Preprocessor

Breaking Down the Compilation Process

gcc triangle.c test_triangle.c –o test_triangle	**gcc -c triangle.c (resulting in triangle.o)** **gcc –c test_triangle.c (resulting in test_triangle.o)** **gcc triangle.o test_triangle.o –o test_triangle**

The Compilation Process

Processor	Input	Output
Text Editor	Program typed from keyboard	C source code (.c file)
Preprocessor	C source code file (.c file)	C source code with the preprocessing directories sorted out
Compiler	C source code with the preprocessing directories sorted out (.c file)	Object code (.o file)
Linker	Object code file and standard C library functions (.o file)	Executable code in machine language

Preprocessor

- Special feature in C
- Analyzes our C program before it’s passed to the compiler
- Preprocessor commands are known as directives
  - #include is a preprocessor directive
- gcc has its own preprocessor

Preprocessing

- Removal of comments since they will not be compiled
- Finds all the preprocessing directives
  - lines that begin with #
- For #include statements followed by angular brackets <>, preprocessor looks for the files in the system folders and places the contents of the file where the directive is
  - #include <stdio.h>
  - Function proto

```c
#include <stdio.h>

double circle_area(double radius);
double cylinder_volume(double radius, double height);

int main() {
    double radius = 10.0;

    printf("The area of a circle with radius %lf is %lf\n", radius,
           circle_area(radius));

    double height = 5.0;

    printf("The volume of a cylinder with radius %lf and height %lf is %lf\n",
           radius, height, cylinder_volume(radius, height));

    return 0;
}
```

```
double circle_area(double radius) {
    return 3.1415 * radius * radius;
}

double cylinder_volume(double radius, double height) {
    return 3.1415 * radius * radius * height;
}
```

Magic Number

Numbers that mean something but have no name associated with it are known as magic number

```
double circle_area(double radius) {
    return 3.1415 * radius * radius;
}

double cylinder_volume(double radius, double height) {
    return 3.1415 * radius * radius * height;
}
```

macro

- Another preprocessing directive
- #define IDENTIFIER replacement
  - #define PI 3.1415

macro

```c
#include <stdio.h>
#define PI 3.1415
```

```c
double circle_area(double radius) {
    return PI * radius * radius;
}

double cylinder_volume(double radius, double height) {
    return PI * radius * radius * height;
}
```

Macro expansion

gcc -E shapes.c

```
double circle_area(double radius) {
    return 3.1415 * radius * radius;
}

double cylinder_volume(double radius, double height) {
    return 3.1415 * radius * radius * height;
}
```

Dividing the program into module

- circle.h is included in shapes.c and cylinder.h
- Redundant prototype for circle_area after preprocessing

gcc -E shapes.c

```
# 2 "shapes.c" 2

# 1 "./circle.h" 1


double circle_area(double radius);
# 4 "shapes.c" 2
# 1 "./cylinder.h" 1
# 1 "./circle.h" 1


double circle_area(double radius);
# 2 "./cylinder.h" 2

double cylinder_volume(double radius, double height);
```

Include Guard

```
#ifndef CIRCLE_H
#define CIRCLE_H

// This preprocessor directive is a macro which replaces every instance of the
// identifier PI with 3.14159265
#define PI 3.14159265

double circle_area(double radius);

#endif
```

Include Guard

- If the preprocessor is including the contents of a file, first it checks to see if the identifier, for example, CIRCLE_H has not been defined.
- If it has not, it defines CIRCLE_H and the contents of the file are included.
- If CIRCLE_H is already defined, nothing will be included. This prevents header files from being included multiple times.

Makefile

```
CFLAGS=-std=c99 -Wall

shapes: shapes.o cylinder.o circle.o
    gcc $(CFLAGS) shapes.o cylinder.o circle.o -o shapes

shapes.o: shapes.c cylinder.h circle.h
    gcc $(CFLAGS) -c shapes.c

cylinder.o: cylinder.c cylinder.h circle.h
    gcc $(CFLAGS) -c cylinder.c

circle.o: circle.c circle.h
    gcc $(CFLAGS) -c circle.c

clean:
    rm shapes.o cylinder.o circle.o shapes
```

e09- stats_module

- Implement range() function to find the range of values in an array
- Instead of using max() and min() which use two separate loops, you'll be implementing range() with a single loop

e10-Fibonacci

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

Source: Wikipedia

e10-Fibonacci

- Your program must be organized in the following files:
- **fibonacci.h** - Contains the prototype and documentation for your **fibonacci()** function, with an include guard.
- **fibonacci.c** - Contains the implementation of your **fibonacci()** function.
- **main.c** - Contains **main()** which gets the value of n from the user, uses
- your **fibonacci** function to calculate the nth Fibonacci number, and prints it
- out.
- **Makefile** - Builds the project to create the executable fibonacci. Creates the
- object file **fibonacci.o**, which is then compiled and linked with **main.c**. This file has no extension, it is simply called **Makefile**.