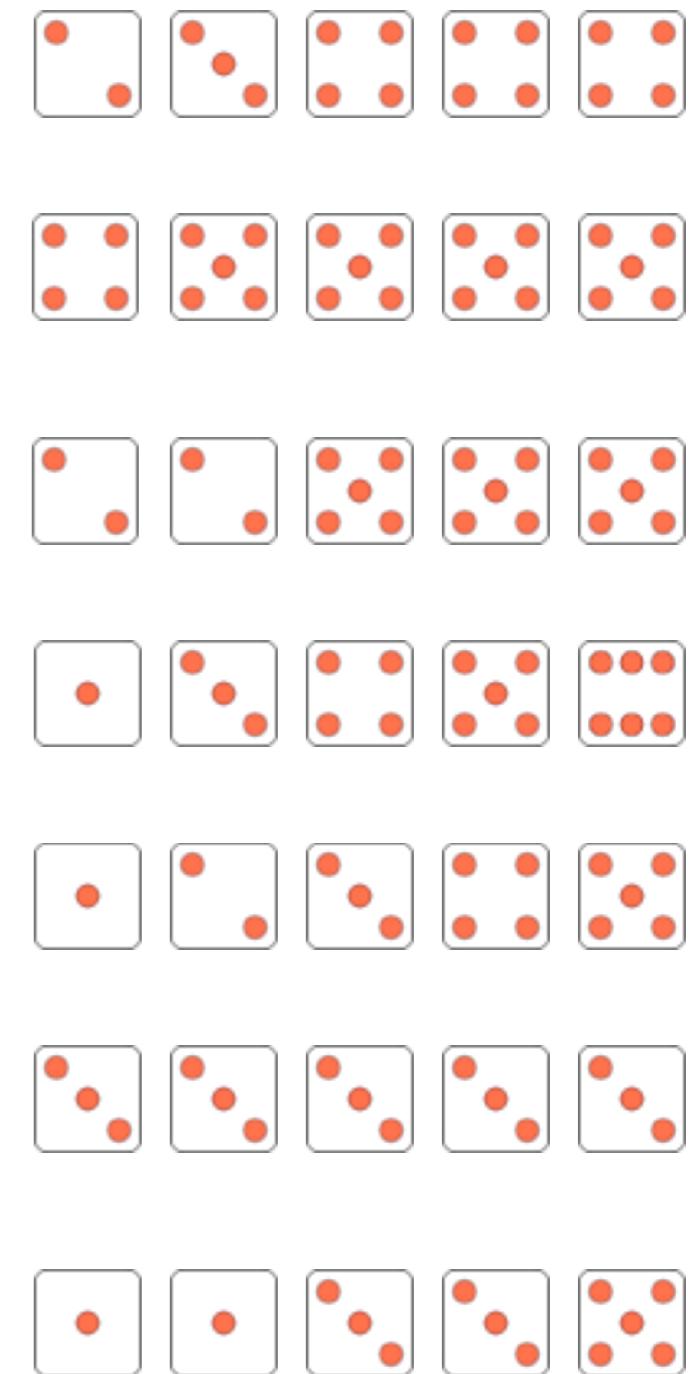
## LOWER SECTION

|              |                  |                              |
|--------------|------------------|------------------------------|
| 3 of a kind  |                  | Add Total<br>Of All Dice     |
| 4 of a kind  |                  | Add Total<br>Of All Dice     |
| Full House   |                  | <b>SCORE 25</b>              |
| Sm. Straight | Sequence<br>of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence<br>of 5 | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of<br>a kind   | <b>SCORE 50</b>              |
| Chance       |                  | Score Total<br>Of All 5 Dice |



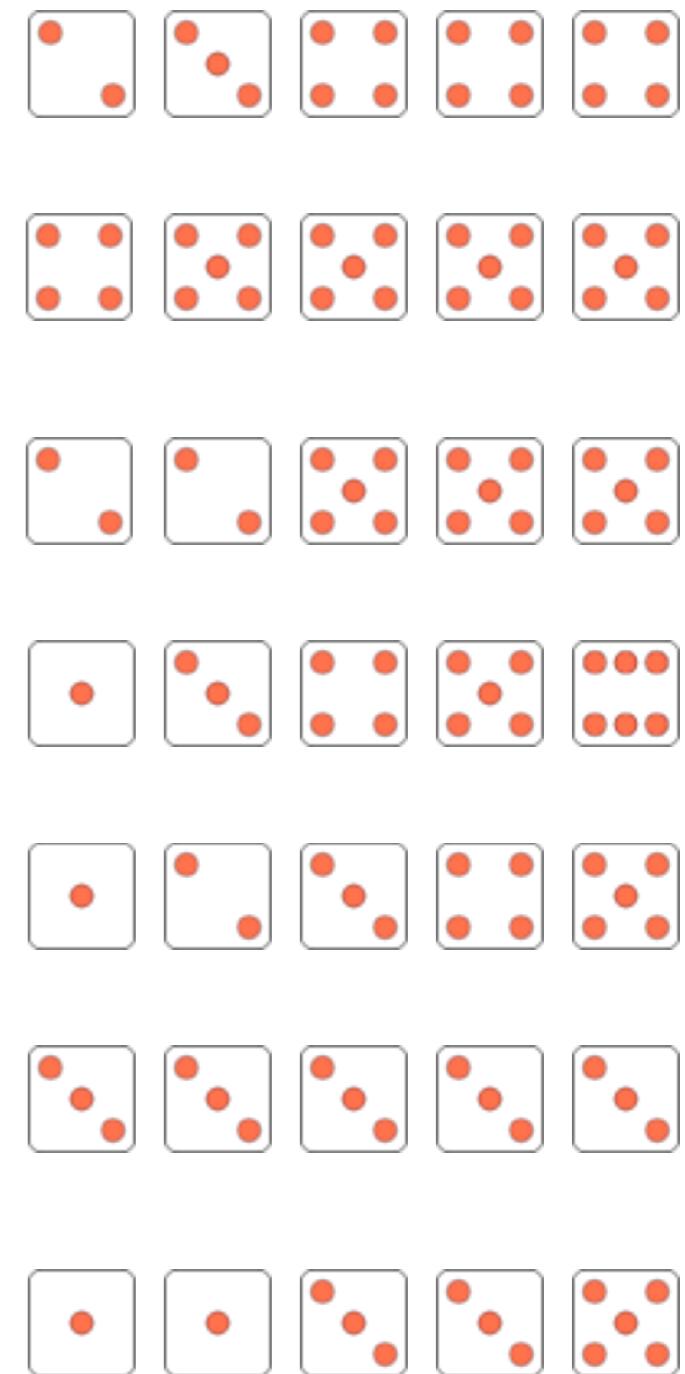
## LOWER SECTION

|              |                  |                              |
|--------------|------------------|------------------------------|
| 3 of a kind  |                  | Add Total<br>Of All Dice     |
| 4 of a kind  |                  | Add Total<br>Of All Dice     |
| Full House   |                  | <b>SCORE 25</b>              |
| Sm. Straight | Sequence<br>of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence<br>of 5 | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of<br>a kind   | <b>SCORE 50</b>              |
| 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 | 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`

`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]);
```

Fail - Fix - Pass

## 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 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

Fix?

```
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]);
```

Fail - Fix - Pass

## 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;
}
```

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

Fail - Fix - Pass

```
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;
}
```

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

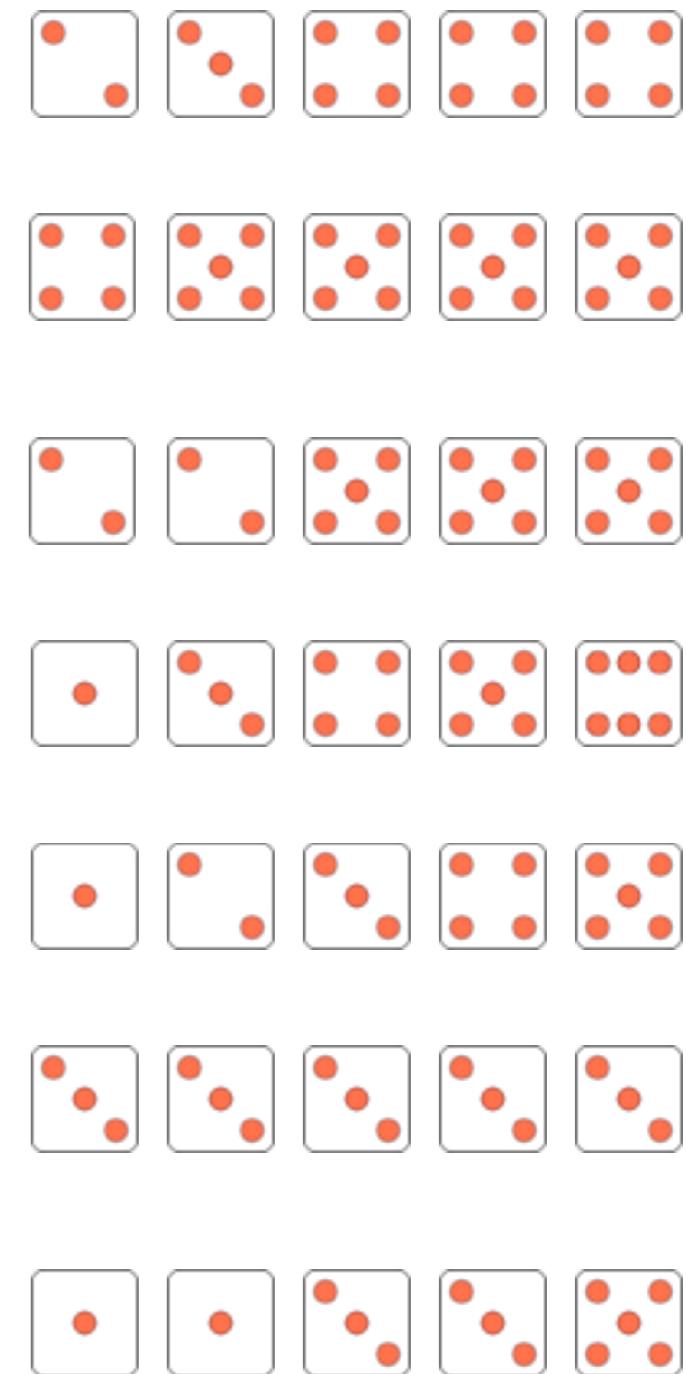
Fail - Fix - **Pass**

```
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

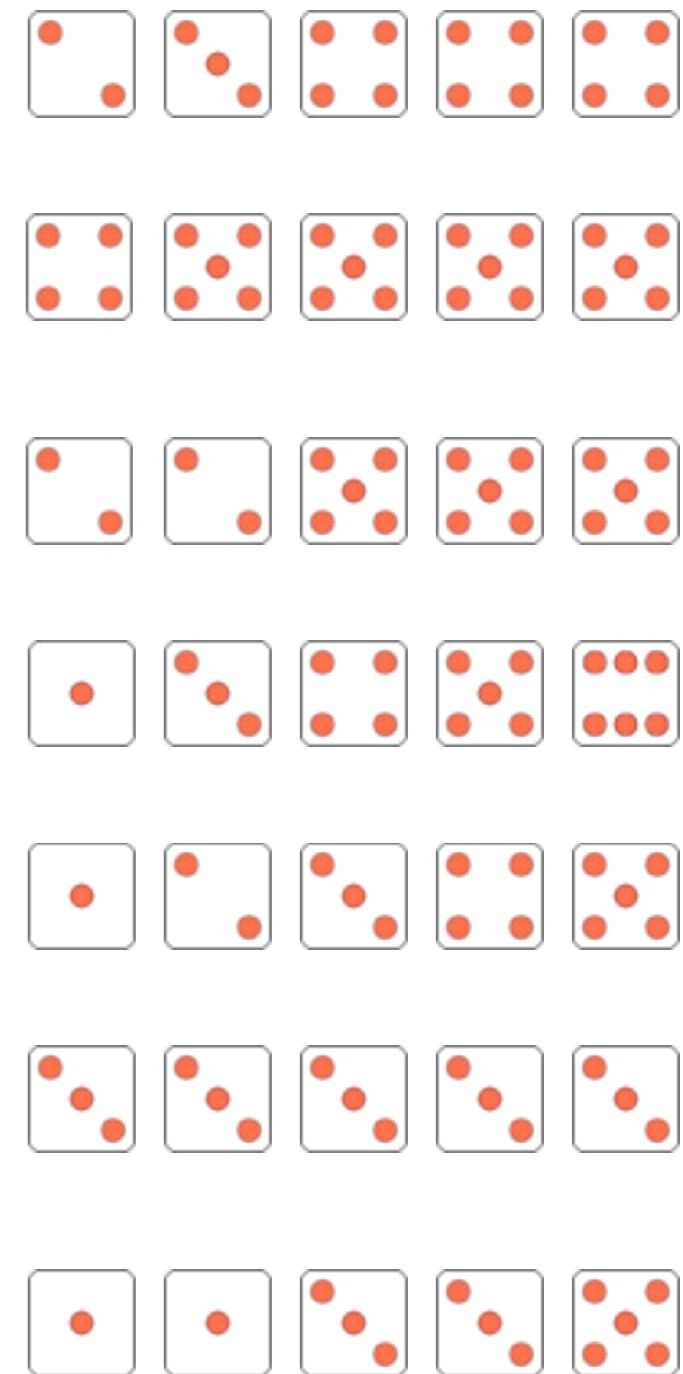
## LOWER SECTION

|              |                  |                              |
|--------------|------------------|------------------------------|
| 3 of a kind  |                  | Add Total<br>Of All Dice     |
| 4 of a kind  |                  | Add Total<br>Of All Dice     |
| Full House   |                  | <b>SCORE 25</b>              |
| Sm. Straight | Sequence<br>of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence<br>of 5 | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of<br>a kind   | <b>SCORE 50</b>              |
| Chance       |                  | Score Total<br>Of All 5 Dice |



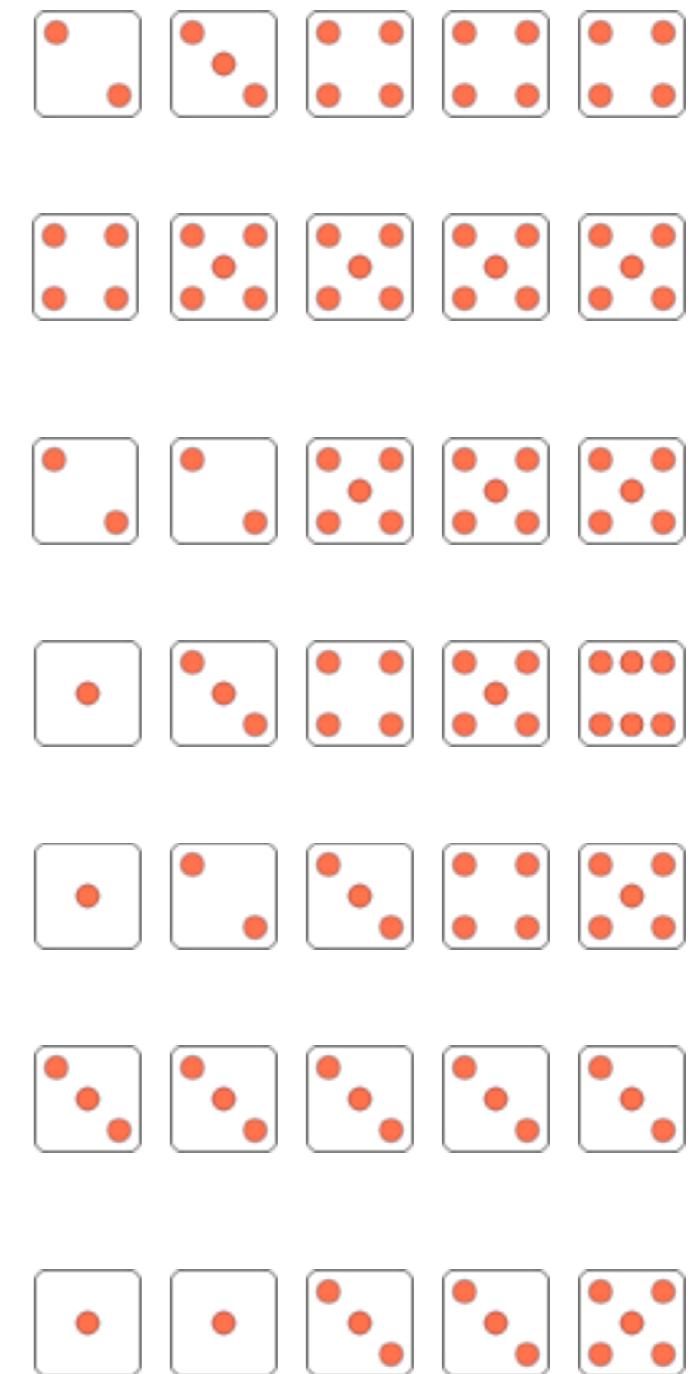
## LOWER SECTION

|              |                  |                              |
|--------------|------------------|------------------------------|
| 3 of a kind  |                  | Add Total<br>Of All Dice     |
| 4 of a kind  |                  | Add Total<br>Of All Dice     |
| Full House   |                  | <b>SCORE 25</b>              |
| Sm. Straight | Sequence<br>of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence<br>of 5 | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of<br>a kind   | <b>SCORE 50</b>              |
| Chance       |                  | Score Total<br>Of All 5 Dice |



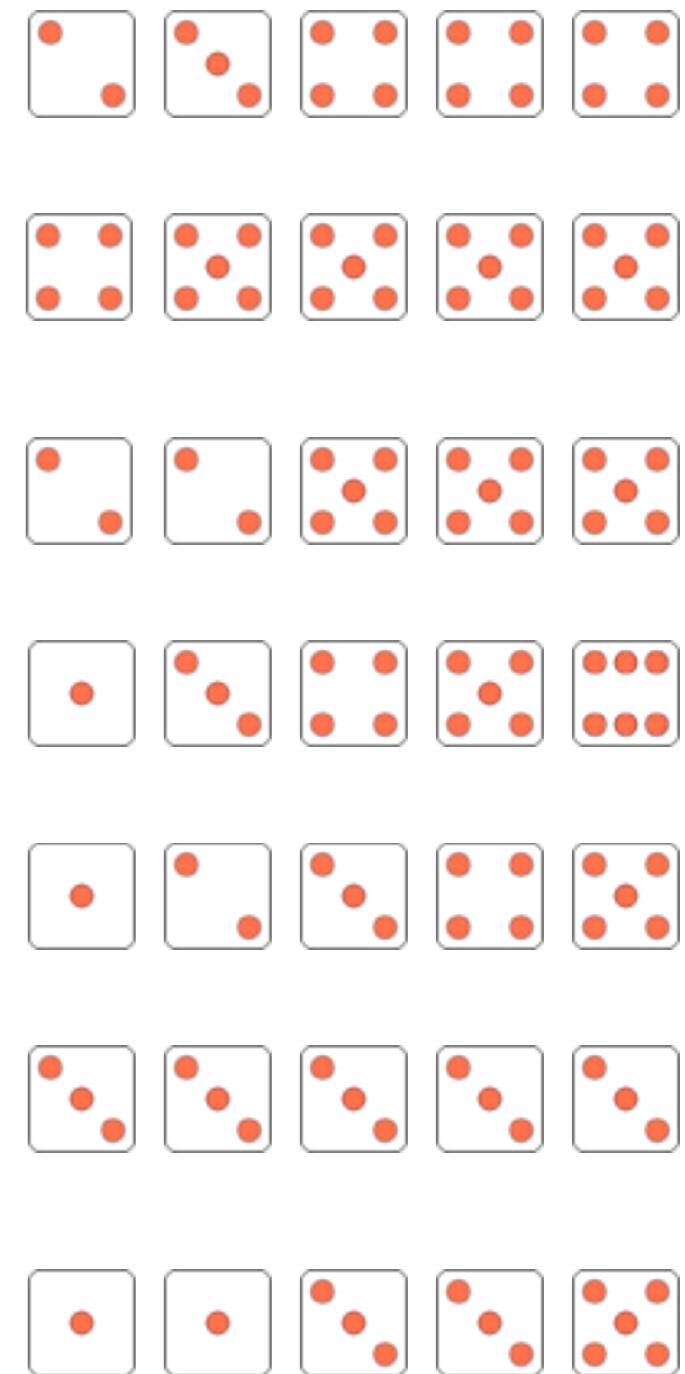
## LOWER SECTION

|              |                  |                              |
|--------------|------------------|------------------------------|
| 3 of a kind  |                  | Add Total<br>Of All Dice     |
| 4 of a kind  |                  | Add Total<br>Of All Dice     |
| Full House   |                  | <b>SCORE 25</b>              |
| Sm. Straight | Sequence<br>of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence<br>of 5 | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of<br>a kind   | <b>SCORE 50</b>              |
| 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 | 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  
implenting 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 therefore  
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 therefore  
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?

```
}
```

```
int score_small_straight(const int dice[5])  
{  
    if (have_at_least_n_in_a_row(4,dice))  
        return 30;  
    return 0;  
}
```

Use your head and think for a while...  
TDD is **not** about not thinking!

## 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?

```
}
```

```
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

Fail - Fix - Pass

```
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;
}
```

Fail - **Fix** - Pass

## yahtzee\_test.c

Fail - Fix - Pass

```
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;
}
```

Fail - **Fix** - Pass

```
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);
```

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;
}
```

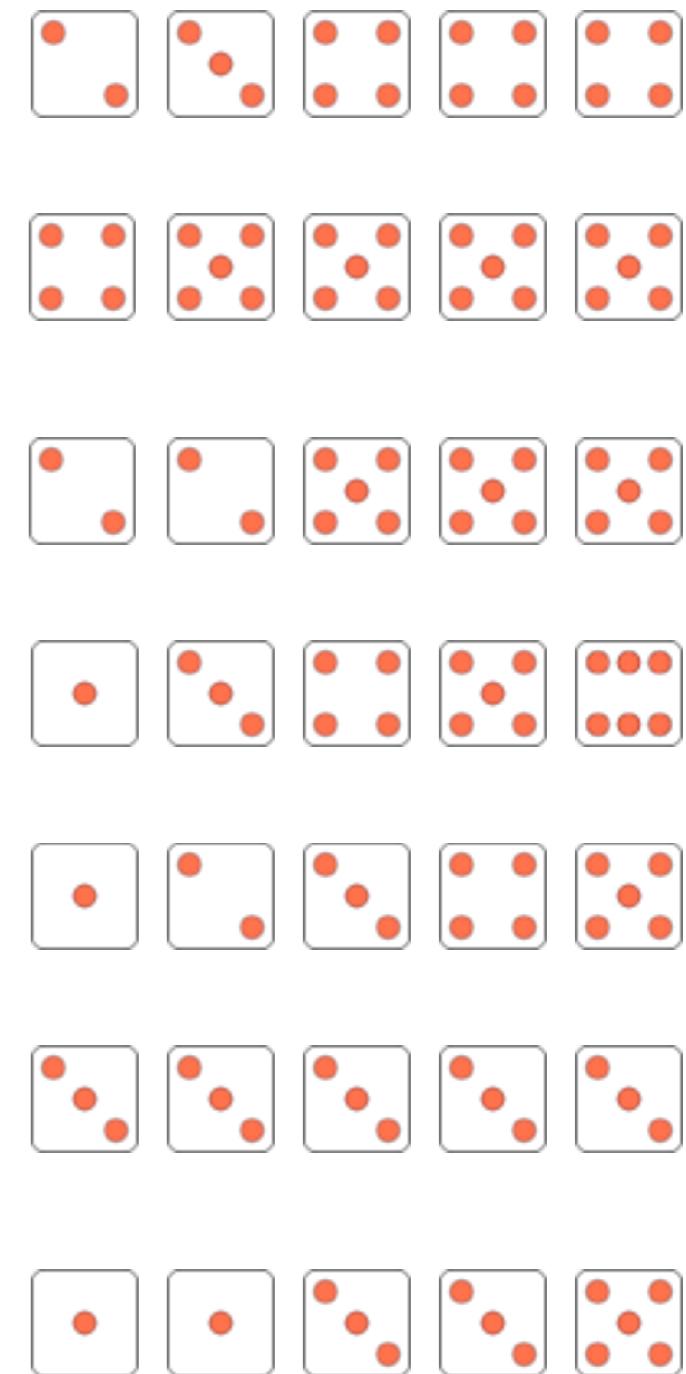
## yahtzee\_test.c

```
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

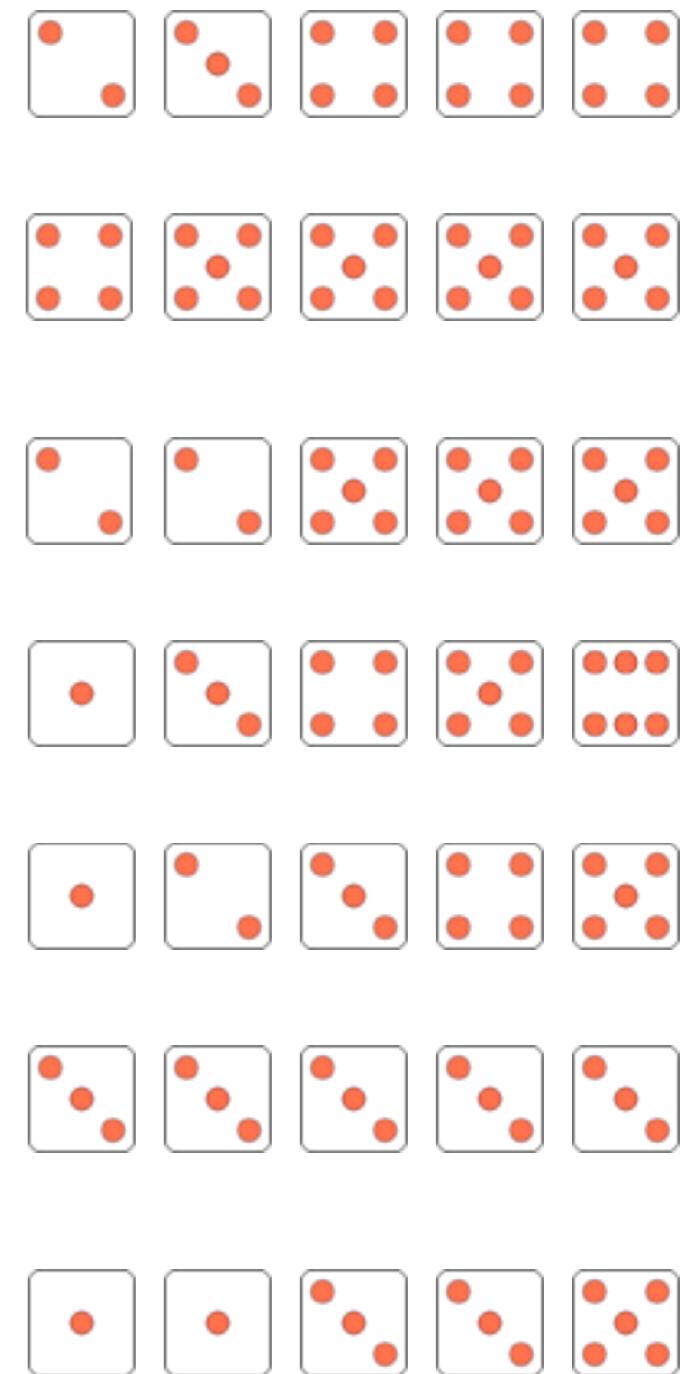
## LOWER SECTION

|              |                  |                              |
|--------------|------------------|------------------------------|
| 3 of a kind  |                  | Add Total<br>Of All Dice     |
| 4 of a kind  |                  | Add Total<br>Of All Dice     |
| Full House   |                  | <b>SCORE 25</b>              |
| Sm. Straight | Sequence<br>of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence<br>of 5 | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of<br>a kind   | <b>SCORE 50</b>              |
| Chance       |                  | Score Total<br>Of All 5 Dice |



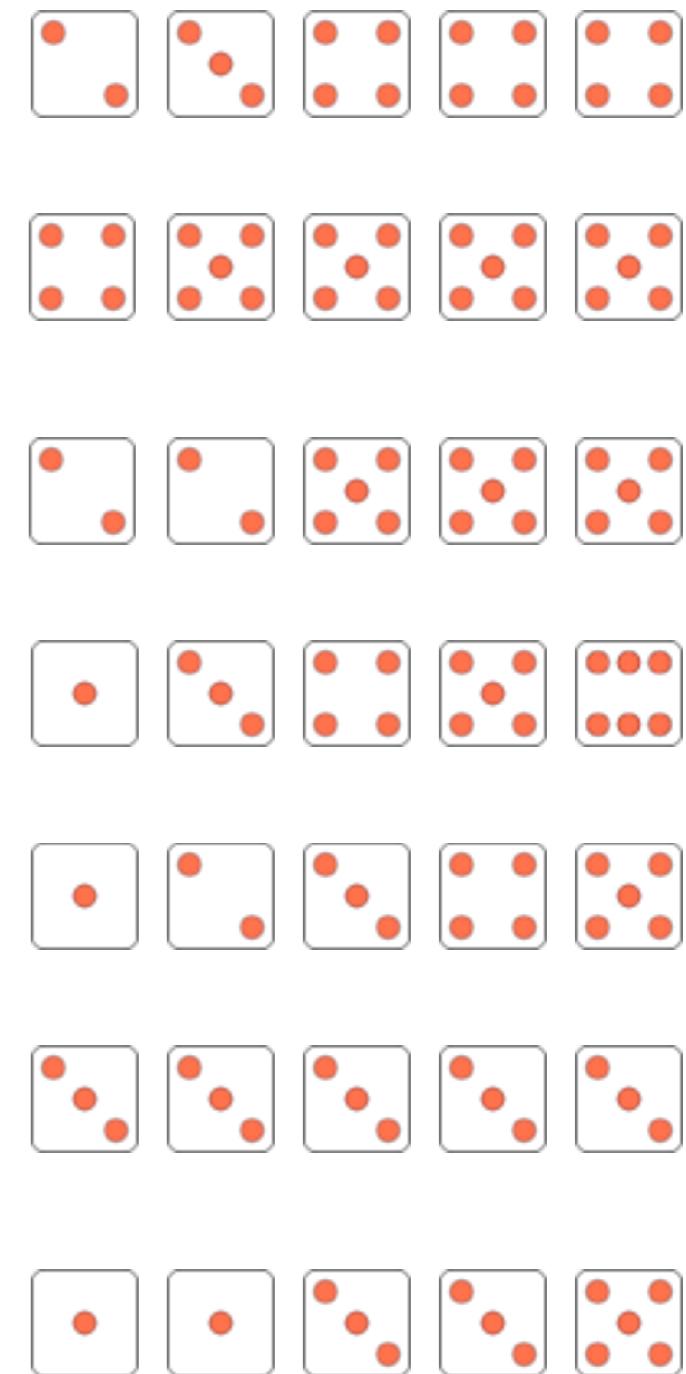
## LOWER SECTION

|              |               |                              |
|--------------|---------------|------------------------------|
| 3 of a kind  |               | Add Total<br>Of All Dice     |
| 4 of a kind  |               | Add Total<br>Of All Dice     |
| Full House   |               | <b>SCORE 25</b>              |
| Sm. Straight | Sequence of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence of 5 | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of a kind   | <b>SCORE 50</b>              |
| 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   |                      | <b>SCORE 25</b>              |
| Sm. Straight | <br>Sequence<br>of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence<br>of 5     | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of<br>a kind       | <b>SCORE 50</b>              |
| 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)

yahtzee.c

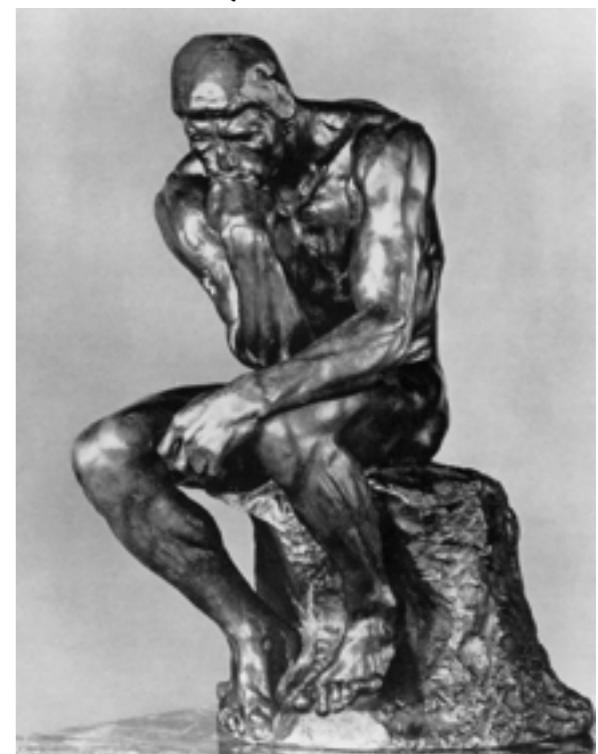
```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

Fail - **Fix** - Pass

What?  
**Inconceivable**

Assertion failed: (score\_large\_straight((int[5])\{1,2,3,4,5\}) == 30)

hmm...



yahtzee.c

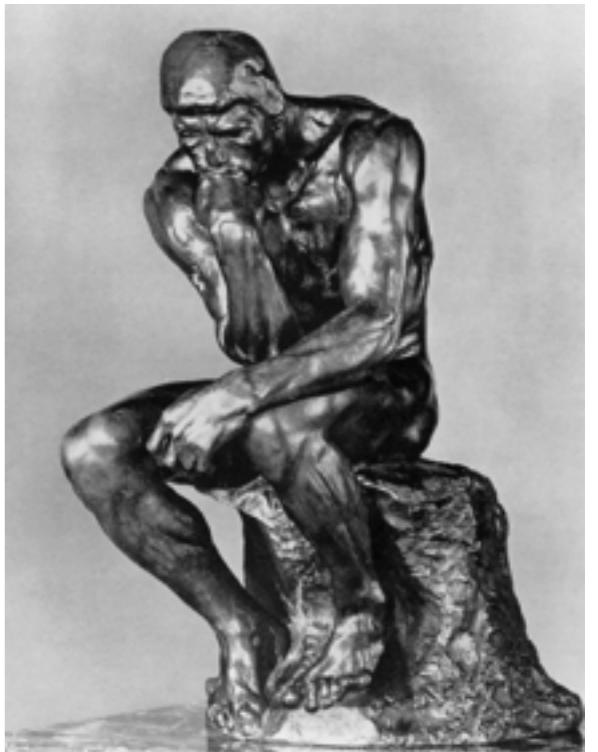
```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

```
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;
}
```

What?  
Inconceivable

Fail - Fix - Pass

hmm...



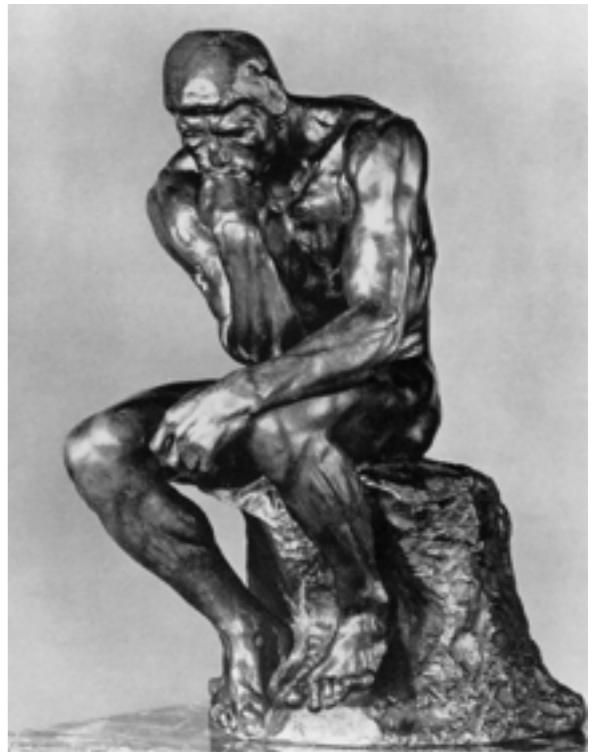
Assertion failed: (score\_large\_straight((int[5]){1,2,3,4,5}) == 30)

yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

```
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;
}
```

hmm...



Fail - Fix - Pass

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;
}
```

```
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;
}
```

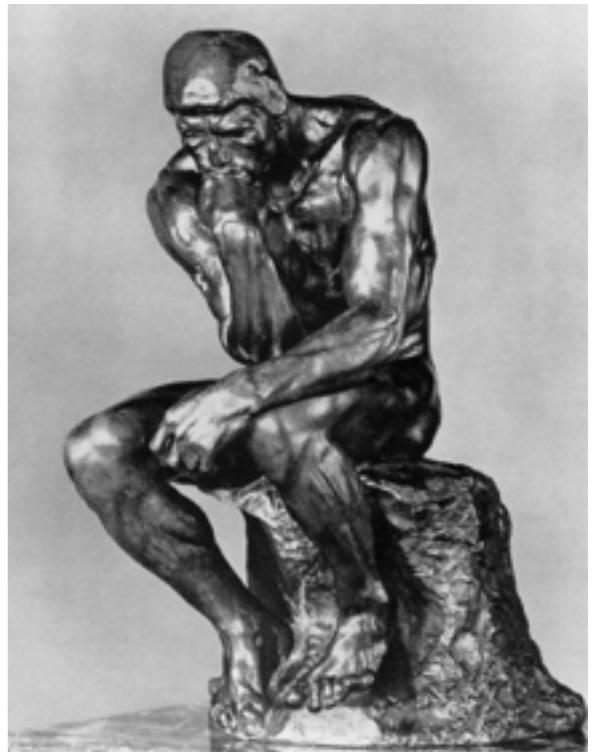
What?  
Inconceivable

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)

hmm...

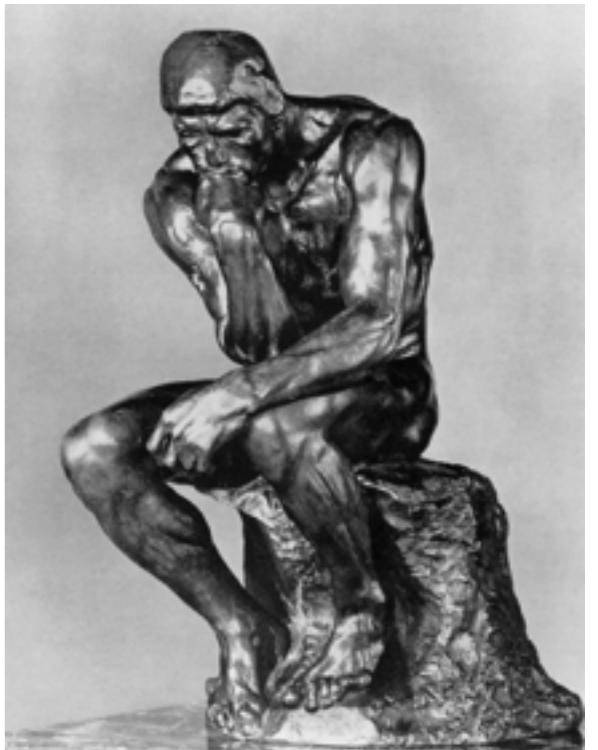


yahtzee.c

```
int score_large_straight(const int dice[5])
{
    if (have_at_least_n_in_a_row(5,dice))
        return 40;
    return 0;
}
```

```
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;
}
```

hmm...



Fail - Fix - Pass

What?  
Inconceivable

yahtzee\_test.c

```
assert(score_large_straight((int[5]){1,2,3,4,5}) == 30);
```

There is a bug in the test.

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

Fail - Fix - **Pass**

## yahtzee\_test.c

```
assert(score_large_straight((int[5]){1,2,3,4,5}) == 40);
```



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;
}
```

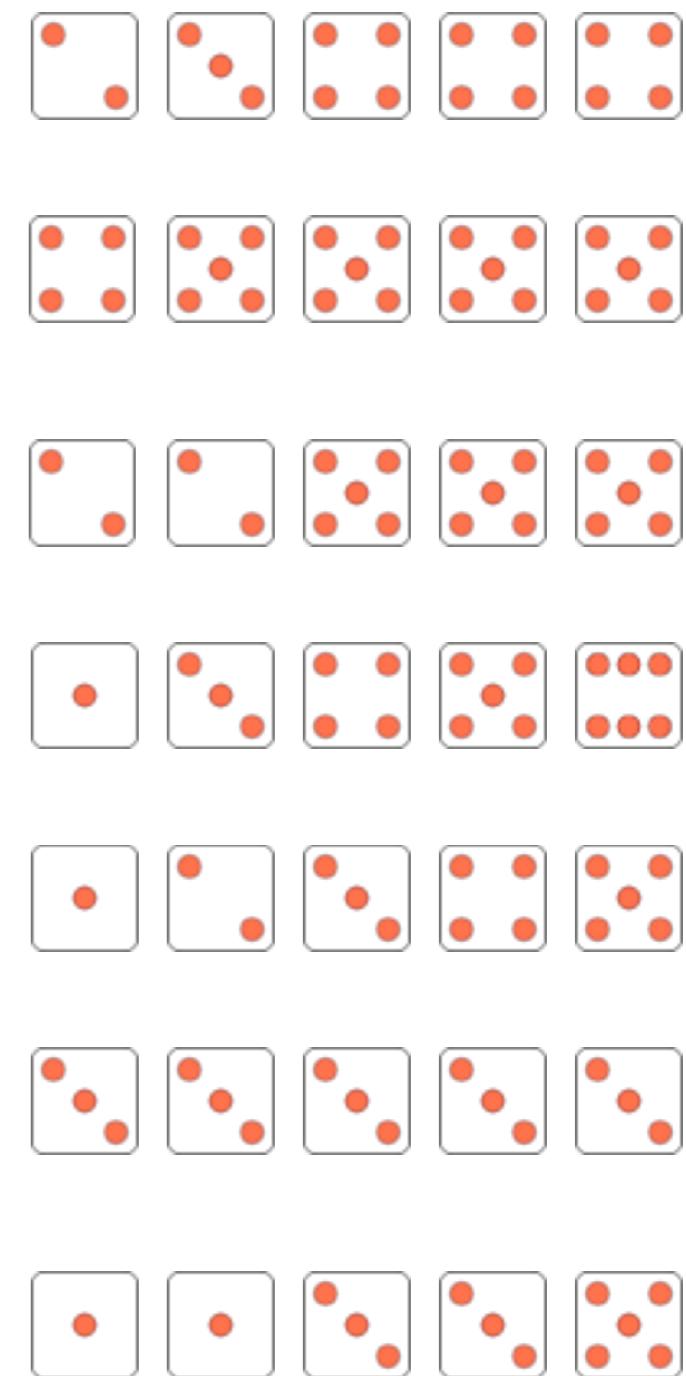
yahtzee\_test.c

```
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

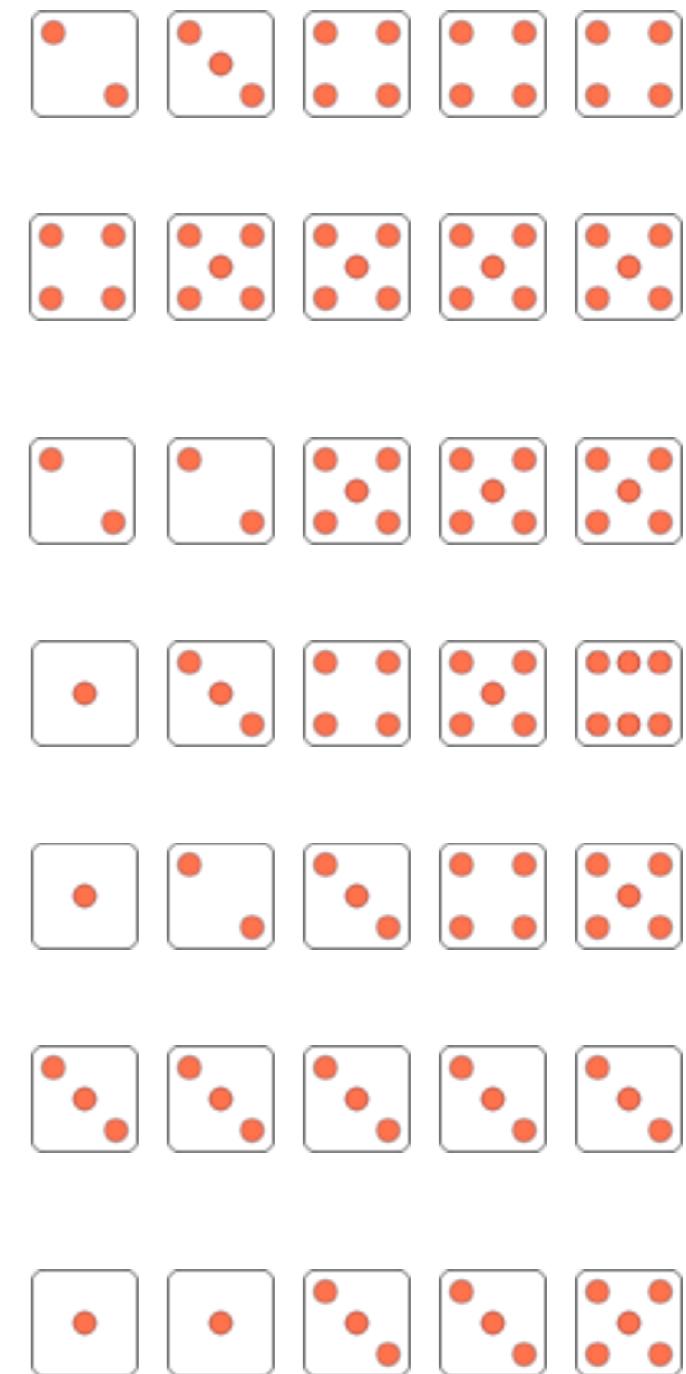
## LOWER SECTION

|              |                  |                              |
|--------------|------------------|------------------------------|
| 3 of a kind  |                  | Add Total<br>Of All Dice     |
| 4 of a kind  |                  | Add Total<br>Of All Dice     |
| Full House   |                  | <b>SCORE 25</b>              |
| Sm. Straight | Sequence<br>of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence<br>of 5 | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of<br>a kind   | <b>SCORE 50</b>              |
| Chance       |                  | Score Total<br>Of All 5 Dice |



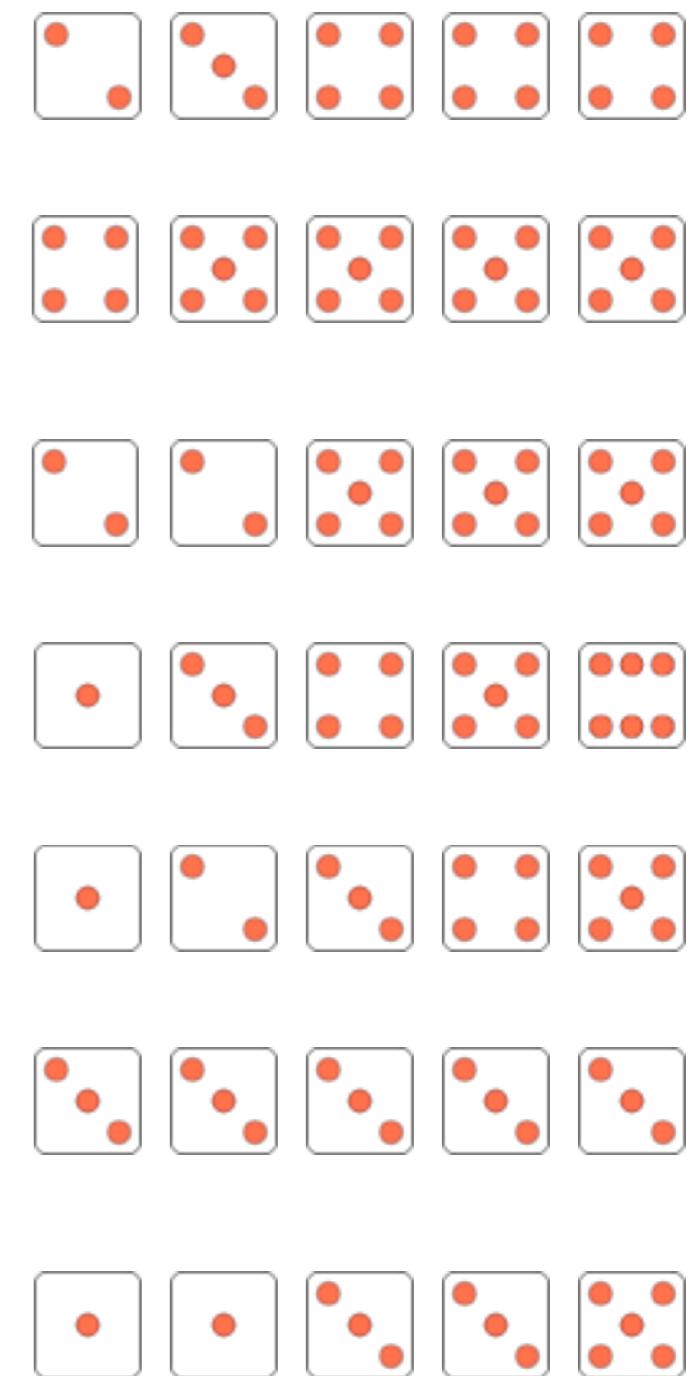
## LOWER SECTION

|              |                  |                              |
|--------------|------------------|------------------------------|
| 3 of a kind  |                  | Add Total<br>Of All Dice     |
| 4 of a kind  |                  | Add Total<br>Of All Dice     |
| Full House   |                  | <b>SCORE 25</b>              |
| Sm. Straight | Sequence<br>of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence<br>of 5 | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of<br>a kind   | <b>SCORE 50</b>              |
| 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   |                  | <b>SCORE 25</b>              |
| Sm. Straight | Sequence<br>of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence<br>of 5 | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of<br>a kind   | <b>SCORE 50</b>              |
| Chance       |                  | Score Total<br>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;
}
```

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);
```

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

Fail - Fix - **Pass**

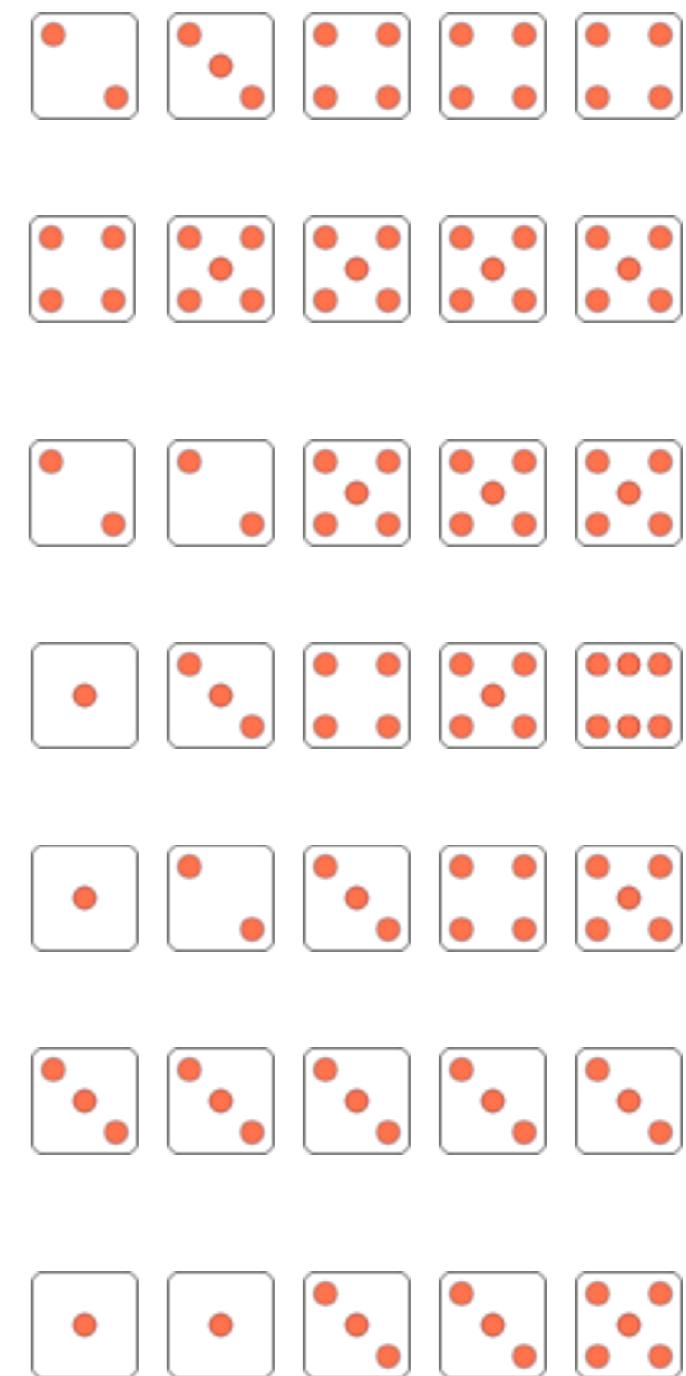
## yahtzee\_test.c

```
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

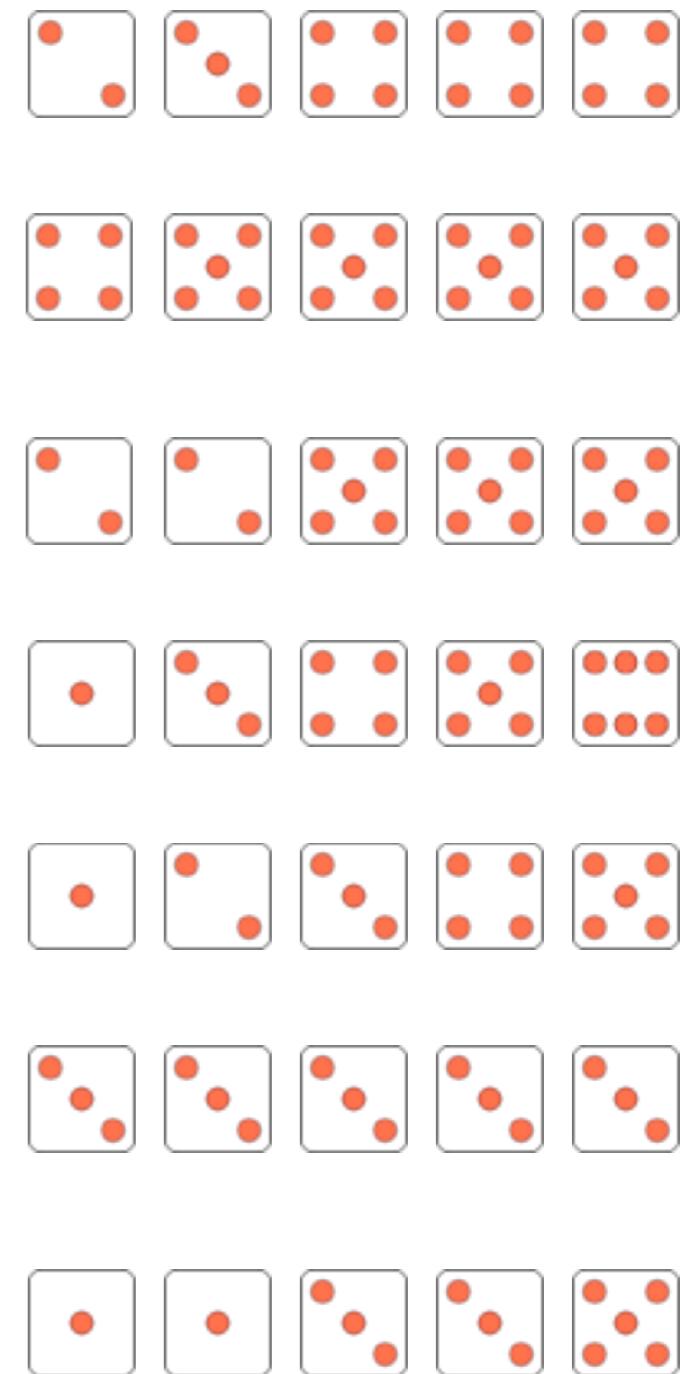
## LOWER SECTION

|              |                  |                              |
|--------------|------------------|------------------------------|
| 3 of a kind  |                  | Add Total<br>Of All Dice     |
| 4 of a kind  |                  | Add Total<br>Of All Dice     |
| Full House   |                  | <b>SCORE 25</b>              |
| Sm. Straight | Sequence<br>of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence<br>of 5 | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of<br>a kind   | <b>SCORE 50</b>              |
| Chance       |                  | Score Total<br>Of All 5 Dice |



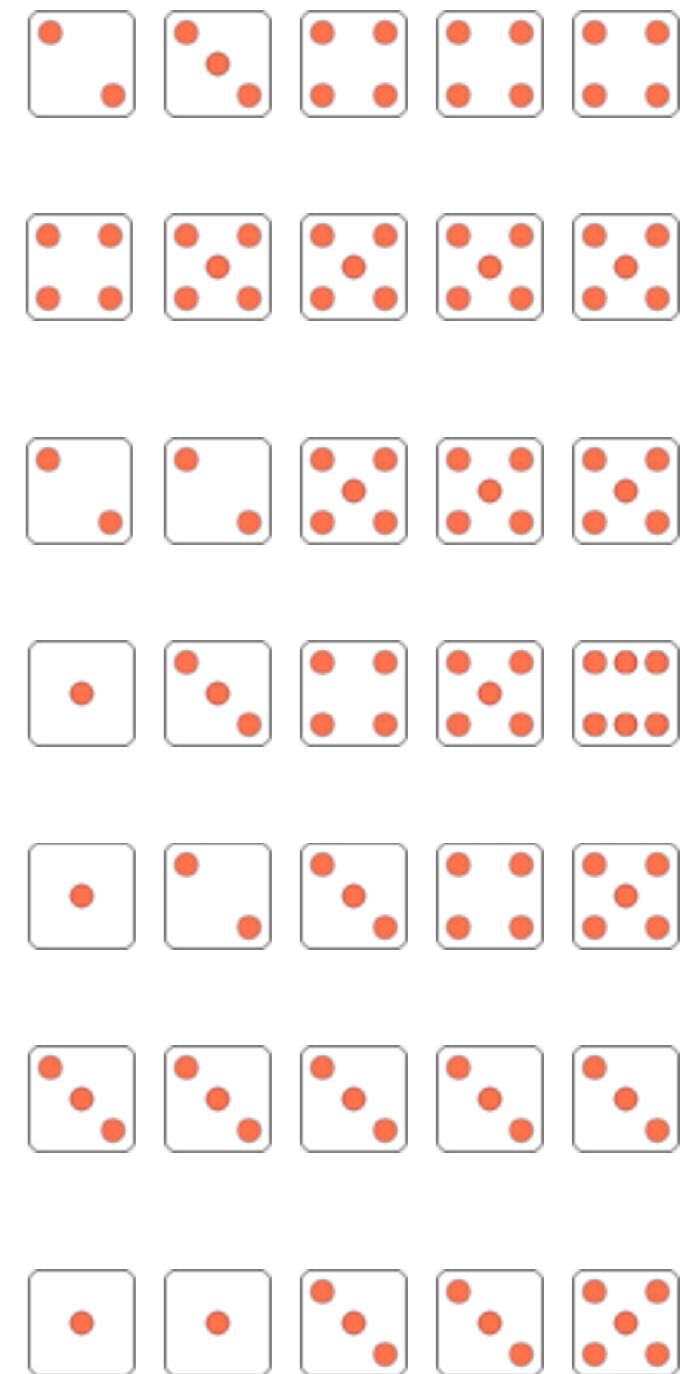
## LOWER SECTION

|              |               |                              |
|--------------|---------------|------------------------------|
| 3 of a kind  |               | Add Total<br>Of All Dice     |
| 4 of a kind  |               | Add Total<br>Of All Dice     |
| Full House   |               | <b>SCORE 25</b>              |
| Sm. Straight | Sequence of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence of 5 | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of a kind   | <b>SCORE 50</b>              |
| 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   |               | <b>SCORE 25</b>              |
| Sm. Straight | Sequence of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence of 5 | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of a kind   | <b>SCORE 50</b>              |
| 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]);
```

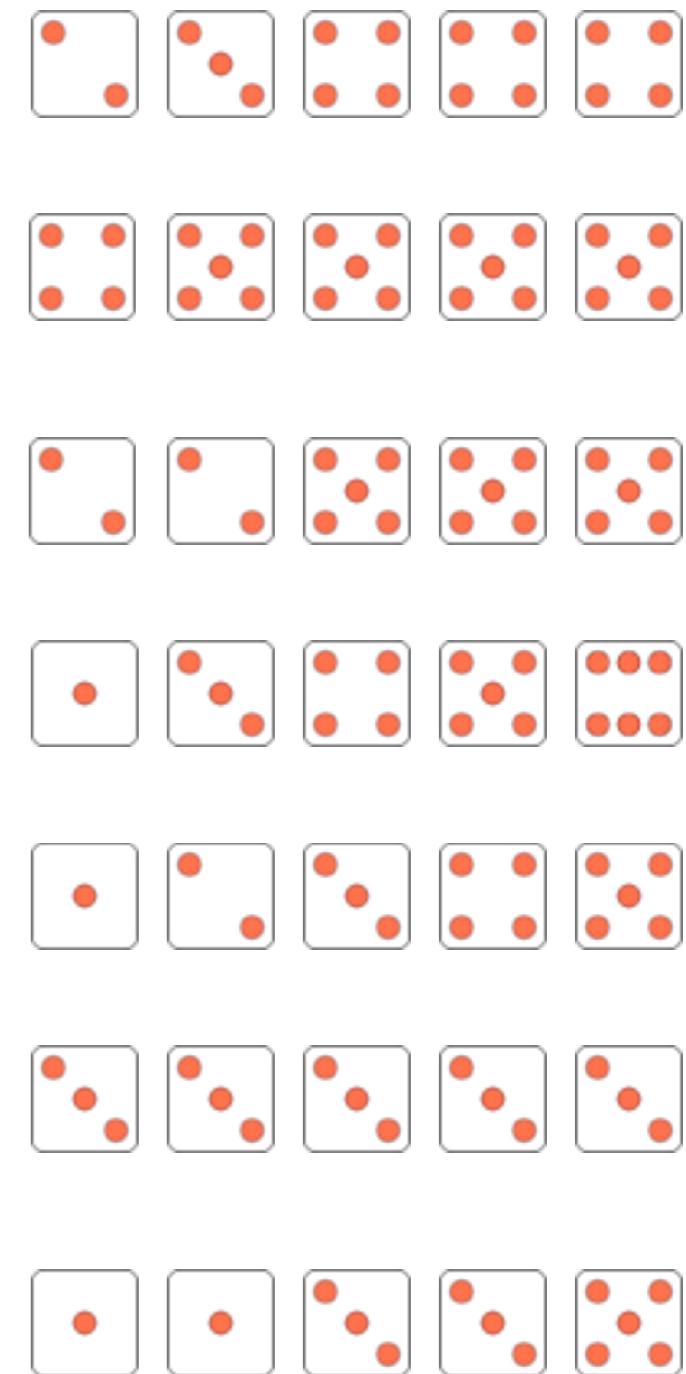
yahtzee\_test.c

```
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

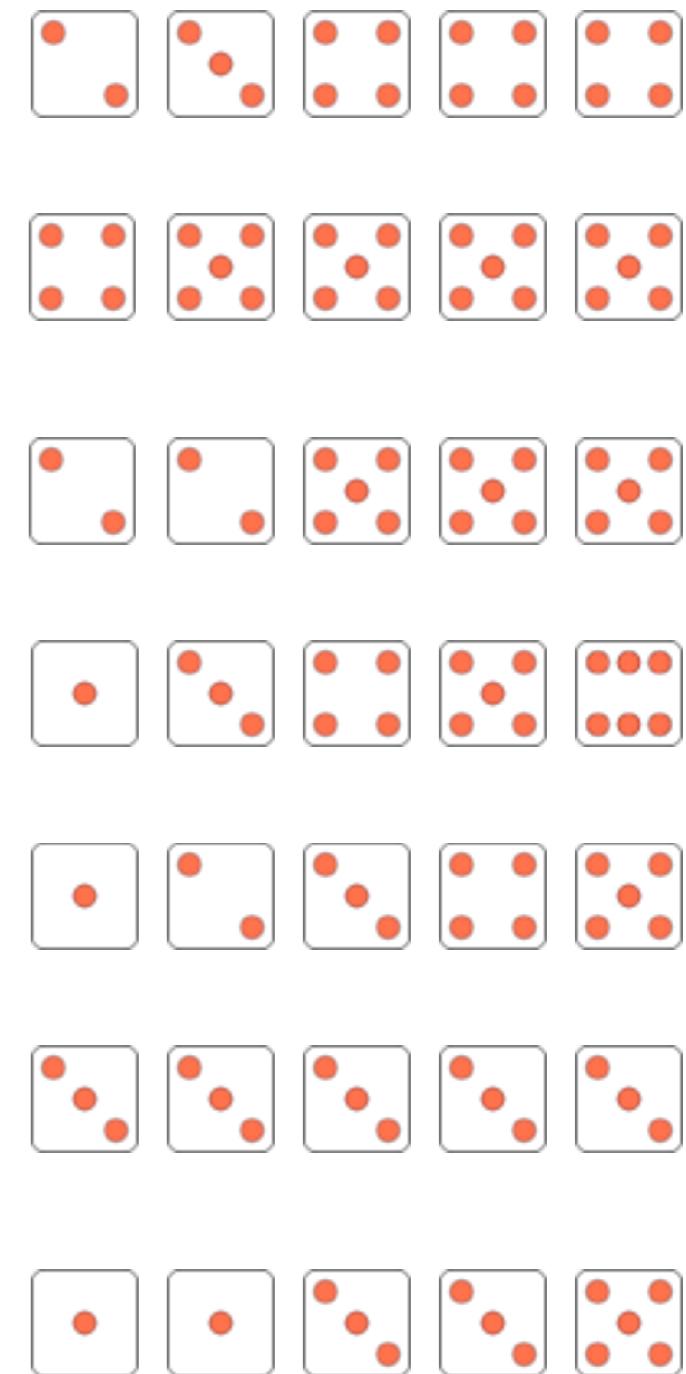
## LOWER SECTION

|              |               |                              |
|--------------|---------------|------------------------------|
| 3 of a kind  |               | Add Total<br>Of All Dice     |
| 4 of a kind  |               | Add Total<br>Of All Dice     |
| Full House   |               | <b>SCORE 25</b>              |
| Sm. Straight | Sequence of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence of 5 | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of a kind   | <b>SCORE 50</b>              |
| 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   |               | <b>SCORE 25</b>              |
| Sm. Straight | Sequence of 4 | <b>SCORE 30</b>              |
| Lg. Straight | Sequence of 5 | <b>SCORE 40</b>              |
| YAHTZEE      | 5 of a kind   | <b>SCORE 50</b>              |
| 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 -O0 -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);
}
```

# Types, Operators and Expressions

- ***implementation-defined***
  - the construct is not incorrect; the code must compile; the compiler must document the behaviour
- ***unspecified***
  - the same as implementation-defined except the behaviour need not be documented
- ***undefined***
  - the standard imposes no requirements ; anything at all can happen ; all bets are off!



examples:

signed integer right shift → implementation-defined  
function argument evaluation order → unspecified  
signed integer overflow → undefined

# auto storage

- an identifier declared inside a function<sup>t</sup>
  - has automatic storage class - it's storage is reserved each time the function is called
  - has local scope
  - has an indeterminate initial value

reading an indeterminate value causes undefined behaviour

```
int outside;  
  
int function(int value)  
{  
    int inside;←  
    ...  
    static int different;  
}
```

automatic storage class  
local scope  
no default initial value

<sup>t</sup> unless declared with the static keyword

# static storage

- an identifier declared outside a function
  - has static storage class - its storage is reserved before main starts
  - has file scope
  - has a default initial value

```
int outside;  
  
int function(int value)  
{  
    int inside;  
    ...  
    static int different;  
}
```

static storage class  
default initial value is zero

† or declared inside the  
function with the static keyword

- come in various flavours
  - also as signed or unsigned
- min-max values are not precisely defined
  - int typically corresponds to the natural word size of the host machine; the fastest integer type
  - for exact size on your computer use <limits.h>

int  
eg  
er  
s

| <i>type</i> | <i>min bits</i> | <i>min limit</i>     |
|-------------|-----------------|----------------------|
| char        | 8               | $2^7 - 1$ (127)      |
| short       | 16              | $2^{15} - 1$ (32767) |
| int         | 16              | $2^{15} - 1$ (32767) |
| long        | 32              | $2^{31} - 1$         |
| long long   | 64              | $2^{63} - 1$         |

- <stdint.h> and <inttypes.h>
  - provide specific kinds of integers

some examples

c99

int  
eg  
er  
s

| <i>type</i>           | <i>meaning</i>                         |
|-----------------------|--|
| <b>int16_t</b>        | signed int, exactly 16 bits            |
| <b>uint16_t</b>       | unsigned int, exactly 16 bits          |
| <b>int_least32_t</b>  | signed int, at least 32 bits           |
| <b>uint_least32_t</b> | unsigned int, at least 32 bits         |
| <b>int_fast64_t</b>   | signed int, fastest at least 64 bits   |
| <b>uint_fast64_t</b>  | unsigned int, fastest at least 64 bits |

- come in three flavours
  - float, double, long double
- again their limits are not precisely defined
  - double corresponds to the natural size of the host machine ; the fastest floating point type (but much much slower than integers)
- can be determined in code via <float.h>
  - e.g. DBL\_EPSILON (max  $10^{-9}$ )
  - e.g. DBL\_DIG (min 10)
  - e.g. DBL\_MIN (min  $10^{-37}$ )
  - e.g. DBL\_MAX (min  $10^{+37}$ )
- not represented with absolute precision

- there are three complex types
  - float complex
  - double complex
  - long double complex
- <complex.h> provides
  - the macro complex for \_Complex
  - lots of function declarations

```
#include <complex.h>

void eg(double complex z)
{
    double real = creal(z);
    double imag = cimag(z);
    ...
}
```

c99

- <stdbool.h> provides three macros
  - **bool** for \_Bool
  - **false** for 0
  - **true** for 1
- the size of the **bool** type is not defined
- any integer value can be converted to a **bool**
  - zero is interpreted as **false**
  - any non-zero is interpreted as **true**

c99

```
#include <stdbool.h>

bool love = true;
bool teeth = false;
```



## the char type represents a single byte

- the smallest addressable unit of memory
- usable as a single character or a very small int

| <i>escaped<br/>chars</i> | <i>meaning</i>  |
|--------------------------|-----------------|
| '\n'                     | newline         |
| '\t'                     | tab             |
| '\b'                     | backspace       |
| '\r'                     | carriage return |
| '\f'                     | form feed       |
| '\\'                     | backslash       |
| '\''                     | single quote    |

# sizeof

- **sizeof is a unary operator**
  - use is **sizeof(type)** or **sizeof expression**
  - **common for dynamic memory allocation**
- **result is number of bytes as a size\_t**
  - **size\_t is a typedef for an unsigned integer**
  - **capable of holding the size of any variable**

```
type * var = malloc(sizeof(type));
```

```
type * var = malloc(sizeof *var);
```

→ this version is slightly better. why?

# lit er als

- literals for simple types are const!
  - their types can be specified

| <i>type</i> | <i>suffix</i> | <i>example</i> |
|-------------|---------------|----------------|
| long int    | L or l        | 42L            |
| unsigned    | U or u        | 42U            |
| float       | F or f        | 42F            |
| long double | L or l        | 42.0L          |

- variables can be const!
  - useful for naming magic numbers

```
const double pi = 3.141592;
```

```
pi += 4.22;
```

compile time error

## C is very liberal in its conversions

- a **widening conversion never loses information**
- a **narrowing conversion may lose information**

```
double mass = 0;
```

int → double

```
int bad_pi = 3.141592;
```

double → int

- an **explicit conversion is called a cast**
  - **syntax is (type)expression**
  - **(void) is sometimes used to make discard explicit**

```
int cast = (int)mass;
```

```
(void)printf("%i", cast);
```

- the usual arithmetic operators
  - + - \* /
  - % is the remainder operator
  - note that integer / integer == integer

```
bool is_even(int value)
{
    return value % 2 == 0;
}
```

- overflow
  - undefined for signed integers
  - well defined for unsigned integers
  - infinities, NaN's, <fenv.h> for floating point
- divide by zero
  - undefined

## initialization != assignment

- initialization occurs at declaration
- assignment occurs after declaration

```
int count;  
count = 0;
```

assignment

?

```
int count = 0;
```

initialization - better

✓

```
const int answer = 42;
```

initialization

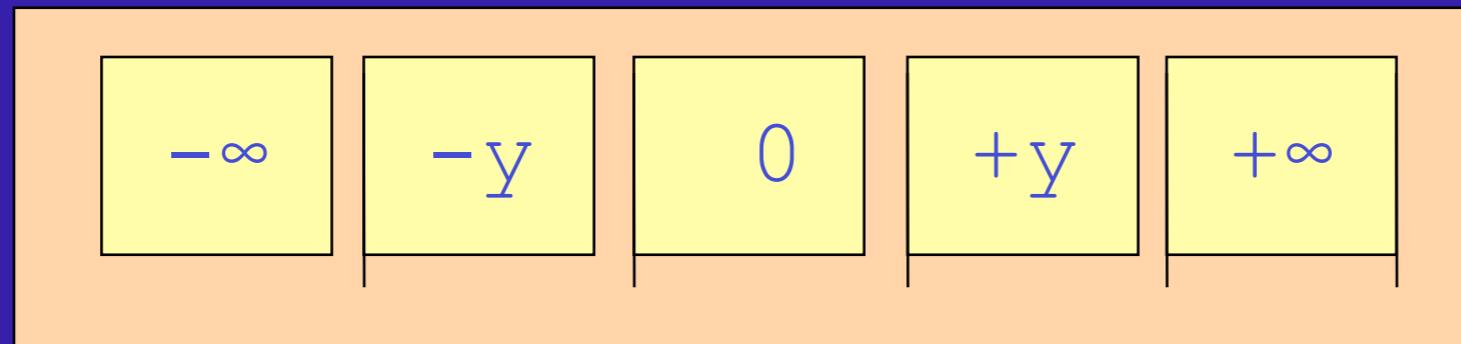
✓

```
const int answer;  
answer = 42;
```

compile-time  
error

✗

- **$== !=$  operators test for equality or identity**
  - don't use  $== !=$  on floating point operands
  - don't use  $== !=$  on boolean literals
- **$< <= > >=$  operators test relational ordering**
  - works all numeric types (but NaNs are unordered)
  - rarely useful on *char*
  - but '0' .. '9' are sequential



floating point  
ordering

# simple assignment

## assignment is an expression

- so it has an outcome – the value of the rhs
- assignment also has a significant side effect!

```
int lower;
int upper;

lower = 0;
printf("%d", lower = 0);

lower = upper = 0;
printf("%d", lower = upper = 0);
```

same as  
upper = 0;  
lower = upper;

# assignment

- common assignment patterns are supported natively with compound assignment operators

non-idiomatic

```
lhs = lhs * rhs;
```

```
lhs = lhs / rhs;
```

```
lhs = lhs % rhs;
```

```
lhs = lhs + rhs;
```

```
lhs = lhs - rhs;
```

?

idiomatic

```
lhs *= rhs;
```

```
lhs /= rhs;
```

```
lhs %= rhs;
```

```
lhs += rhs;
```

```
lhs -= rhs;
```



✓

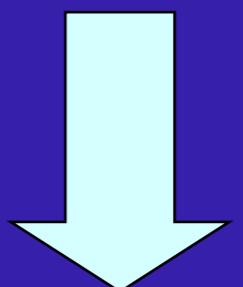
- adding/subtracting one is supported directly
  - **++** is the increment operator
  - **--** is the decrement operator

non-idiomatic

```
lhs = lhs + 1;
```

?

```
lhs = lhs - 1;
```

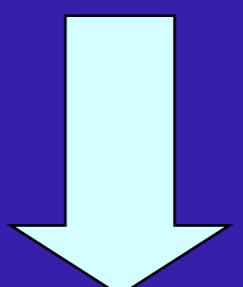


non-idiomatic

```
lhs += 1;
```

?

```
lhs -= 1;
```



idiomatic

```
lhs++;
```



```
lhs--;
```

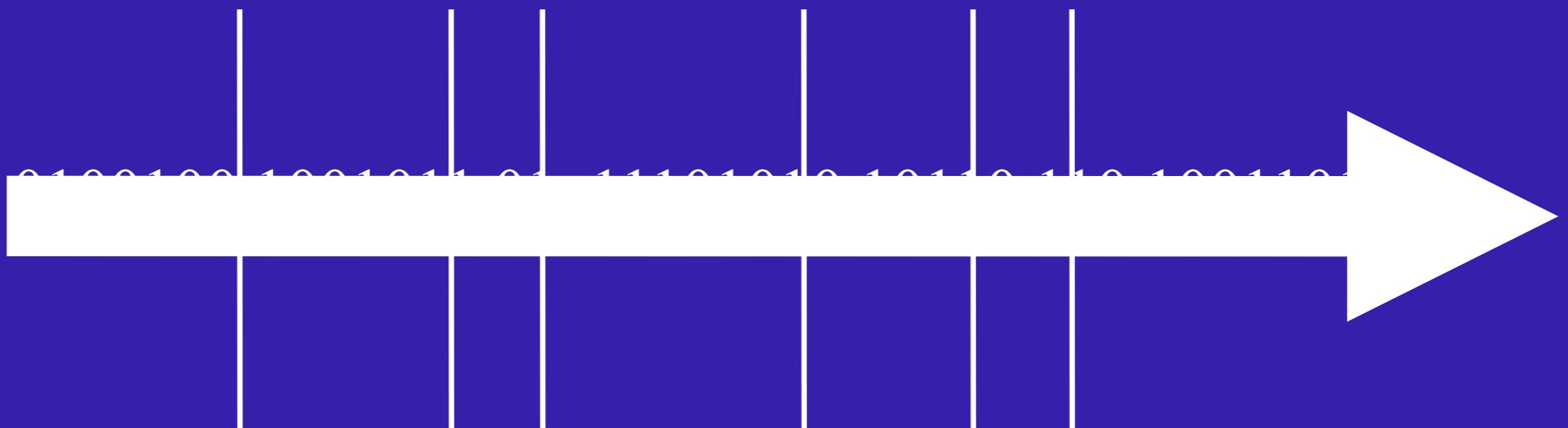
**++ and -- come in two forms**

- result of **++var** is var *after* the increment
- result of **var++** is var *before* the increment
- no other operators behave like this :-)

**prefix = ++m;****m = m + 1;**  
**prefix = m;**equivalent<sup>†</sup>**postfix = m+**  
**+;****postfix =**  
**m;**  
**m = m + 1;**

<sup>†</sup>except that m is evaluated only once

- a sequence point is...
  - a point in the program's execution sequence where all previous side-effects shall have taken place and where all subsequent side-effects shall not have taken place



- sequence points occur...
  - at the end of a full expression
    - a full expression is an expression that is not a sub-expression of another expression or declarator (6.8p2)
  - after the first operand of these operators
    - && logical and
    - || logical or
    - ?: ternary
    - , comma
  - after evaluation of all arguments and function expression in a function call
    - note that the comma used for separating function arguments is not a sequence point
  - at the end of a full declarator

C Standard: 6.5 Expressions  
Between the previous and next sequence point an object shall have its stored value modified at most once by the evaluation of an expression.

...  
in other words, if an object is modified more than once between sequence points the result is undefined

## C Standard: 6.5 Expressions

...

Between the previous and next sequence point...the prior value shall be read only to determine the value to be stored.

in other words, if an expression reads the value of a modified object more than once between sequence points the result is undefined

$n = n++$  $n + n++$ 

Are these expressions  
undefined?

## Ihs && rhs

- if lhs is false the rhs is **not evaluated**
- if lhs is true, sequence point, rhs is evaluated

## Ihs || rhs

- if lhs is true the rhs is **not evaluated**
- if lhs is false, sequence point, rhs is evaluated

&amp;&amp;

false

true

false

false

false

true

false

true

||

false

true

false

false

true

true

true

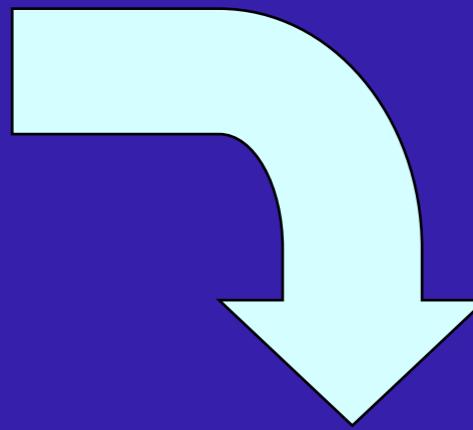
true

# ternary operator

252

- the only operator with three arguments
  - $a ? b : c \rightarrow \text{if } (a) b; \text{else } c;$
  - sequence point at the  $?$
  - an expression rather than a statement
  - useful in macros and to avoid needless repetition

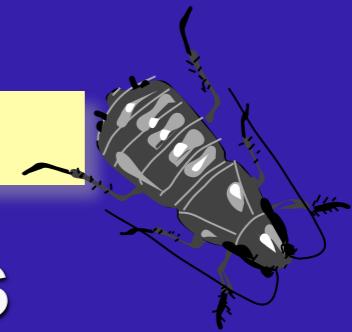
```
void some_func(void)
{
    bool found = search(...);
    if (found)
        printf("found");
    else
        printf("not found");
}
```



```
void some_func(void)
{
    bool found = search(...);
    printf("%sfound", found ? "" : "not
");
}
```

- **Ihs , rhs**
  - Ihs is evaluated and the result is discarded
  - sequence point at the comma
  - rhs is evaluated and is the result

```
int last = (2, 3, 4, 5, 6);
```

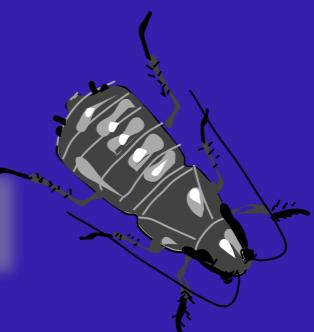


- sometimes seen in for statements

```
for (octave = 0, freq = 440;  
     is_audible(freq);  
     ++octave, freq *= 2) ...
```

does this access an element of a 2-d array?

```
int element = matrix[row,col];
```



- in these statements...
  - where are the sequence points?
  - how many times is m modified?
  - which ones are undefined?

1

`f (++m * m++);`

2

`m = m++;`

3

`m = m = 0;`

pr  
ec  
ed  
en  
ce

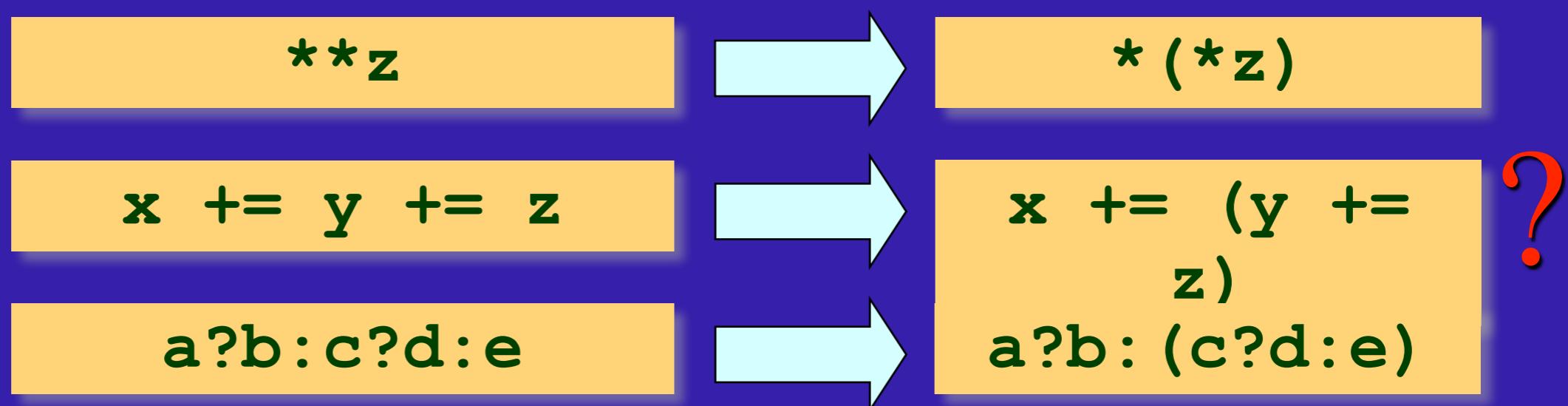
255

|                 |                                  |
|-----------------|----------------------------------|
| primary         | ( ) [ ] -> .                     |
| unary           | ! ~ + - ++ -- (T) sizeof + - * & |
| multiplicative  | * / %                            |
| additive        | + -                              |
| shift           | << >>                            |
| relational      | < > <= >=                        |
| equality        | == !=                            |
| bitwise/boolean | & then ^ then                    |
| boolean         | && then    then ?:               |
| assignment      | = *= /= %= += -= ...             |

- rule 1
  - except for assignment all binary operators are left-associative



- rule 2
  - unary operators, assignment and ?: are right-associative



- very important
  - precedence controls operators not operands
  - order of evaluation of operands is unspecified
  - only sequence points guarantee evaluation order

```
int x = f() + g() * h();
```

In this example the three functions f( ) and g( ) and h( ) can be called in any order

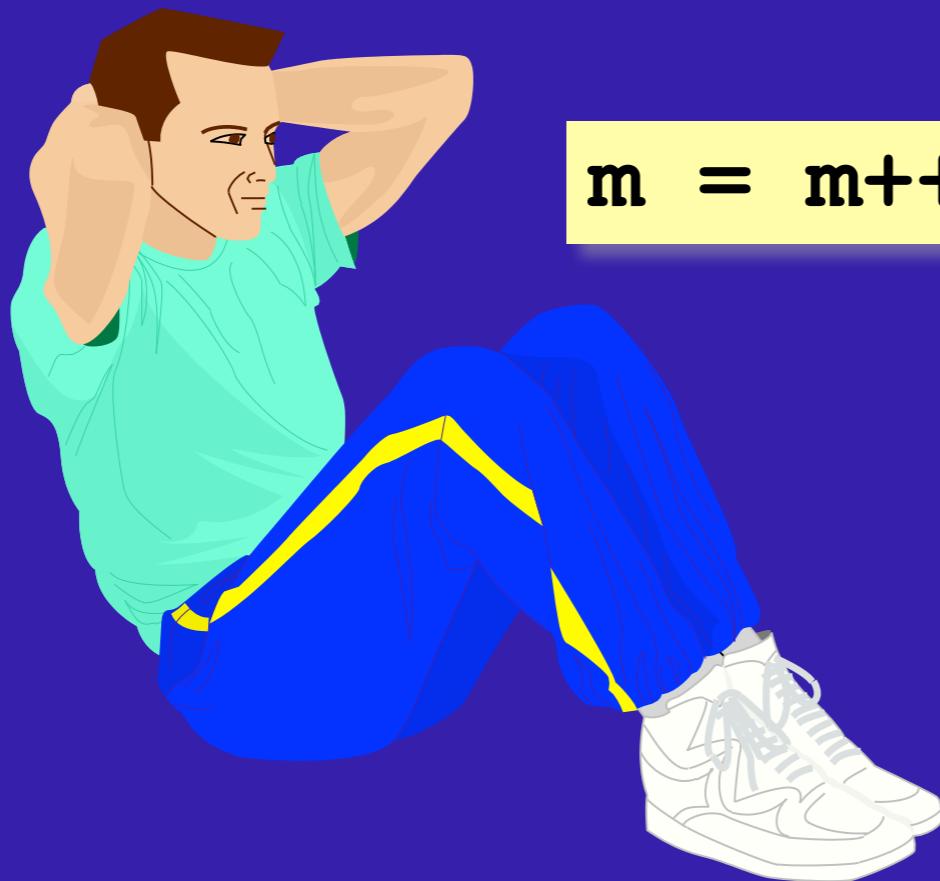
```
int v1 = f();
int v2 = g();
int v3 = h();
int x = v1 + v2 * v3;
```

?

258

ex  
er  
cis  
e

- in the given statement...
  - where are the sequence points?
  - what are the operators?
  - what is their relative precedence?
  - how many times is m modified between sequence points?
  - is it undefined?



`m = m++, m;`



- **know your enemy!**
  - **undefined vs unspecified vs imp-defined**
  - **local variables do not have a default value**
  - **a char is the smallest addressable unit of memory**
  - **many integer types have minimum sizes**
  - **integers can be interpreted as true/false**
  - **integer arithmetic overflow can be undefined**
  - **type conversions are implicit and liberal!**
  - **initialisation != assignment**
  - **assignment is an expression**
  - **sequence points knowledge is vital**
  - **precedence controls operators not operands**
  - **order of evaluation between sequence points is unspecified**
  - **strive for simplicity**

# Control Flow



Deep C - a 3 day course  
Jon Jagger & Olve Maudal  
(March 28, 2012)

# blocks / compound statement

a compound statement (aka a block) is:

- an unnamed sequence of statements and declarations
- grouped together inside { braces }

;

the null statement

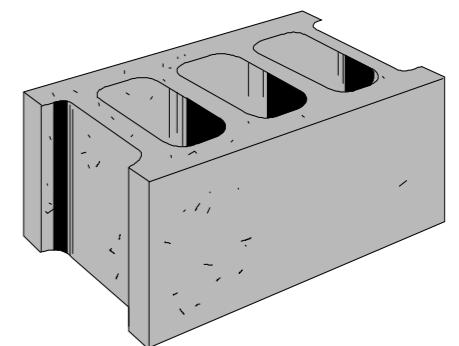
declaration;

expression;

semicolons required

```
{  
    ... ;  
    ... ;  
    ... ;  
}
```

trailing  
semicolon  
not required



**if**

the simplest selection statement

**if** ( *expression* )  
*statement*

```
bool leap_year(int year)
{
    if (year % 400 == 0)
        return true;
    if (year % 100 == 0)
        return false;
    if (year % 4 == 0)
        return true;
    return false;
}
```

# if-else

```
if ( expression )  
    statement  
else  
    statement
```

```
bool leap_year(int year)  
{  
    if (year % 400 == 0)  
        return true;  
    else if (year % 100 == 0)  
        return false;  
    else if (year % 4 == 0)  
        return true;  
    else  
        return false;  
}
```

# dangling else

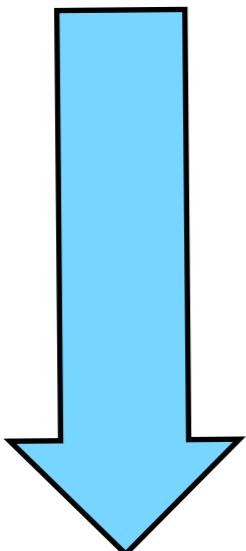
in a nested if an else associates with its nearest lexical if

physical indentation != logical indentation

```
if (value >= 0)
    if (value <= 100)
        return true;
else
    return false;
```



??



physical indentation == logical indentation

```
if (value >= 0)
    if (value <= 100)
        return true;
else
    return false;
```

?



in this code fragment x it is "intuitively obvious"  
that x incremented before being used  
as an argument to func()

```
if (x++ < 10) {  
    func(x);  
}
```

How do you **know** this?

For sure?



because  $x++ < 10$  is the *full-expression* controlling the if statement

and there is a sequence-point at the end of each *full-expression*

**if** ( *expression* )  
*statement*

**if** ( *expression* )  
sequence-point  
*statement*

```
if (x++ < 10)
{
    func(x);
}
```

```
if ( x++ < 10 )
sequence-point
{
    func(x);
}
```

# discussion

using `=` instead of `==` is a common bug

- compiles because assignment is an expression



```
if (x = 42)
```

...



```
if (x == 42)
```

...



oops: but not a  
compile time error

how about reversing the operands?

- a common guideline, but what do you think?

```
if (42 = x)
```

...



compile time error

```
if (42 == x)
```

...





```
if (42 == x)
```

...



My advice is that writing `if (42 == x)` is not a good idea. It makes the code harder to read. And if you are aware enough of the problem to write the variable on the right hand side surely you are aware enough of the problem to write `==` instead of `=?` And it doesn't apply all the time. What about if both arguments are variables? Most crucially of all, writing `if (x = 42)` should be found by tests. Time and time again studies have shown the two major factors in building software are (1) how interdependent the various parts of your software are – so that when you change one part only some of the rest is affected, and (2) how easy the code is to comprehend.

When programming, **readability** should have very high priority!

# switch

stylised if (value == constant)-else chain

- switch on integral types only
- case labels must be compile time constants
- no duplicate case labels, no shortcut for ranges

```
const char * day_suffix(int days)
{
    const char * result = "th";
    if (days / 10 != 1)
        switch (days % 10 == 1)
    {
        case 1 :
            result = "st"; break;
        case 2 :
            result = "nd"; break;
        case 3 :
            result = "rd"; break;
        default:
            result = "th"; break;
    }
    return result;
}
```

# fall through

case labels provide "goto" style jump points

- when inside the switch they play no part
- there is no implicit break
- this is known as fall-through

```
void send(short * to, short * from, int count)
{
    int n = (count + 7) / 8;
    switch (count % 8)
    {
        case 0 : do { *to++ = *from++;
        case 7 :           *to++ = *from++;
        case 6 :           *to++ = *from++;
        case 5 :           *to++ = *from++;
        case 4 :           *to++ = *from++;
        case 3 :           *to++ = *from++;
        case 2 :           *to++ = *from++;
        case 1 :           *to++ = *from++;
                           } while (--n > 0);
    }
}
```



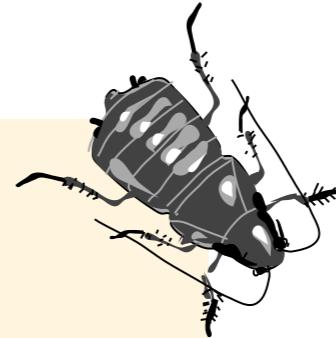
Duff's Device

# switch gotchas

how do you  
spell default?

```
int at;  
...  
switch (days % 10)  
{  
    at = 0; ←  
  
    case 1 :  
        while (at != 0) ... break;  
    case 2 :  
        while (at != 0) ... break;  
    case 3 :  
        while (at != 0) ... ←  
    defualt:  
        error(); break;  
}
```

will this assignment  
happen?



fall-through is  
usually an error

why write this break?

# while

the simplest iteration statement

*initialization*  
**while** ( *continuation-condition* )  
{  
    *loop-task*  
    *update-loop*  
}

```
int digit = 0;
while (digit != 10) {
    printf("%d ", digit);
    digit++;
}
```

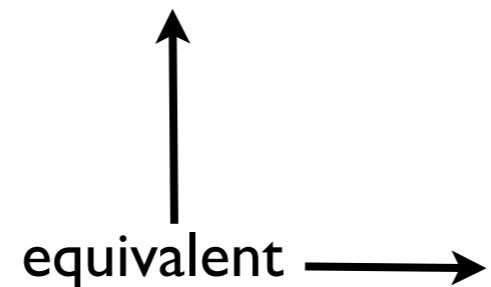
0 1 2 3 4 5 6 7 8 9

# for

stylized iteration - "scaffolding" all together

- you can omit any part of the `for` statement
- declared variables are scoped for `for` statement

```
for (initialization; continuation-condition; update-loop)
{
    loop-task
}
```



```
{
    initialization
while ( continuation-condition )
{
    loop-task
    update-loop
}
}
```

```
for (int digit = 0; digit != 10; digit++) {
    printf("%d ", digit);
}
```

0 1 2 3 4 5 6 7 8 9

# do

continuation condition is tested at end

- means loop executed at least once
- much rarer than for or while statement

*initialization*

```
do
{
    loop-task
    update-loop
}
while (continuation-condition) ;
```

; needed here

```
int digit = 0;
do {
    printf("%d ", digit);
    digit++;
}
while (digit != 10);
```

0 1 2 3 4 5 6 7 8 9

# return

the return statement returns a value!

- the expression must match the return type
- either exactly or via implicit conversion
- to end a void function early use return;

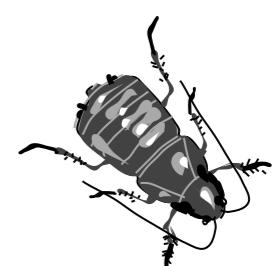
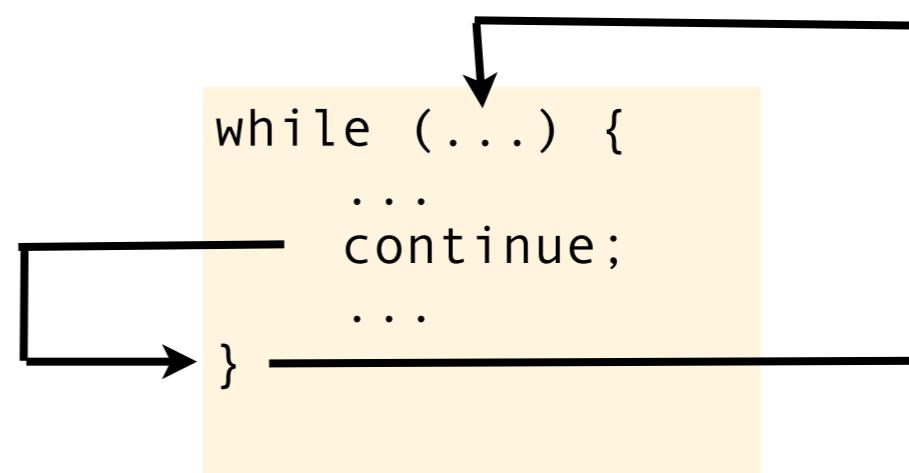
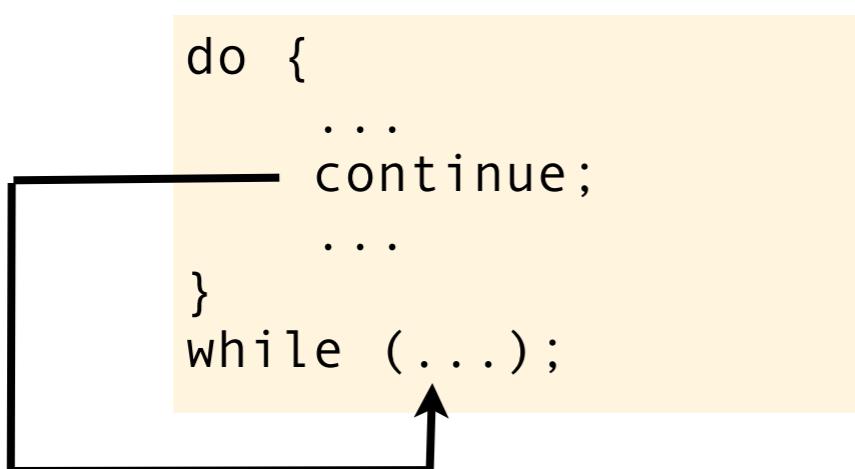
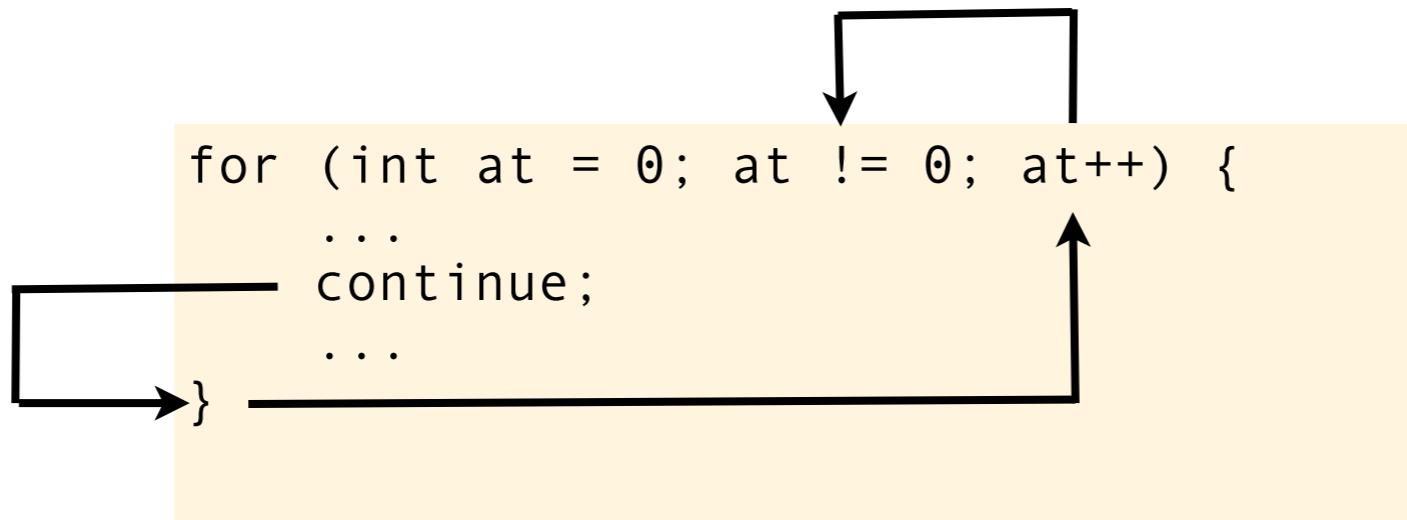
```
return expressionopt;
```

```
const char * day_suffix(int days)
{
    const char * result = "th";
    if (days / 10 != 1)
        switch (days % 10 == 1)
    {
        ...
    }
    return result;
}
```

# continue

starts a new iteration of nearest enclosing while/do/for

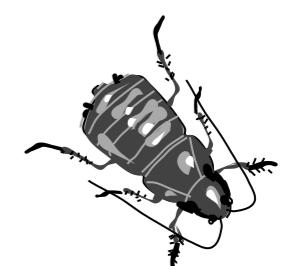
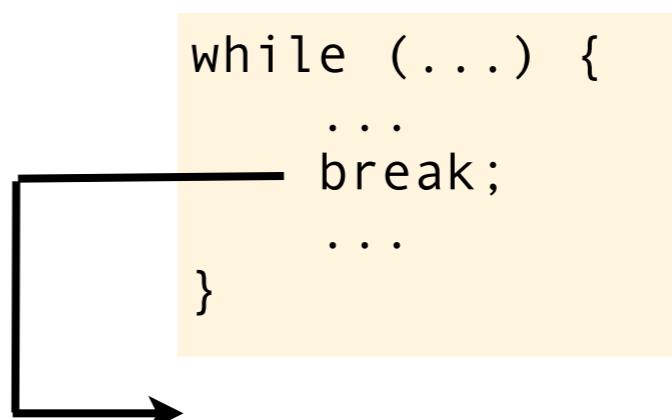
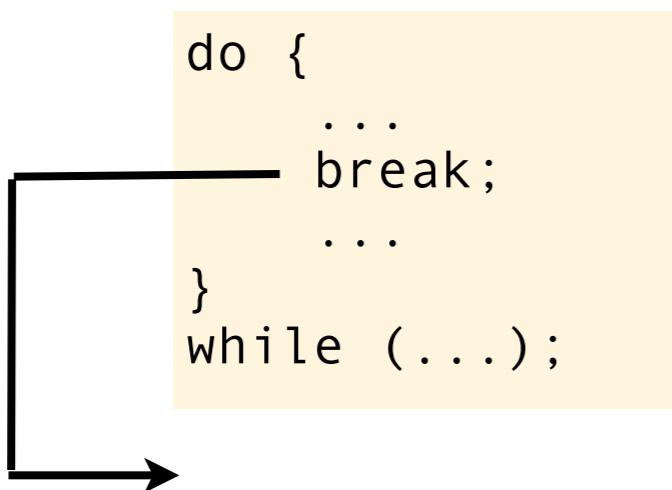
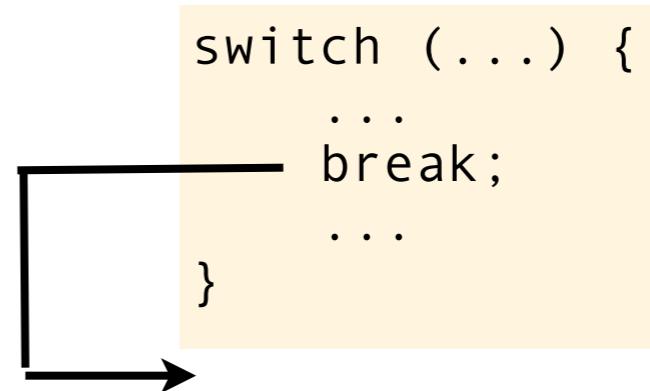
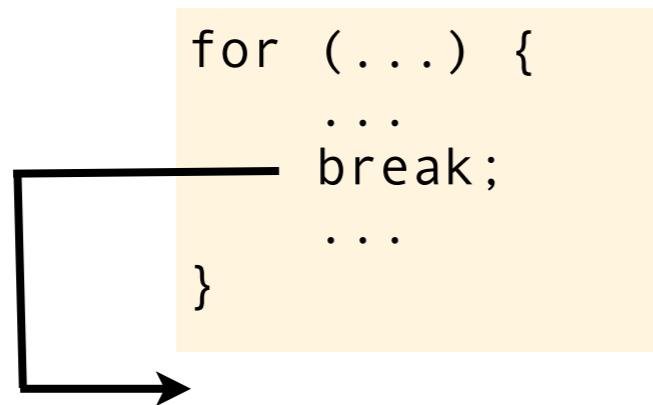
- highly correlated with bugs



# break

exits nearest enclosing while/do/for/switch

- less highly correlated with bugs in loops
- normal in a switch statement



# goto

jumps to a named label

- can jump forward or backward
- can bypass variable initialization!

*identifier : statement*

```
void some_function(void)
{
    FILE * f1 = fopen("data1.txt", "r");
    FILE * f2 = fopen("data2.txt", "r");
    ...
    if (error)
        goto cleanup;
    ...

cleanup:
    if (f1) fclose(f1);
    if (f2) fclose(f2);
}
```



*The goto statement goes to a labelled statement not to a label.*

# identifiers

larger scope  $\leftrightarrow$  longer identifier

smaller scope  $\leftrightarrow$  shorter identifier

```
for (int loop_index = 0; loop_index != 42; loop_index++)
{
    widgetize(wibbles[loop_index]);
}
```

```
for (int index = 0; index != 42; index++)
{
    widgetize(wibbles[index]);
}
```

```
for (int at = 0; at != 42; at++)
{
    widgetize(wibbles[at]);
}
```

better

better



Ehm... I just have to say something about the previous  
for-loop.

```
for (int at = 0; at != 42; at++)  
{  
    widgetize(wibbles[at]);  
}
```

Using != like this is a C++ idiom, made popular by iterators,  
but it is **not** a C idiom. Also it might be difficult to optimize.  
Do not be surprised if I one day refactor it into:

```
for (int at = 0; at < 42; at++)  
{  
    widgetize(wibbles[at]);  
}
```

and while I am at it:

```
for (int at = 0; at < 42; at++)  
    widgetize(wibbles[at]);
```

# assert

- assert macro can be found in <assert.h>
- takes a condition as its argument
- aborts program if condition is false
- compiled out if NDEBUG is defined
- useful for expressing function-call contracts or for data-structure invariants

```
#include <assert.h>

void example(int year, int month, int day)
{
    → assert(month > 0);
    → assert(month <= 12);
    → assert(is_valid_day(year, month, day));
    ...
}
```

aborts with a diagnostic stating the condition, the file, the line number and (optionally) the function that failed

# testing

assert can also be used for writing simple automatic tests

- this places the assertions outside the production code rather than inside it

```
#include "rational.h"
#include <assert.h>

void identical_rationals_compare_equal(void)
{
    rational lhs = { 42, 6 }, rhs = { 42, 6 };
    assert(equal_rationals(lhs, rhs));
}

void unnormalized_integral_values_rationalize(void)
{
    rational original = { 42, 6 };
    rational normalized = normalize_rational(original);
    rational expected = { 7, 1 };
    assert(equal_rationals(normalized, expected));
}
```

# summary

## selection statements

- if : beware of dangling else

## iteration statements

- for : common, loop scaffolding all done together
- while : also common
- do : much less common

## jump statements

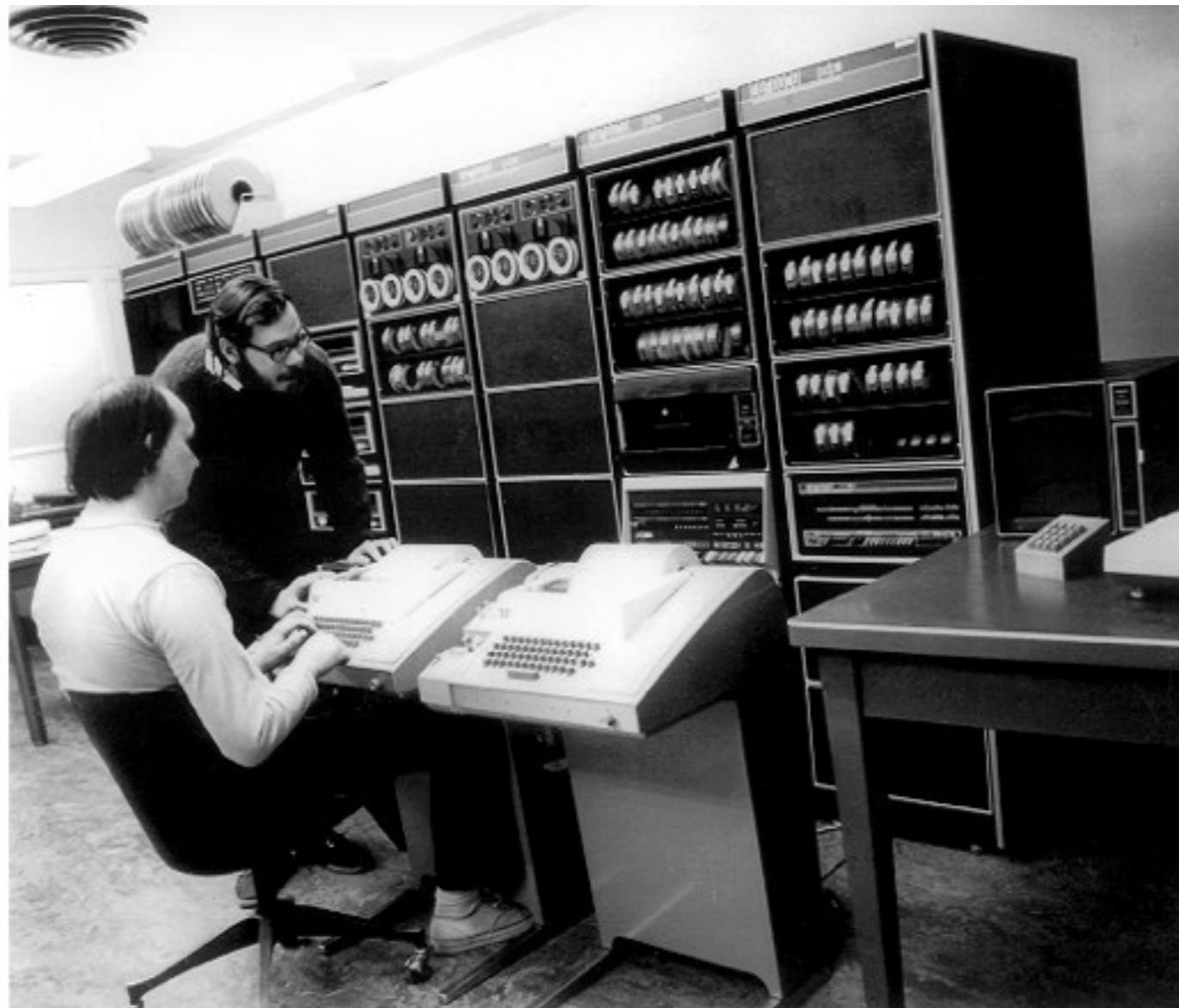
- continue, break, goto : all correlated with bugs
- best avoided
- break in a switch is ok

## assertions

- useful for external testing and asserting invariants

# Pointers and Arrays

“to get a deeper understanding of the language”



a 3 day course  
by  
Olve Maudal & Jon Jagger

# pointers

a \* in a declaration declares a pointer

- read declarations from right to left
- beware: the \* binds to the identifier and not the type

```
int * stream;
```

stream

```
int * stream;
```

is-a

pointer

```
int * stream;
```

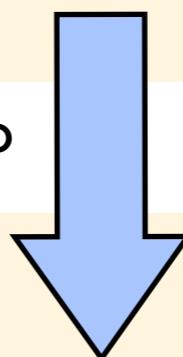
to-an

int

```
int * pointer, value;
```

equivalent to

```
int * pointer;  
int value;
```



# the null pointer (NULL or 0)

- null ever equals an objects' address
- the default for pointers with static storage class
- no default for pointers with auto storage class

```
int * pointer = NULL;
```

Equivalent

```
int * pointer = 0;
```

NULL is in <stddef.h>  
(and others)

```
int * top_level;  
  
void eg(void)  
{  
    int * local;  
    static int * one;  
    ...  
}
```

implicit static storage class,  
defaults to null

implicit auto storage class,  
no default

explicit static storage class,  
defaults to 0

# pointer true/false

a pointer expression can implicitly be interpreted as true or false

- a null pointer is considered false
- a non-null pointer is considered true

```
int * pos; ...
```

```
if (pos)
if (pos != 0)
if (pos != NULL)
```

}

equivalent

```
if (!pos)
if (pos == 0)
if (pos == NULL)
```

}

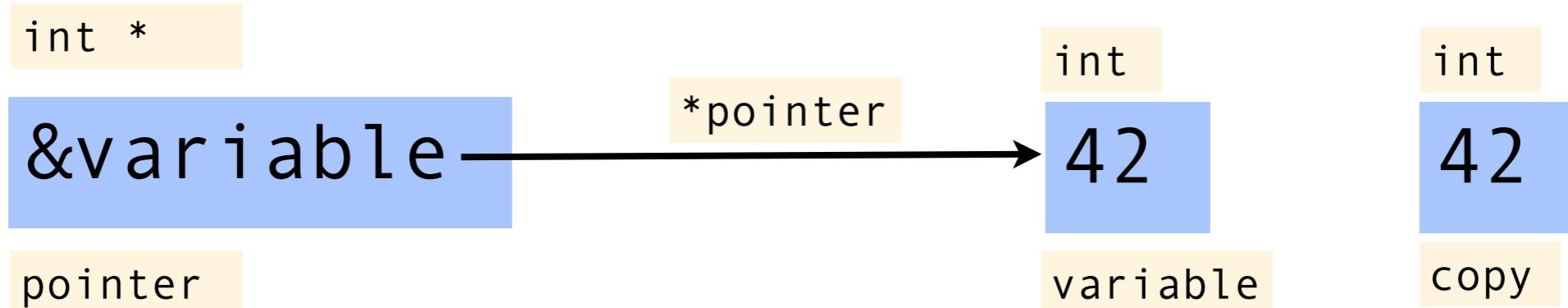
equivalent

# address-of / dereference

unary & operator returns a pointer to its operand

- unary \* operator dereferences a pointer
- & and \* are inverses of each other:  $*\&x == x$
- $*p$  is undefined if p is invalid or null

```
int variable = 42;  
...  
int * pointer = &variable; ← * used in a declarator  
...  
int copy = *pointer; ← * used in an expression
```



# pointer function arguments

```
#include <stdio.h>

void swap(int * lhs, int * rhs)
{
    int temp = *lhs;
    *lhs = *rhs;
    *rhs = temp;
}

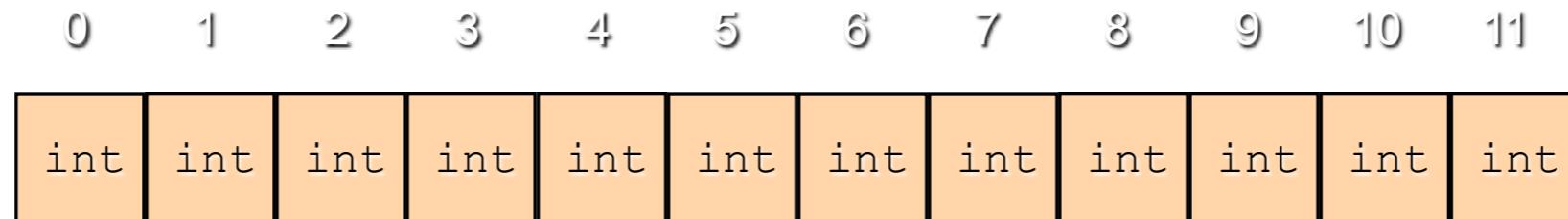
int main(void)
{
    int a = 4;
    int b = 2;
    printf("%d,%d\n", a, b);
    swap(&a, &b);
    printf("%d,%d\n", a, b);
}
```

# arrays

an array is a fixed-size contiguous sequence of elements

- all elements have the same type
- default initialization when static storage class
- no default initialization when auto storage class

```
int days_in_month[12];
```



the type of days\_in\_month is int[12]

# pointer function arguments

arrays support aggregate initialization

- syntax not permitted for assignment
- any missing elements are default initialized
- arrays cannot be initialized/assigned from another array

```
const int days_in_month[12] =  
{  
    31, // January  
    28, // February  
    31, // March  
    ...  
    31, // October  
    30, // November  
    31 // December  
};
```



size is  
optional



- a trailing comma is allowed
- an empty list is not allowed (it is in C++)

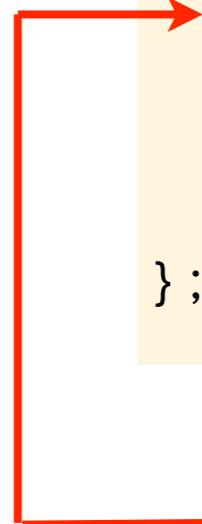
## [ designators ]

- arrays support [int] designators
- int must be a constant-expression

c99

```
enum { january, february, march, ...
       october, november, december };
```

```
const int days_in_month[] =
{
    [january] = 31,
    [february] = 28,
    [march] = 31,
    ...
    [october] = 31,
    [november] = 30,
    [december] = 31
};
```



these initializer list elements can now appear in any order

# array indexing

- starts at zero and is not bounds-checked
- out of bounds access is undefined

```
int days_in_month[12];
```

```
printf("%d", days_in_month[january]);
```



```
printf("%d", days_in_month[-1]);  
printf("%d", days_in_month[12]);
```



# pointers == arrays

in an expression the name of an array "decays" into a pointer to element zero†

- array arguments are not passed by copy

these two declarations are equivalent

```
void display(size_t size, wibble * first);  
void display(size_t size, wibble first[]);
```

```
wibble table[42] = { ... };
```

these two statements are equivalent

```
display(42, table);  
display(42, &table[0]);
```



```
const size_t size =  
    sizeof array / sizeof array[0];
```

†except in a sizeof expression

# exercise

what does the following program print?

- why?

```
#include <stdio.h>

int main(void)
{
    int array[] = { 0,1,2,3 };
    int clone[] = { 0,1,2,3 };
    puts(array == clone
        ? "same" : "different");
    return 0;
}
```

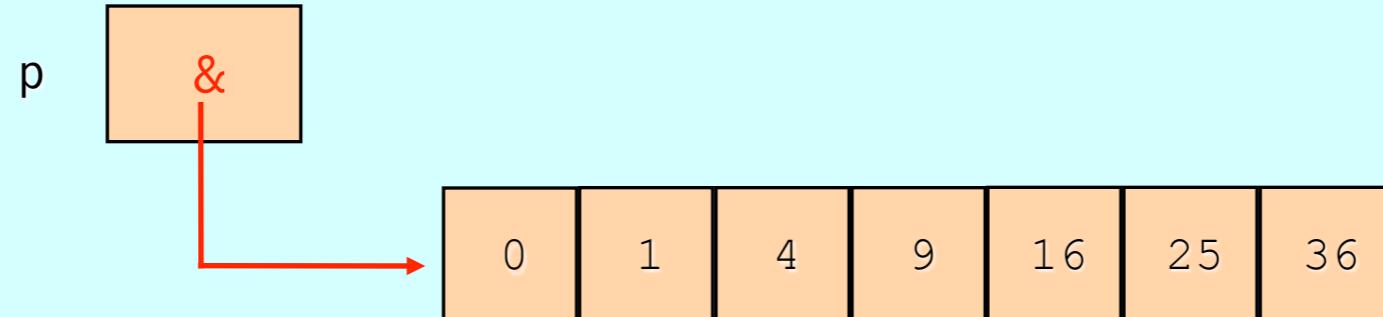
# array literal

an aggregate initializer list can be "cast" to an array type

- known as a compound literal
- can be useful in both testing and production code

c99

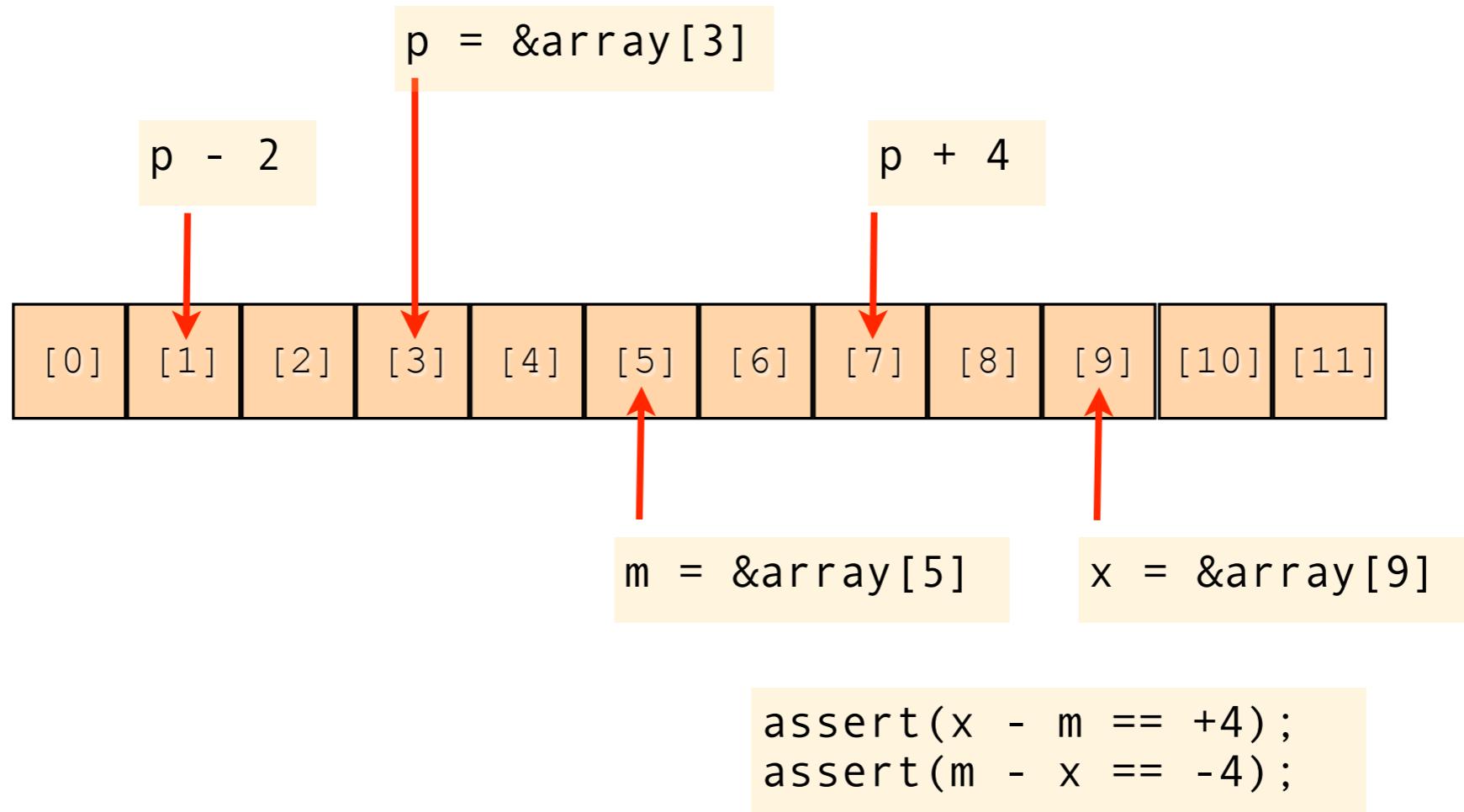
```
int * p = (int []){ 0,1,4,9,16,25,36 };
```



# pointer arithmetic

is in terms of the target type, not bytes

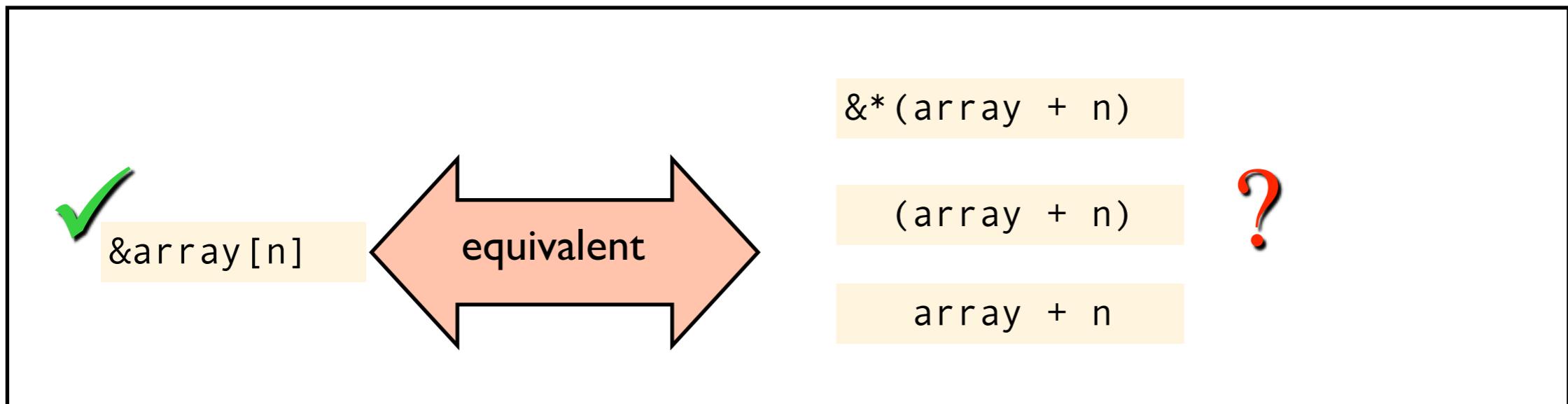
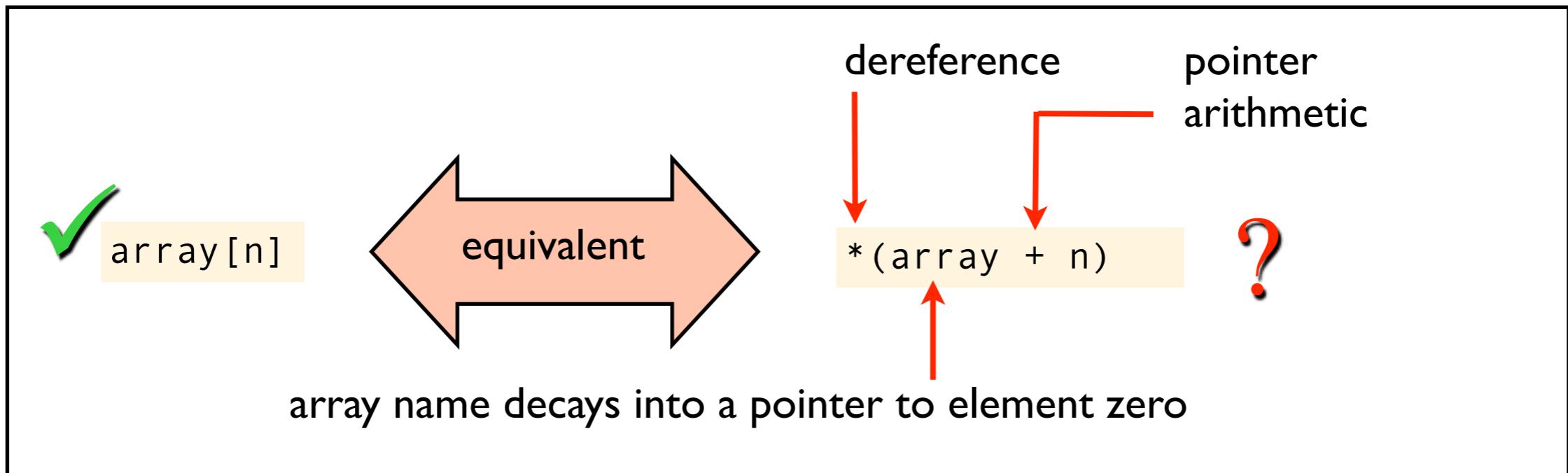
- $p++$  moves  $p$  so it points to the next element
- $p--$  moves  $p$  so it points to the previous element
- $(\text{pointer} - \text{pointer})$  is of type `ptrdiff_t <stddef.h>`



# pointers == arrays

array indexing is syntactic sugar

- the compiler converts  $a[i]$  into  $*(\mathbf{a} + i)$





We know  $a[n]$  is syntactic sugar for  $*(a + n)$

We also know that  
 $a+n == n+a$   
Therefore;

$$*(a + n) == *(n + a)$$

$$*(n + a) == n[a]$$

$$a[n] == n[a]$$

# one beyond the end

a pointer can point just beyond an array

- can't be dereferenced
- can be compared with
- can be used in pointer arithmetic

```
int array[42];
```

this is undefined

```
array[42]
```

this is not undefined

```
&array[42]
```

```
int * search(int * begin, int * end, int find)
{
    int * at = begin;
    while (at != end && *at != find) {
        at++;
    }
    return at;
}
```

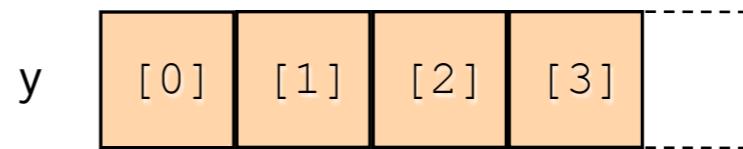
# pointers != arrays

very closely related but not the same

- declare as a pointer → define as a pointer
- declare as an array → define as an array

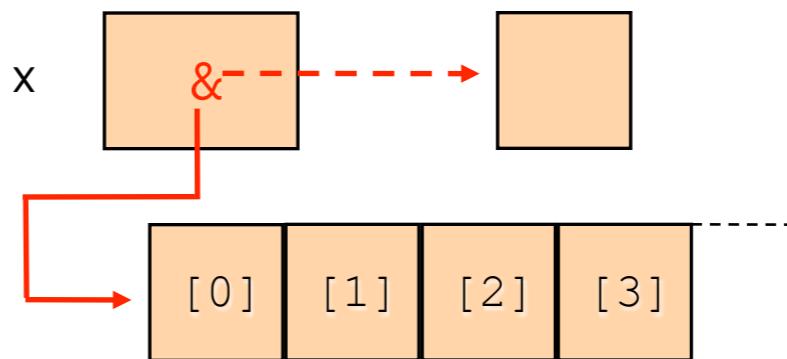
y is an array of int (of unspecified size)

```
extern int y[];
```



x is a pointer to an int (or to an array of ints)

```
extern int * x;
```





C was designed so that the syntax of use  
mirrors the the syntax of declaration

```
int days_in_month[12];  
...days_in_month[at]...
```

```
int *pointer = &variable;  
int copy = *pointer;  
*pointer = 42;
```

```
int identifier;  
typedef int identifier;
```

```
void func(int a, int b);  
func(    4,      2);
```

# pointer confusion

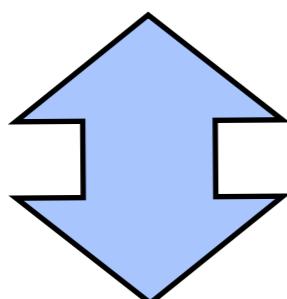
be clear what your expression refers to

- the pointer, the thing the pointer points to, both?

```
int array[42];
int * pointer = &array[0];
```

```
pointer = &array[9];           ← the pointer
pointer++;                   ← the pointer
*pointer = 0;                ← the int the pointer points to
```

```
int v = *pointer++;          ← both!
```



equivalent

```
int v = *pointer;
pointer++;
```

# const + pointer



another notorious source of confusion

- again, be clear what your expression refers to
- read const on the pointer's target as readonly

```
int value = 0;
```

```
int * ptr = &value;  
*ptr = 42;           // ok  
ptr = NULL;         // ok
```

\*ptr is not const ✓

\*ptr is not const ✓

```
const int * ptr = &value;  
*ptr = 42;           // error  
ptr = NULL;         // ok
```

\*ptr must be treated ✗  
as readonly

ptr is not const ✓

# const + pointer



another notorious source of confusion

- again, be clear what your expression refers to
- read const on the pointer's target as readonly

```
int value = 0;
```

```
int * const ptr = &value;  
*ptr = 42;           // ok  
ptr = NULL;         // error
```

\*ptr is not const ✓

ptr is const ✗

```
const int * const ptr = &value;  
*ptr = 42;           // error  
ptr = NULL;         // error
```

\*ptr must be treated as readonly ✗

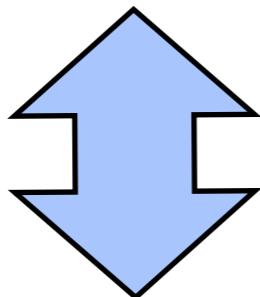
ptr is const ✗

# string literals

strings are arrays of char

- automatically terminated with a null character, '\0'
- a convenient string literal syntax

```
char greeting[] = "Bonjour";
```

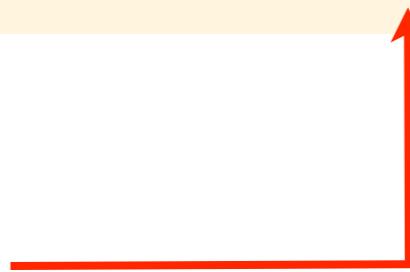


equivalent

```
char greeting[] =  
{ 'B', 'o', 'n', 'j', 'o', 'u', 'r', '\0' };
```



terminating nul character



# string manipulation

strcpy: in <string.h>, copies a string

- why is this a dangerous function to call?

## array version

```
char * strcpy(char dst[], const char src[])
{
    int at = 0;
    while ((dst[at] = src[at]) != '\0')
        at++;
    return dst;
}
```

why are the parentheses needed?  
why is the comparison with '\0' optional?  
note the empty statement here

```
char * strcpy(char * dst, const char * src)
{
    char * destination = dst;
    while (*dst++ = *src++)
        ;
    return destination;
}
```

equivalent pointer version - very terse - typical of C code

# **void \***

a generic object pointer

- any object pointer can be converted to a void\* and back again
- const void\* is allowed
- dereferencing a void\* is not allowed

```
void * generic_pointer;
int * int_specific_pointer;
char * char_specific_pointer;
```

these compile

```
generic_pointer = int_specific_pointer;
int_specific_pointer = generic_pointer;
```



these dont compile

```
*generic_pointer;
generic_pointer[0];
```



# int - pointer conversions

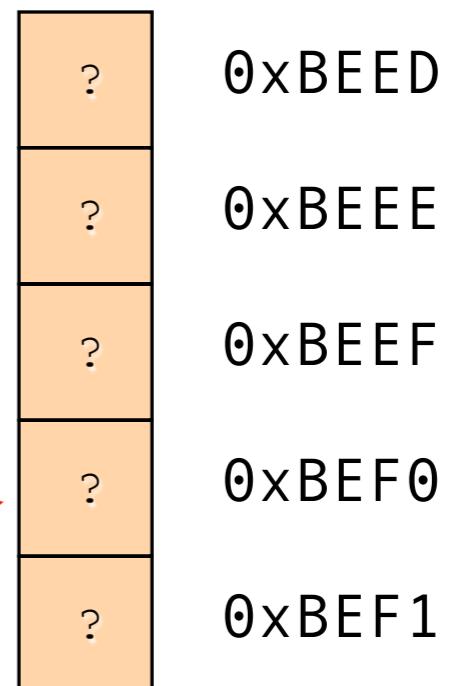
any object pointer can safely be held in...

- `intptr_t` - a signed integer typedef
- `uintptr_t` - an unsigned integer typedef
- both declared in `<stdint.h>`

c99

```
#include <stdint.h>
intptr_t address = 0xBEF0;
int * pointer = (int*)address;
```

int \*  
**0xBEF0**  
pointer



# restrict

applies only to pointer declarations

- type`*restrict` p → \*p is accessed only via p in the surrounding block
- enables pointer no-alias optimizations
- a compiler is free to ignore it

c99

```
void f(int n, int * restrict p, int * restrict q)
{
    while (n-- > 0) {
        *p++ = *q++;
    }
}
```

```
void g(void)
{
    int d[100];
    f(50, d + 50, d); // ok
    f(50, d + 1, d); // undefined-behaviour
}
```

# dynamic memory

dynamic memory can be requested using malloc() and released using free()

- both functions live in <stdlib.h>
- one way to create arrays whose size is not a compile-time constant

```
?#include <stdlib.h>

void dynamic(int n)
{
    void * raw = malloc(sizeof(int) * n);
    if (raw != NULL)
    {
        int * cooked = (int *)raw;
        cooked[42] = 99;
        ...
        free(raw);
    }
}
```

see also calloc() and realloc()

# summary

- pointers can point to...
  - nothing, i.e., null (expressed as NULL or 0)
  - a variable whose address has been taken (&)
  - a dynamically allocated object in memory (from malloc, calloc or realloc – don't forget to free)
  - an element within or one past the end of an array
- pointer arithmetic is scaled
- pointers and arrays share many similarities
  - but they are not the same and the differences are as important as the similarities
- strings are conventionally expressed as arrays of char (or wchar\_t)
  - <string.h> supports many common string-handling operations
- be clear about what you can do with a pointer
  - respect restrict and be clear about what's const

# Structures

# typedefs

defines a new name for an existing type

- does not create a new type
- useful for expressing intention
- can aid portability too

```
unsigned int at;  
typedef unsigned int size_t;
```

syntax mirrors declaration

```
unsigned int at;  
size_t at;
```

equivalent definitions  
(compile time error)

```
void f(unsigned int variable); ?  
void f(size_t variable); ?
```

equivalent declarations  
(allowed)

```
void f(unsigned int variable);  
...  
void f(size_t variable)  
{  
    ...  
}
```

declared without typedef  
defined with typedef  
(allowed)

## enum

an enum definition introduces a new type

- and a sequence of enumerators
- each enumerator is *not* scoped to its enum
- each enumerator has a constant integer value
- by default starts at zero and increments by one

```
enum suit ← type name
{
    clubs, diamonds, hearts, spades ← enumerators
};

enum suit trumps = clubs;
```

```
enum season
{
    spring=1, summer, autumn, fall=autumn, winter
};
```

enumerators with the same value are allowed

# enum typedef

enum on declarations is just noise?

```
enum suit { ... };  
enum suit trumps = clubs;
```

?

better

```
enum suit_tag { ... };  
typedef enum suit_tag suit;  
suit trumps = clubs;
```

?

better

```
enum suit { ... };  
typedef enum suit suit;  
suit trumps = clubs;
```

✓

commonly compressed into this

```
typedef enum suit { ... } suit;  
suit trumps = clubs;
```

✓

tag name no longer required

```
typedef enum { ... } suit;  
suit trumps = clubs;
```

✓

## enum ↔ int

a thinly wrapped integer – not type safe

- *any* int value can be converted to an enum
- an enum can be converted to an int

```
const char * suit_name(suit s)
{
    switch (s)
    {
        case clubs      : return "clubs";
        case diamonds   : return "diamonds";
        case hearts     : return "hearts";
        case spades      : return "spades";
        default         : return NULL; // can happen
    }
}
```

```
int main(void)
{
    suit trumps = (suit)42;
    int value = (int)trumps;
    printf("%s\n", suit_name(trumps));
}
```

neither cast is required

?

# anonymous enums

## useful for designators

```
enum { january, february, ...
       november, december } ;

const int days_in_month[] =
{
    [january] = 31,
    [february] = 28,
    ...
    [november] = 30,
    [december] = 31
};
```

c99

## alternative to #define for array sizes

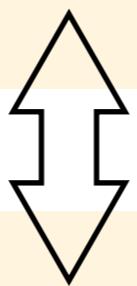
```
#define MAX_LEN (1024)
```

```
char buffer[MAX_LEN];
```

```
enum { max_len = 1024 };
```

```
char buffer[max_len];
```

alternatives



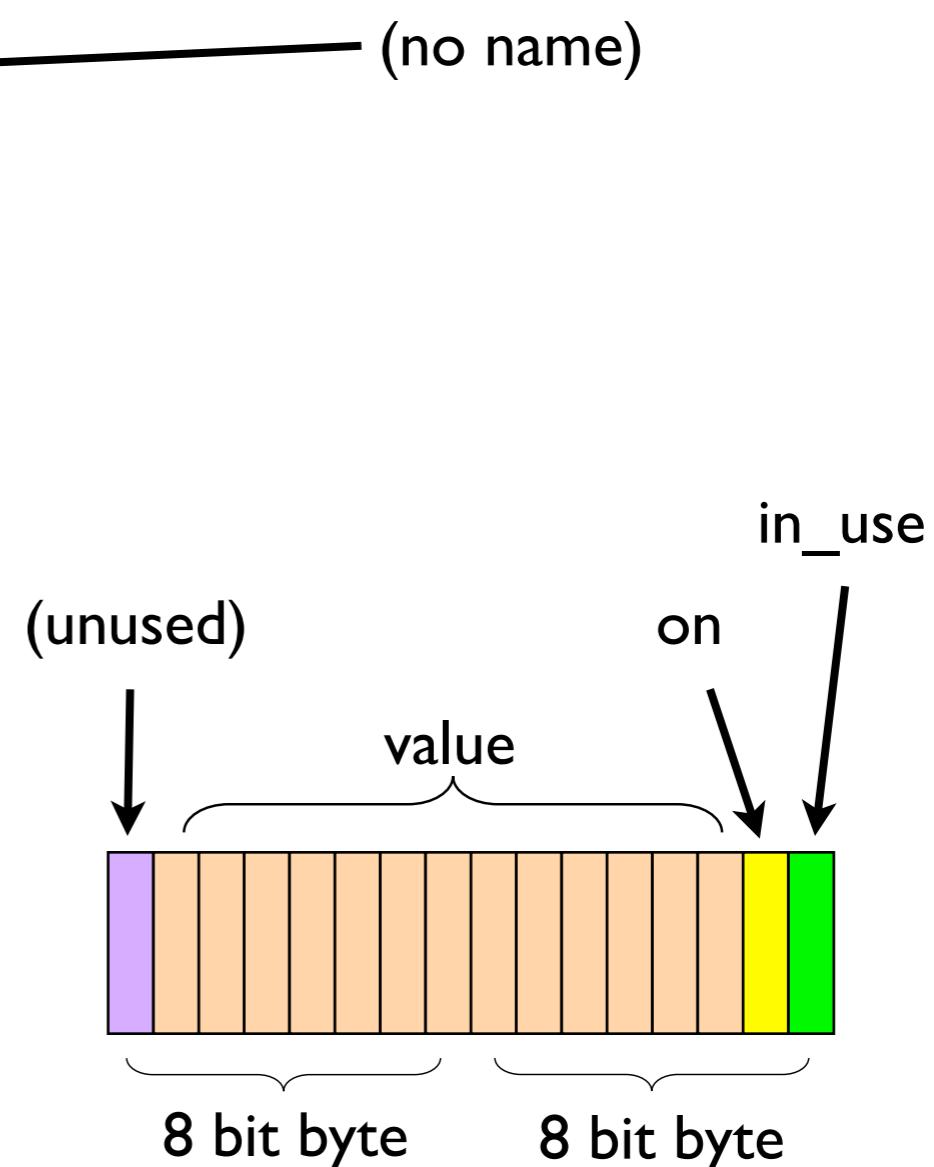
# bit fields

you can use bitfields to control memory allocation right down to the bit level

- compiler dependent; not portable

```
struct fields
{
    unsigned int : 1;
    unsigned int value : 13;
    unsigned int on : 1;
    unsigned int in_use : 1;
};
```

```
fields widget;
...
if (widget.in_use)
    ...
```



# unions

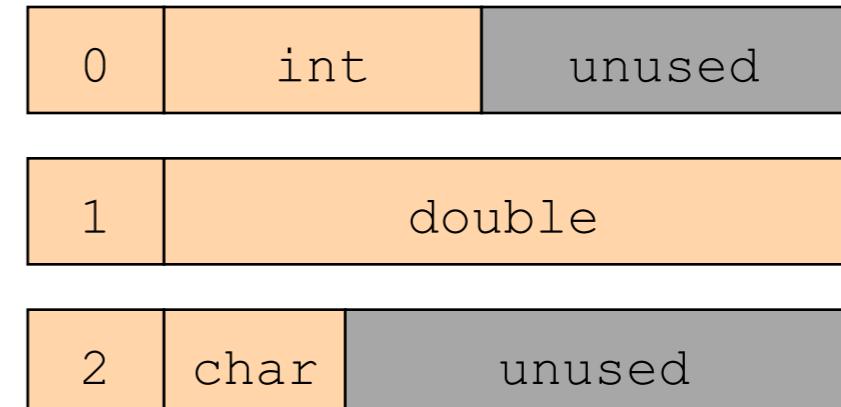
the members of a union overlay one another

- typically used to save memory
- often accompanied by a discriminator

```
enum descrim { int_u, double_u, char_u };
```

```
union spring
{
    int i;
    double d;
    char c;
};

struct ure
{
    enum descrim is_a;
    union spring value;
};
```



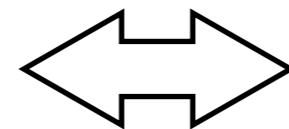
# structs

a struct definition introduces a new type

- aggregation of heterogeneous data members
- structs may contain other structs

```
struct date
{
    int year;
    int month;
    int day;
};
```

equivalent



```
struct date
{
    int year, month, day;
};
```



```
struct project
{
    const char * name;
    struct date deadline;
    ...
};
```

## struct typedef

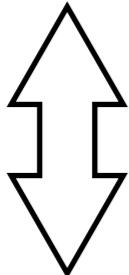
a struct definition introduces a new type

- aggregation of heterogeneous data members
- structs may contain other structs

— struct declarations is “noise”?

```
struct date { ... };  
struct date deadline;
```

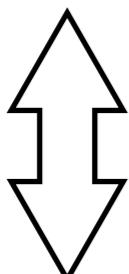
alternatives



use the same tag name

```
struct date { ... };  
typedef struct date date;  
date deadline;
```

equivalent



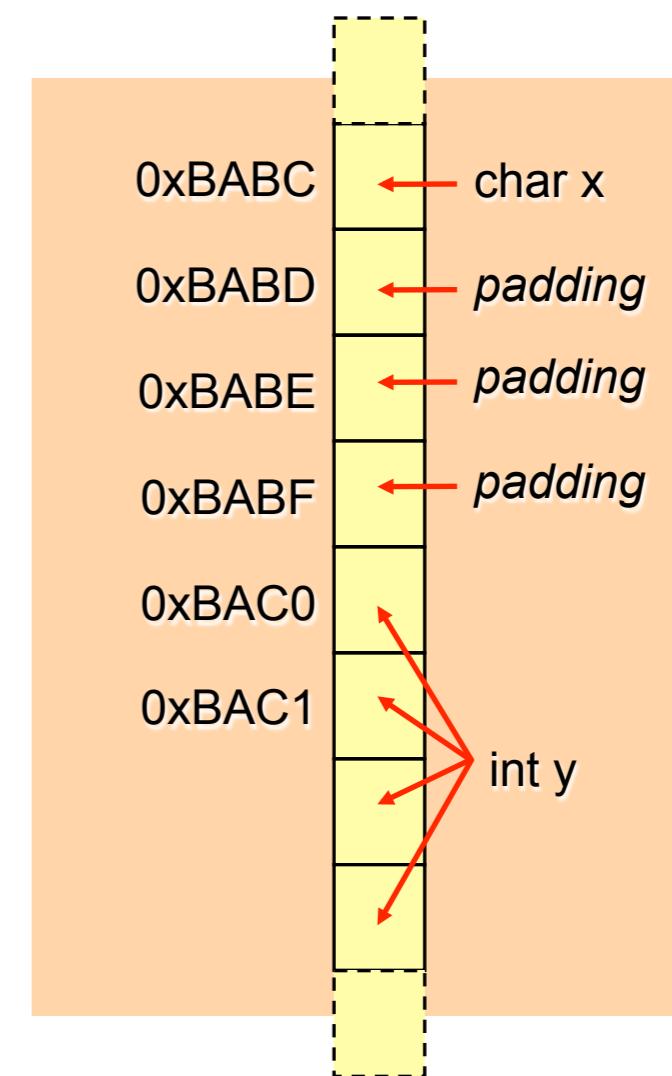
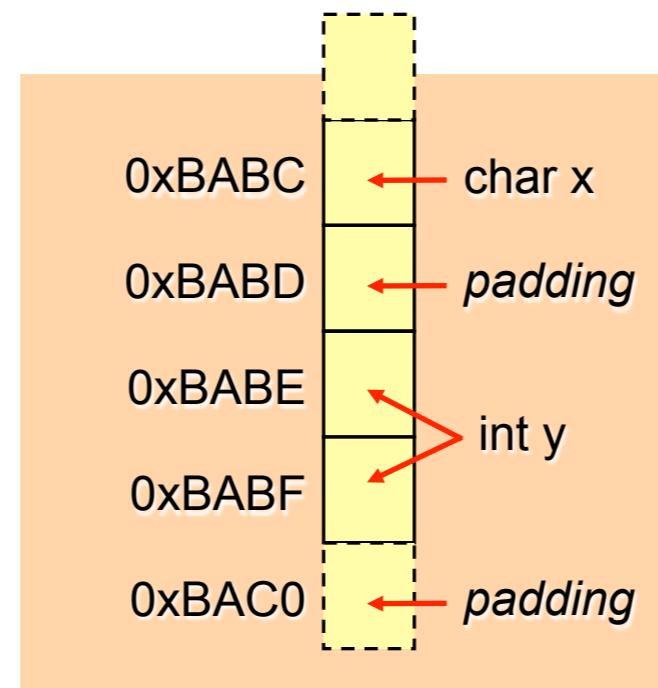
```
typedef struct date { ... } date;  
date deadline;
```

# alignment

types can have alignment restrictions

- first struct member determines alignment
- padding can be added between struct members and after the last struct member

```
struct point
{
    char x;
    int y;
};
```

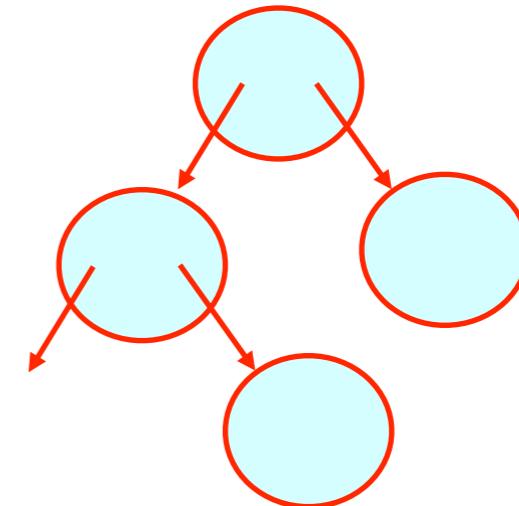


# recursive structs

- a struct S member can be of type struct S \*



```
struct tree_node
{
    int count;
    struct tree_node * left;
    struct tree_node * right;
};
```



- a struct S member can't be of type struct S



```
struct infinite
{
    struct infinite descent;
};
```

TODO: drawing

(compile time error)

# initialization

structs support a convenient initialisation

- allows const struct variables
- not permitted for assignment
- list cannot be empty
- missing fields are default initialised

```
date deadline = { 2008, may, 1 };
```



```
const project fubar =  
    { "fubar", { 2008, may, 1 } };
```



```
const project fubar =  
    { "fubar", { 2008, may, 1 } };
```



```
date deadline = { };
```



# dot operator

the dot operator accesses struct members

- left associative: `a.b.c` means `(a.b).c`
- initialisation/assignment from another struct

```
struct date
{
    int year;
    int month;
    int day;
};
```

```
struct project
{
    const char * name;
    struct date deadline;
    ...
};
```

```
date deadline;
deadline.year = 2008;
deadline.month = may;
deadline.day = 1;
```

```
project fubar;
fubar.name = "fubar";
fubar.deadline = deadline; ←
fubar.deadline.year++;
```

simple bitwise copy

# designators

structs support designator identifiers

- allows list elements to be reordered
- missing members are default initialised

```
struct date
{
    int year;
    int month;
    int day;
};
```

```
date deadline = { .day = 1, .month = may, .year = 2008 };
```

equivalent

```
date deadline = { .month = may, .day = 1, .year = 2008 };
```

c99

# compound literals

aggregate initialization list can be “cast”

c99

- “cast” type becomes type of expression
- assignment works with the “cast”

```
deadline = (date){ 2008, may, 1 };
```

cast works for plain initialiser lists

```
deadline =
    (date){ .year = 2008, .month = may, .day = 1 };
```

cast works for designators

```
call((date){ .year = 2008, .month = may, .day = 1 });
```

```
call((const date){ 2008, may, 1 });
```

also allows creation of anonymous objects

# arrays and structs

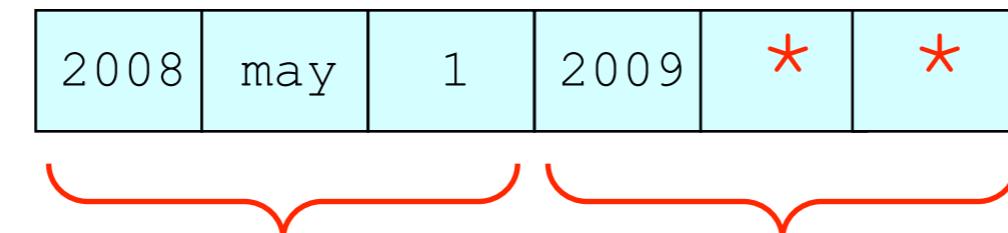
arrays and struct may contain each other

c99

- [int] and .identifier designators can be combined

array of structs

```
date milestones[] =  
{  
    [0] = { 2008, may, 1 },  
    [1].year = 2009  
};
```



milestone[0]

milestone[0]

\* default value

# struct hack

last member may be incomplete array type

- can't be a member of a struct
- can't be an element in an array

```
struct hack
{
    size_t count;
    char message[];
```

incomplete array type

c99

```
size_t n = get_size();
struct hack * ptr = malloc(sizeof(*ptr) + n);

ptr->count = n;
strcpy(ptr->message, "Hello world");
```

?

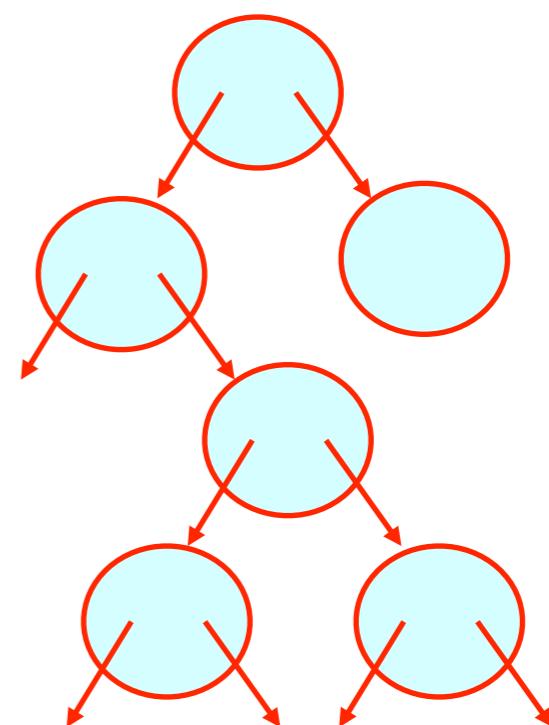
# restrict

c99

the restrict pointer modifier can be used on struct pointer members

- enables the same no-alias optimizations

```
struct tree_node
{
    int count;
    struct tree_node * restrict left;
    struct tree_node * restrict right;
};
```



# arrow operator

`p->m` is an idiomatic shorthand for `(*p).m`

- arrow has same precedence as dot
- associates left to right: `p->q->r == (p->q)->r`
- 

```
date deadline = { 2008, may, 1 };  
date * ptr = &deadline;
```

`*ptr.year = 2008;`  compile time error

`(*ptr).year = 2008;`  non-idiomatic

`ptr->year = 2008;`  idiomatic

## summary

- creating new types is important
- `typedef` does not create a new type
- enums are thinly wrapped integers
- bitfields
- unions for saving memory or expressing mutually exclusive possibilities
- structs are the most common
- structs have a rich aggregate list syntax
- struct hack allows variable length struct
- struct dot operator for values
- struct arrow operator for pointers

# Exercise, day 2 (phone\_book)

Deep C - a 3 day course  
Jon Jagger & Olve Maudal  
(March 27, 2012)

# Create a phone book library

Your task is to create a very simple phone book for names and numbers where you can add and remove entries. The initial implementation should support name based lookup, and it should be possible to print all the entries to the console.

This seems like a very trivial task. No points is given for finishing this exercise as quick as possible. Instead try to explore alternative designs and discuss with others. Use this exercise as a catalyst for discussions about good and bad design decisions. Peek at what the other teams are doing, ask course instructor for suggestions, etc. Think about what you can contribute to the class by demonstrating different techniques and discuss pros and cons.

When working with this exercise you must use a Makefile and you should provide proper unit tests and also write a demo application actually using the phone book.

Please name the source files like this:

```
phone_book.h  
phone_book.c  
phone_book_test.c  
phone_book_demo.c  
Makefile
```

Extra: Make sure all phone numbers are “reachable”, ie that you don’t allow both 911 and 9113456 to be inserted. Use gdb, gcov, gperf and valgrind to analyze your implementation. Can your implementation scale up to millions of entries?

# PreProcessing

```
#include <stdio.h>

#define PP_NARG(...) \
    PP_NARG__(__VA_ARGS__, PP_RSEQ_N())

#define PP_NARG_(...) \
    PP_ARG_N(__VA_ARGS__)

#define PP_ARG_N( \
    _1, _2, _3, _4, _5, _6, N, ...) (N)

#define PP_RSEQ_N() \
    6,5,4,3,2,1,0

int main(void)
{
    printf("%d", PP_NARG(a,b,c,d));
    printf("%d", PP_NARG(a+b,c+d));
    return 0;
}
```

42

## Deep C - a 3 day course

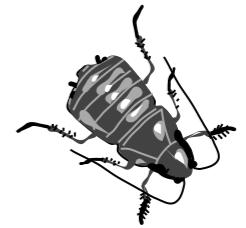
“to get a deeper understanding of the language”

by

Jon Jagger & Olve Maudal  
(Last revised: March 26, 2012)

# Translation Phases

1. multibyte character mapped, trigraphs replaced
2. \ newline deleted to form logical lines
3. decomposed into preprocessing tokens
4. preprocessing directives executed  
(#includes phases 1-4 recursively)
5. source character set and escape sequences mapped
6. adjacent string literals are concatenated
7. preprocessing tokens converted to tokens,  
translation unit is semantically analysed and translated



# gotcha

phase 6: adjacent string literals are concatenated

```
const char * lines[] =  
{  
    "the boy stood on the burning deck" , ←  
    "his heart was all a quiver" , ←  
    "he gave a cough, his leg fell off" , ←  
    "and floated down the river"  
};  
assert(sizeof(lines) / sizeof(lines[0]) == 4);
```

3 commas

```
const char * lines[] =  
{  
    "the boy stood on the burning deck" , ←  
    "his heart was all a quiver" , ←  
    "he gave a cough, his leg fell off"  
};  
assert(sizeof(lines) / sizeof(lines[0]) == 3);
```

2 commas



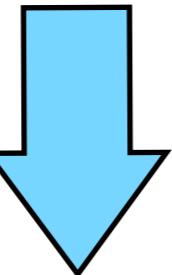
# object macros

you can define an identifier as a macro name with a replacement list

```
# define identifier pp-tokensopt
```

one or more spaces after  
the identifier

preprocesses to...

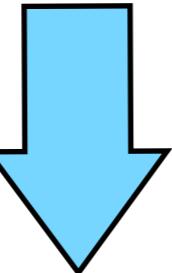


```
#define BUFFER_SIZE (100)
```

```
char buffer[BUFFER_SIZE];
```

```
char buffer[(100)];
```

preprocesses to...



```
#define printf my_printf
```

```
printf("error: %s", message);
```



```
my_printf("error: %s", message);
```

## predefined macros

\_\_func\_\_

- the name of the current function (a string literal) (c99)

\_\_FILE\_\_

- the name of the current source file (a string literal)

\_\_LINE\_\_

- the line number of the current source line (an integer constant)

\_\_DATE\_\_

- the date of translation of the preprocessing translation unit (a string literal)

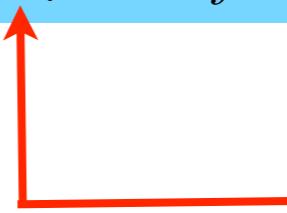
\_\_TIME\_\_

- the time of translation of the preprocessing translation unit (a string literal)

# function macros

a function-like macro accepts arguments

```
# define identifier( identifier-listopt ) pp-tokensopt
```



no space between the identifier and  
the left parentheses

```
#define MAX(a,b) ((a) > (b) ? (a) : (b))
```

```
func(MAX(precision, delta + epsilon));
```

preprocesses to...

```
func(((precision) > (delta + epsilon)  
? (precision) : (delta + epsilon)));
```

# macro guidelines

macro names should use **UPPERCASE** and \_ underscore only

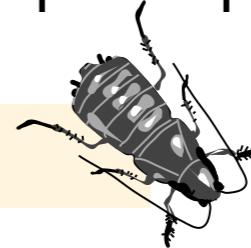
- never lowercase
- this is a very strong convention

```
#define max(a,b) ((a) > (b) ? (a) : (b))
```



this looks like a function call (with a sequence point)  
but it's not, it's a macro :-)

```
max(delta, precision);
```



```
#define MAX(a,b) ((a) > (b) ? (a) : (b))
```

this looks like a function MACRO (without a sequence point)  
and it is indeed a MACRO

```
MAX(delta, precision);
```

# macro guidelines

```
#define MAX(a,b)  a > b ? a : b;
```

don't include a trailing semi-colon



better

```
#define MAX(a,b)  a > b ? a : b
```

put each argument in parentheses



better

```
#define MAX(a,b)  (a) > (b) ? (a) : (b)
```

if the replacement tokens form an expression  
put the whole replacement text in parentheses



better

```
#define MAX(a,b)  ((a) > (b) ? (a) : (b))
```

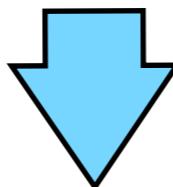
# macro side effects

Macro arguments with side effect can happen multiple times

- Surprising enough even without considering sequence points!

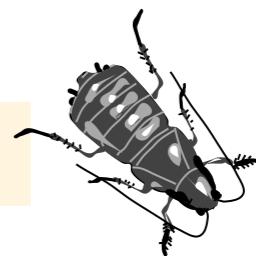
```
#define MAX(a,b) ((a) > (b) ? (a) : (b))
```

```
f(MAX(n++, limit));
```



preprocesses to...

```
f(((n++) > (limit) ? (n++) : (limit)));
```



# macro guidelines

Consider if you can replace the macro with an inline function

```
#define IS_ODD(n) ((n) % 2 == 1)
```



better

```
static inline bool is_odd(int n)
{
    return n % 2 == 1;
}
```

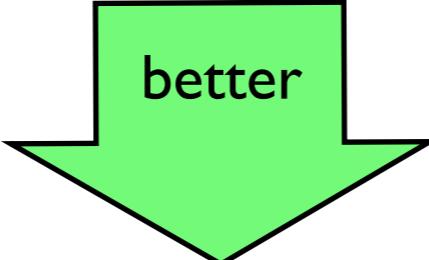


# logical lines

A backslash followed immediately by a newline is deleted (phase 2)

- Allows multiple physical lines to form one logical line
- Used to increase readability of macro replacement

```
#define TRACE(msg) do { if (dbg_mode) puts(msg); } while (0)
```



better

```
#define TRACE(msg) do {  
    if (dbg_mode) \n  
    puts(msg); \n  
} while (0)
```



where `\n` is being used to  
represent a newline character

# macro problem

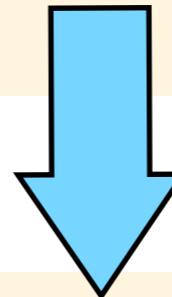
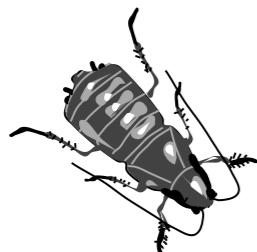
macros bigger than a single expression...

- can easily interfere with their surrounding context

```
#define LOG(msg)  if (in_log_mode) \n    puts(msg)
```

??

```
if (toggled())  
    LOG("on");  
else  
    LOG("off");
```



preprocesses to...

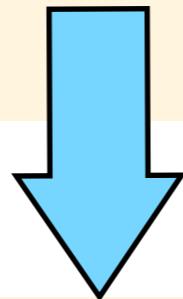
```
if (toggled())  
    if (in_log_mode)  
        puts("on");  
    else if (in_log_mode)  
        puts("off");
```

# macro solution 1

rephrase the logic in a single expression

```
#define LOG(msg) \n\n    ((void) (in_log_mode && puts(msg)))
```

```
if (toggled())\n    LOG("on");\nelse\n    LOG("off");
```



preprocesses to...

```
if (toggled())\n    ((void) (in_log_mode && puts("on")));\nelse\n    ((void) (in_log_mode && puts("off")));
```

# macro solution 2

embed the macro replacement inside a do-while(0) loop

```
#define LOG(msg)  do {  
    if (in_log_mode)  \n  
        puts(msg);  \n  
} while (0)
```

don't include a trailing semi-colon here  
instead put it here at each point of use

```
if (toggled())  
    LOG("on");  
else  
    LOG("off");
```

preprocesses to...

```
if (toggled())  
    do { if (in_log_mode) puts("on"); } while (0);  
else  
    do { if (in_log_mode) puts("off"); } while (0);
```

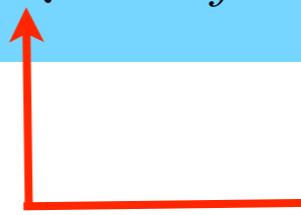
# variadic function macros

c99

can accept a variable number of arguments

**\_VA\_ARGS\_** expands to the elided arguments

```
# define identifier( identifier-listopt , ... ) pp-tokensopt
```

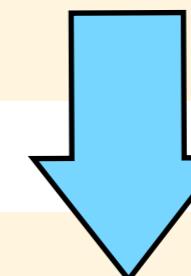


no space between the identifier and  
the left parentheses

```
#define DEBUG(...) fprintf(stderr, __VA_ARGS__)
```

```
DEBUG("error: %s", message);
```

```
fprintf(stderr, "error: %s", message);
```



preprocesses to...



Here's an amazing c99 macro to count how many arguments a function macro is called with!

c99

```
#include <stdio.h>

#define PP_NARG(...) \
    PP_NARG__(__VA_ARGS__, PP_REV_SEQ_N())

#define PP_NARG__(...) \
    PP_ARG_N(__VA_ARGS__)

#define PP_ARG_N( \
    _1, _2, _3, _4, _5, _6, N, ...) (N)

#define PP_REV_SEQ_N() \
    6,5,4,3,2,1,0

int main(void)
{
    printf("%d", PP_NARG(a,b,c,d));
    printf("%d", PP_NARG(a+b,c+d));
    return 0;
}
```

# #include

the commonest directive

- #include X is replaced by the entire contents of X
- >50% of compilation is typically for #inclusions!
- three forms

`# include < h-chars >`

h-char == any character except > or newline

`# include " q-chars "`

q-char == any character except " or newline

`# include pp-tokens`

must expand to < > or " " form

" " for local headers → `#include "widget\table.h"` is this a tab character?

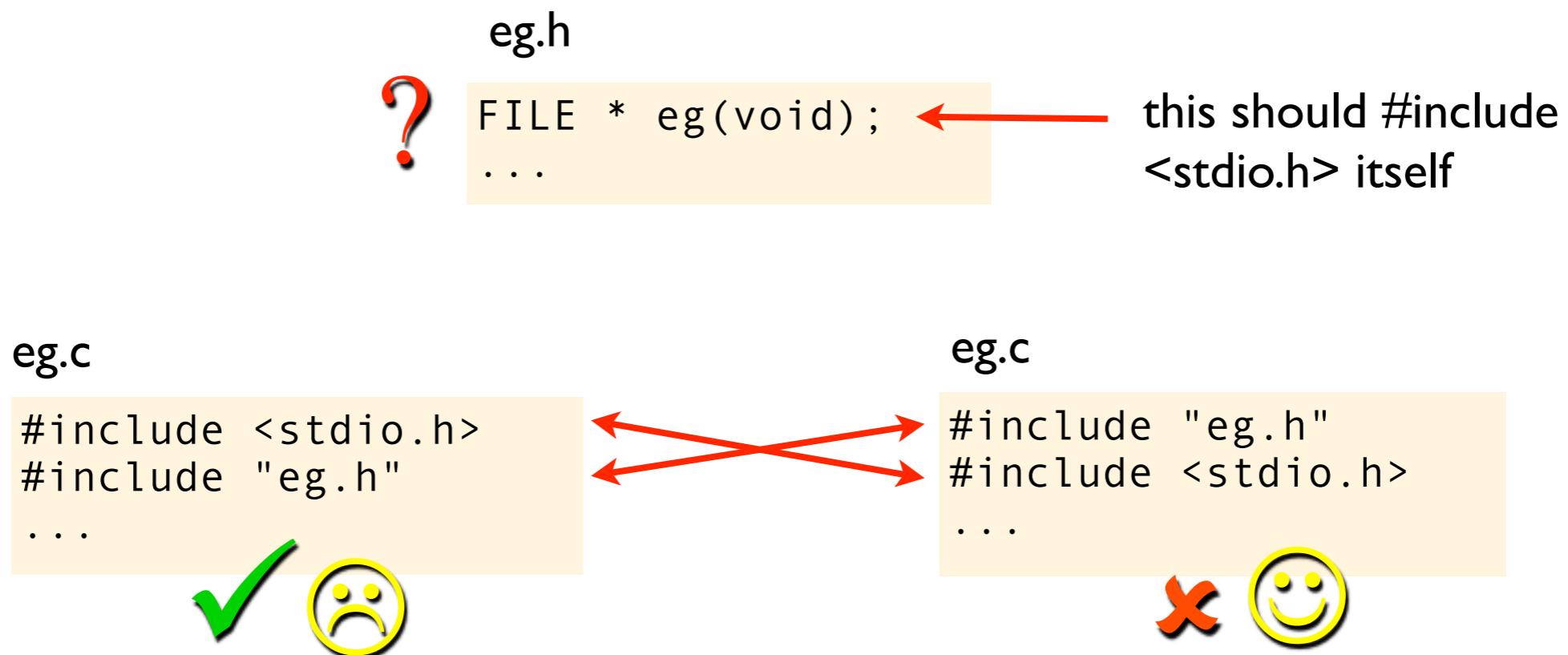
< > for system headers → `#include <stdio.h>` this is not a string literal

rarer → `#include INCFILE`

# include order

a source file should #include its own header before any other header

- this helps to ensure they don't accidentally compile because of a previous #include
- consider compiling each header compiles individually as part of the build



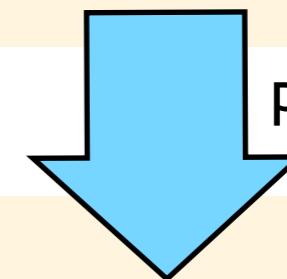
# # operator

the # operator converts its argument to a string literal

- if the argument is a string literal or character constant \ is inserted before " and \

```
#define REPORT(test, ...) \
  ((test) \
   ? puts(#test) \
   : printf(__VA_ARGS__))

REPORT(x > y, "x is %d but y is %d", x, y);
```



preprocesses to...

```
((x > y)
 ? puts("x > y")
 : printf("x is %d but y is %d", x, y));
```



The point of the # operator is to create a string representation of its argument  
*as the developer sees it*

```
#include <stdio.h>

#define STR(arg)    #arg

int main(void)
{
    puts("x\ty");
    puts(STR("x\ty"));
    return 0;
}
```

x y  
x\ty

For example

"x\ty" is { 'x', '\t', 'y', '\0' }

but

STR("x\ty") is { 'x', '\\', 't', 'y', '\0' }

# ## operator

## operator concatenates two arguments

preprocesses to...

```
#define DEBUG(s, t)  printf("x" # s "= %d, " \n
                           "x" # t "= %s", \n
                           x ## s, x ## t)
```

preprocesses to...

```
printf("x" "1" "= %d, " "x" "2" "= %s", x1, x2);
```

phase 6 string literal concatenation

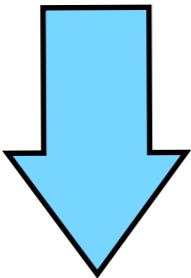
```
printf("x1= %d, x2= %s", x1, x2);
```

# #define #undef

you can define and undefine an identifier as a macro name

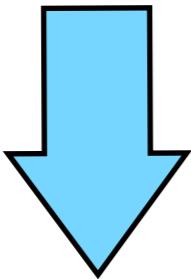
```
# define identifier ...
# undef identifier
```

preprocesses to...



```
#define BUFFER_SIZE /*nothing*/
char buffer[BUFFER_SIZE];
char buffer[];
```

preprocesses to...



```
#define BUFFER_SIZE (100)
#undef BUFFER_SIZE
char buffer[BUFFER_SIZE];
char buffer[BUFFER_SIZE];
```

# conditionals

sections of code can be conditionally included/excluded from preprocessing (and hence from translation)

#elif == #else #if →

```
# if constant-expression  
...  
# elif constant-expression  
...  
# else  
...  
# endif
```

```
#if VERSION == 1  
# define INCFILE "version1.h"  
#elif VERSION == 2  
# define INCFILE "version2.h"  
#else  
# define INCFILE "versionN.h"  
#endif
```

```
#if 0  
...  
#endif
```

how to exclude code when the excluded code contains /\*comments\*/(remember /\* comments \*/ do not nest)

# conditionals

the #if expression can determine if a macro token has been #defined or not

```
# if defined ( identifier )
# if !defined ( identifier )
```

```
# ifdef identifier
# ifndef identifier
```

equivalent

This is the idiomatic way to make header files idempotent †

widget\_table.h

```
#ifndef WIDGET_TABLE_INCLUDED
#define WIDGET_TABLE_INCLUDED
...
...
...
#endif
```

in a single translation  
make sure every  
source  
file has a unique token

† idempotent basically means "once-only"

# #error

the #error directive

- issues the specific diagnostic message
- terminates the translation as a failure
- useful when conditional

```
# error pp-tokensopt
```



diagnostic message

```
#if TARGET == 1
# define INCFILE "version1.h"
#elif TARGET == 2
# define INCFILE "version2.h"
#else
# error "TARGET must be 1 or 2"
#endif
```



# #pragma

c99

causes implementation-defined behaviour

```
# pragma pp-tokensopt
```

also available via the `_Pragma` operator

```
_Pragma pp-tokensopt
```

```
#pragma ivdep /* vectorization hint */  
while (n-- > 0)  
    ...
```

```
#define VECTOR_HINT _Pragma("ivdep")  
VECTOR_HINT  
while (n-- > 0)  
    ...
```

## summary

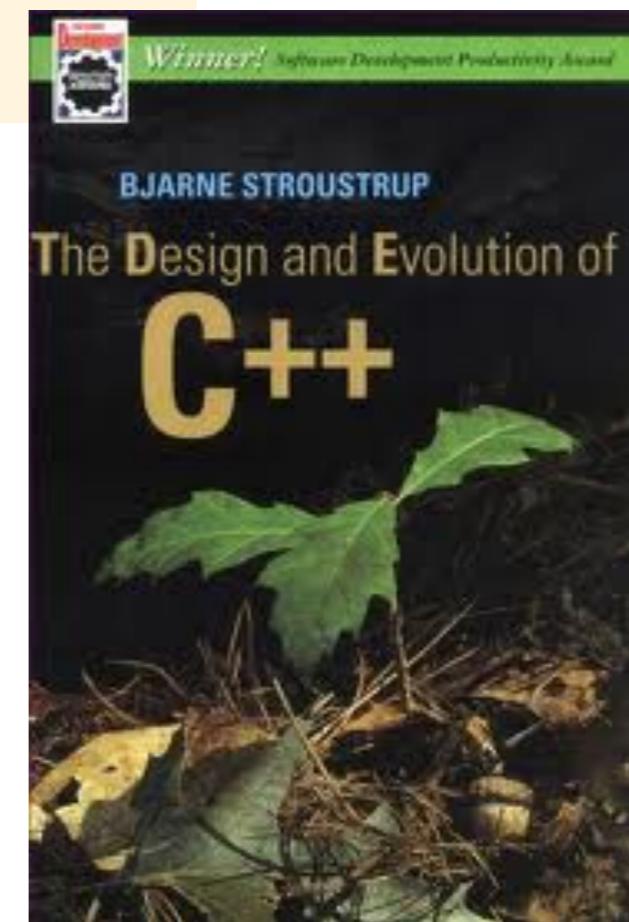
- be wary of the preprocessor
- it knows practically nothing about C
- it silently changes the source being compiled
- header guards and includes are unavoidable
- but for other #directives consider alternatives
  - function-like macro → inline function
  - object-like macro → const variable
  - object-like int macro → enumerator

"I'd like to see Cpp [the C pre processor] abolished."

p426

"In retrospect, maybe the worst aspect of Cpp is that it has stifled the development of programming environments for C."

p424



# Advanced Techniques



**Deep C - a 3 day course**

“to get a deeper understanding of the language”

by

Jon Jagger & Olve Maudal

(Last revised: March 26, 2012)

# the spirit of C



You will get a deeper understanding of C if you know  
the rationale behind its design...

## 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

# the spirit of C

## trust the programmer

- let them do what needs to be done
- the programmer is in charge not the compiler



```
void send(short * to, short * from, int count)
{
    int n = (count + 7) / 8;
    switch (count % 8)
    {
        case 0 : do { *to++ = *from++;
                      } while (--n > 0);
    }
}
```



# the spirit of C

keep the language small and simple

- small amount of code → small amount of assembler
- provide only one way to do an operation



```
void f(int *);  
  
int main(void)  
{  
    int array[] = { 42 };  
    f(array); ← ✓   
  
    int copy[] = { 42 };  
    array = copy; ← ✗   
}
```

# the spirit of C

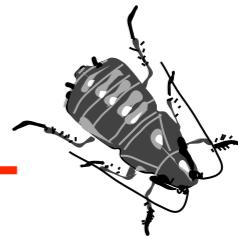


**make it fast, even if its not portable**

- target efficient code generation
- int preference, int promotion rules
- sequence points, maximum leeway to compiler

```
int main(void)
{
    int n = 42;

    printf("%d", ++n + n); ←
}
```



# compile time asserts



Sometimes you'd like to verify some assumptions at compile time rather than at runtime.

Despite appearances, you **cannot** do this using the preprocessor

```
struct widget  
{  
    ...  
};
```

```
#if sizeof(struct widget) > 42  
#error widget is too big  
#endif
```



# compile time asserts

However, you can write C constructs that don't compile if the assumption is false...



cta.h

```
#define COMPILE_TIME_ASSERT(assertion) \
    extern char CTA_NAME [ (assertion) ? 1 : -1 ]\n\n#define CTA_NAME \
    CTA_GLUE.compile_time_assert_at_line_, __LINE__\n\n#define CTA_GLUE(lhs, rhs)      CTA_GLUE_2(lhs, rhs)\n\n#define CTA_GLUE_2(lhs, rhs)  lhs ## rhs
```

```
#include "cta.h"\n\nCOMPILE_TIME_ASSERT(sizeof(struct date) > 42);
```

**bad typedef**



# bad typedef

Hiding a pointer in a `typedef` is not advisable.  
If it's a pointer make it look like a pointer.  
Abstraction is about hiding *unimportant* details.



# bad typedef

Hiding a pointer in a `typedef` is not advisable.  
If it's a pointer make it look like a pointer.  
Abstraction is about hiding *unimportant* details.



In this code fragment what is `const`?

```
typedef struct date_tag * date;  
  
void f(const date ptr);
```

# bad typedef

Hiding a pointer in a `typedef` is not advisable.  
If it's a pointer make it look like a pointer.  
Abstraction is about hiding *unimportant* details.



In this code fragment what is `const`?

```
typedef struct date_tag * date;  
  
void f(const date ptr);
```

Is the object pointed to `const`?

```
struct date_tag;  
  
void f(const struct date_tag * ptr);
```

# bad typedef

Hiding a pointer in a `typedef` is not advisable.  
If it's a pointer make it look like a pointer.  
Abstraction is about hiding *unimportant* details.



In this code fragment what is `const`?

```
typedef struct date_tag * date;  
  
void f(const date ptr);
```

Is the object pointed to `const`?

```
struct date_tag;  
  
void f(const struct date_tag * ptr);
```

Or is the pointer `const`?

```
struct date_tag;  
  
void f(struct date_tag * const ptr);
```

# bad typedef

Hiding a pointer in a `typedef` is not advisable.  
If it's a pointer make it look like a pointer.  
Abstraction is about hiding *unimportant* details.



In this code fragment what is `const`?

```
typedef struct date_tag * date;  
  
void f(const date ptr);
```

Is the object pointed to `const`?

```
struct date_tag;  
  
void f(const struct date_tag * ptr);
```

Or is the pointer `const`?

```
struct date_tag;  
  
void f(struct date_tag * const ptr);
```

It's the pointer!

**strong typedef?**



# strong typedef?

Remember, a **typedef** does not create a new type.  
A **typedef** does **not** give you any type safety.



```
typedef int mile;
typedef int kilometer;

void weak(mile lhs, kilometer rhs)
{
    lhs = rhs; ← ✓ ☹
```

# strong typedef?

Remember, a **typedef** does not create a new type.  
A **typedef** does **not** give you any type safety.



```
typedef int mile;
typedef int kilometer;

void weak(mile lhs, kilometer rhs)
{
    lhs = rhs; ← ✓ ☹
```

An alternative technique is to  
use a **struct wrapper** instead...

```
typedef int mile;
typedef int kilometer;

void weak(mile lhs, kilometer rhs)
{
    lhs = rhs; ← ✗ ☺
```

# strong enum?

Remember, enums are thinly wrapped ints.  
Enums also do **not** give you any type safety.



```
enum suit
{
    clubs, diamonds, hearts, spades
};

enum season
{
    spring, summer, autumn, winter
};

void weak(enum suit lhs, enum season rhs)
{
    lhs = rhs; ← ✓ ☹
}
```

# strong enum



Again, an alternative technique is to use a struct wrapper instead...

```
struct suit {  
    enum {  
        clubs, diamonds, hearts, spades  
    } value;  
};
```

```
struct season {  
    enum {  
        spring, summer, autumn, winter  
    } value;  
};
```

```
void weak(struct suit lhs, struct season rhs)  
{  
    lhs = rhs; ← ✘ ☺  
}
```

# modules?



In most APIs the idea that a set of functions are part of a module is quite weakly expressed.  
e.g. the f prefix on stdio.h function names

```
typedef struct FILE FILE;
struct FILE { ... };
FILE * fopen(const char *, const char *);
...
int fclose(FILE *);
```

```
#include <stdio.h>

int main(int argc, char * argv[])
{
    FILE * in = fopen(argv[1], "r");
    ...
    fclose(in);
}
```

# modules

A companion struct containing function pointers offers an alternative.



```
typedef struct FILE FILE;  
  
struct FILE { ... };  
  
struct file_api  
{  
    FILE * (*open)(const char *, const char *);  
    ...  
    int (*close)(FILE *);  
};  
extern const struct file_api file;
```

```
#include <stdio.h>  
  
int main(int argc, char * argv[])  
{  
    FILE * in = file.open(argv[1], "r");  
    ...  
    file.close(in);  
}
```

# header dependencies



#include dependencies are transitive  
if you change a header file you have to  
recompile all files that #include it at any depth.  
A visible reflection of the physical coupling

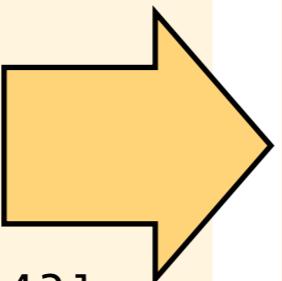
```
#include "flange.h"
#include "grommit.h"
...
struct wibble
{
    struct grommit w[42];
    struct flange f;
    ...
};
```

# header dependencies



#include dependencies are transitive  
if you change a header file you have to  
recompile all files that #include it at any depth.  
A visible reflection of the physical coupling

```
#include "flange.h"
#include "grommit.h"
...
struct wibble
{
    struct grommit w[42];
    struct flange f;
    ...
};
```



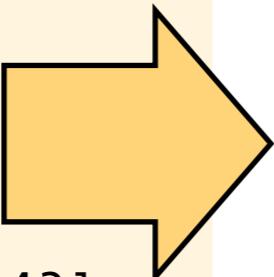
```
#include "sink.h"
#include "washer.h"
...
struct grommit
{
    struct sink dest;
    struct washer * w;
    ...
};
```

# header dependencies

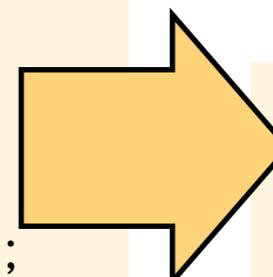


#include dependencies are transitive  
if you change a header file you have to  
recompile all files that #include it at any depth.  
A visible reflection of the physical coupling

```
#include "flange.h"
#include "grommit.h"
...
struct wibble
{
    struct grommit w[42];
    struct flange f;
    ...
};
```



```
#include "sink.h"
#include "washer.h"
...
struct grommit
{
    struct sink dest;
    struct washer * w;
    ...
};
```



# header dependencies



#include dependencies are transitive  
if you change a header file you have to  
recompile all files that #include it at any depth.  
A visible reflection of the physical coupling

```
#include "flange.h"
#include "grommit.h"
...
struct wibble
{
    struct grommit w[42];
    struct flange f;
    ...
};
```

```
#include "sink.h"
#include "washer.h"
...
struct grommit
{
    struct sink dest;
    struct washer * w;
    ...
};
```

# header dependencies



#include dependencies are transitive  
if you change a header file you have to  
recompile all files that #include it at any depth.  
A visible reflection of the physical coupling

```
#include "flange.h"
#include "grommit.h"
...
struct wibble
{
    struct grommit w[42];
    struct flange f;
    ...
};
```

```
#include "sink.h"
#include "washer.h"
...
struct grommit
{
    struct sink dest;
    struct washer * w;
    ...
};
```

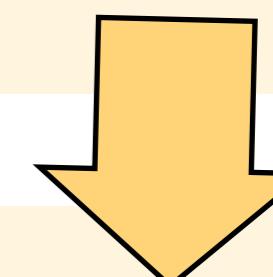
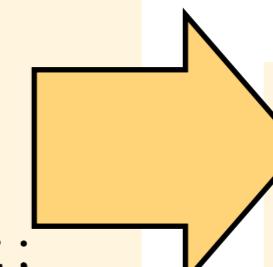
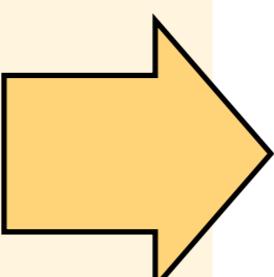
# header dependencies



#include dependencies are transitive  
if you change a header file you have to  
recompile all files that #include it at any depth.  
A visible reflection of the physical coupling

```
#include "flange.h"
#include "grommit.h"
...
struct wibble
{
    struct grommit w[42];
    struct flange f;
    ...
};
```

```
#include "sink.h"
#include "washer.h"
...
struct grommit
{
    struct sink dest;
    struct washer * w;
    ...
};
```



# header dependencies



# header dependencies



# including a struct definition tells the compiler three things

# header dependencies



# including a struct definition tells the compiler three things

I. The name of the struct

This is needed whenever the code mentions the name of the struct!

# header dependencies



# including a struct definition tells the compiler three things

**I. The name of the struct**

This is needed whenever the code mentions the name of the struct!

**2. The definition of the struct**

This is needed whenever the code accesses struct member using the . and -> operators

# header dependencies



# including a struct definition tells the compiler three things

**1. The name of the struct**

This is needed whenever the code mentions the name of the struct!

**2. The definition of the struct**

This is needed whenever the code accesses struct member using the . and -> operators

**3. The size of the struct**

This is needed whenever the code creates an instance of the struct

# forward declaration



# forward declaration



In situations where the compiler needs to know **only** the name of the struct you can replace a #include with a forward declaration.

# forward declaration



In situations where the compiler needs to know **only** the name of the struct you can replace a #include with a forward declaration.

Which of the 6 cases below require a #include and which require only a forward declaration?

#include "washer.h"

struct washer;

...

struct wibble

{

    struct washer     value\_member;        // 1

    struct washer \* pointer\_member;        // 2

};

struct washer     value\_return(void);     // 3

struct washer \* pointer\_return(void);    // 4

void    value\_parameter(struct washer ); // 5

void pointer\_parameter(struct washer \*); // 6

?

# forward declaration



Only a value data member requires a #include!  
Remember, 3,4,5 and 6 were function prototypes,  
not function definitions, not function calls.

```
#include "washer.h" ← ✓ ☺  
  
struct wibble  
{  
    struct washer value_member;           // 1  
};
```

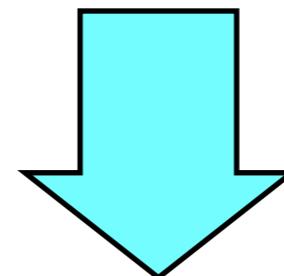
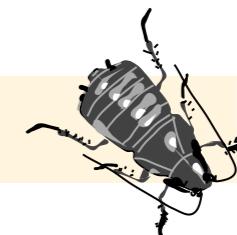
```
struct washer; ← ✓ ☺  
  
struct wibble  
{  
    struct washer * pointer_member;       // 2  
};  
  
struct washer     value_return(void);    // 3  
struct washer *  pointer_return(void);   // 4  
  
void   value_parameter(struct washer ); // 5  
void  pointer_parameter(struct washer *); // 6
```

# forward declaration gotcha

Function prototypes have their own namespace.  
This means the two code fragments below are **not** equivalent.  
Make sure you forward declare a type *before* using it in a function prototype.



```
void pointer_parameter(struct washer *);
```

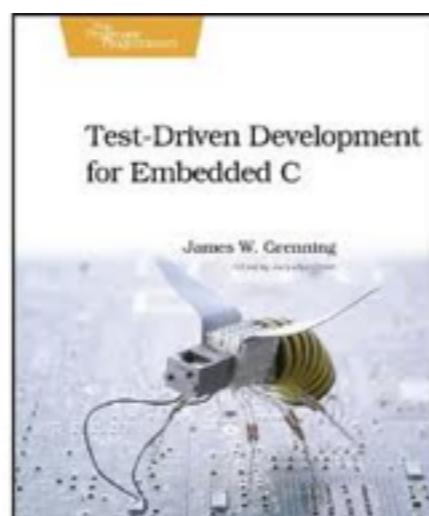


```
struct washer;
```

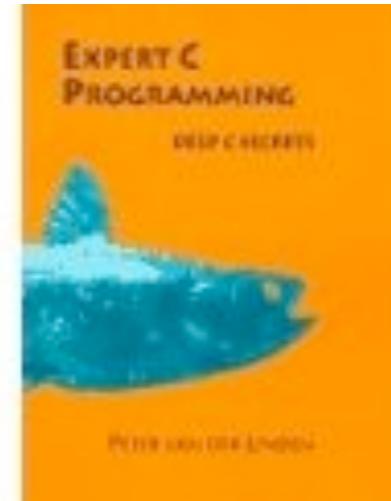
```
void pointer_parameter(struct washer *);
```



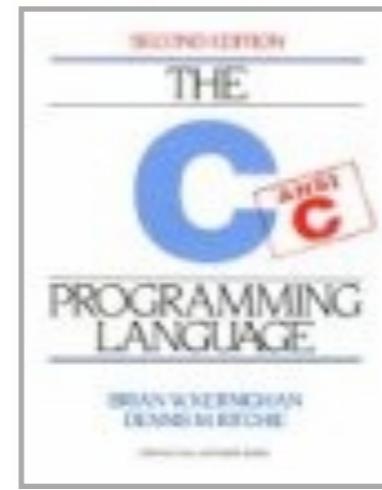
# recommended reading



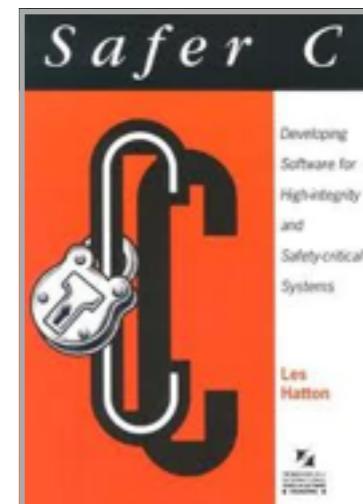
James Grenning



Peter van der Linden



Dennis Ritchie



Les Hatton

!