**Functions, Modules, and Compiling:**

**------------------------------------------------**

**Functions:**

- int main(void) is a function, we have already used them

Syntax:

return-type fcn\_name (parameters(s))

{ statements/body}

-return type must be specified in C99 and later; it is void if the function does not return anything

- fcn\_name follows the same rules as function name

- parameters must have types declared per variable (int a, int b) not (int a, b)

-can call functions without parameters

- return ends the functions and gives back the value after it (return 0 will give back 0 from the function)

1. int max (int a , int b){
2. return a > b ? a : b;
3. }
4. int main(void){
5. printf("%d", max(5,10));
6. return 0;
7. }

in this case, a and b are parameters, 5 and 10 are arguments. A copy of the arguments are assigned to the parameters

stack: main ---(writing return address)---(space allocated for a,b )---> max

- the brackets above are new frames that represent memory stored

return address is a specific code that will be stored for the return value so that it can be called back

NOTE: if you don't want anything to be returned, instead of int put void and don't have a return

NOTE: be careful with where the return statements are, if inside of for loop it will return after the first time

**Boolean Variables:**- no boolean variables in C

- in C99 and later <stdbool.h> will give boolean variables

- still 0 or 1, but assigned to a variable

1. bool isPrime(int n){
2. //prime functions (contains if statement with return false)
3. return true;
4. }

- The bool library makes our code easier to read, so instead of 0 or 1 we can put true or false

**Function Declarations:**

- C doesnt force you to define variable before using it

- C guesses the return type of the funciton and defaults to int

- can include function declaration, a sort of promise that it will be defined later

**Assert:**

assert(expr)

- if expr is true, this line does nothing

- otherwise, terminates program

- great for debugging

- also good to leave it in, helps to remember assumptions and will cause program to fail loudly or quitly

- good for regression testing, that is checking that changes havent broken anything in another part of the code

1. bool isleapyear(int year){
2. assert (year > 1752);
3. if (arguments and program)
4. int main (void){
5. print is or isnt a leap year)

**Separate Compilation:**

- in real world programs are coded by many programmers

- can be inefficient to all be working on the same file, not to mention it can get very confusing when theres a lot of code

- by modularizing design we can reduce compile time

**Powers.h:**

powers.h is a header file with some function definiteions

#ifndef POWERS\_H //prevents multiple inclusion

#define POWERS\_H

includes square(), cube(), quartic(), cubic()

#endif

by doing the syntax above, we avoid two files being defined with the same name, and it checks at first if the file is already defined

to ensure that the file is unique, just include filename\_H, and it will ensure that if the file is already put it will just use the original function

**Powers.c:**

#include "powers.h" <--notice the quotes (our model, not c library)

int square (int num) {return num \* num;}

int cube (int num) {return num\*num\*num;}

return cube(int num)\*square(int num)

this reduces compile time

**Compiling:**

To compile, use

gcc -o powers powers.c main.c

- This compiles code into assembly code then object code and then automatically passed through linker which merges them into one file

- the -o command will give a meaningful name to the compiled file, as if not c will automatically do a.out as the filename

- if you just want to compile and not link the files, use -c

**Macros:**

- #include, #ifndef, #define

- the preprocessors

- will check for certain cases before processing the file or code

eg)

#define PI 3.1415

in the code, we can use this all over the file as a constant, so that when the compiler is run, they will replace anhy part of the code with PI with the expression written after