Lecture 16 – COMP500

Modularity

Functions –

* Every c program you have written so far has had one function.

The entry point, **main**, is a function.

Each function has a: Name, parameter (input data), and return type (output data).

Int main(void)

* Name = main
* Input = none or void
* Int = the type of output data.

Invoking a function:

To run a function, it must be *invoked* (also known as *called*)

Scanf, printf, rand, srand, time, atoi, powf, sprint, ect…..

To call a function, we must specify the name of the function, and provide the required input.

The () is how to call the function.

Call, Caller, Callee.

Call:

The invoke of the function

Caller:

The function that function that invoked the function currently running.

Callee:

The function being called.

The return keyword:

Used to stop the execution of a function and return an output.

Void:

The keyword void represents a type of data. But it stores no data.

It can be used to indicate that a function takes no arguments.

You can change the main from int to void. So that it passes nothing back. No data to be passed back. You have to change return 0; to return;

Creating our own functions:

* Modular program.

Good funtions:

* Clear purpose
* A name that reflects its purpose
* Conduct the actions that result in fulfilling its purpose.
* Do a single job.

Hence good functions would be reusable.

* Easier to understand, use and call.

Defining a function:

Void print\_welcome (void)

* Takes nothing in (void), when it’s done the job, it returns nothing cause of void. You can put one step in it.

Void print\_welcome (void)  
{  
 Printf(“welcome”)  
 return;  
}

Int main(void)  
{  
 print\_welcome();  
 return 0;  
}

[output] [name] [input] <-------------------- this is a function. The suntax.

IN DEBUGGER: Clicking F11. Which means if the line of code is a function, it will step through the function.

Beware, the order of function definitions matter:

* You have to define the function first before calling the function. This is specific so far to C (other languages might work differently).
* You have to define the function OUTSIDE the int main(void) because that’s a function within a function.

Prototypes: declaring a function without instructions:

Void print\_welcome(void);

You can set the instructions later. Proper practise is setting the instructions after the int main(void). But setting the name, output and input at the start. Simply define it.

Black box: Function Declaration. You simply define it. You can write what it is and no need for instructions. You can write those later.

A linker is something that finds where the instructions of a declared function is.

ERROR: LINK2019 – Unresolved external symbole\_print\_welcome referenced in function\_main. This means there is no instructions that are meant to be linked to a function you’ve called.

The Call Stack –

Stores all the information about the active functions of a program when executing.

* Such as input to the function
* The data that will be returned from the function.
* Variables that are declared in the function (local variables) …

The call stack can be examined during debugging to see:

* The information about and in the active functions.

Stack frame

* Activation records or activation frames.
* When a function is called, the function is destroyed, when declared, it is created.

#include <stdio.h>

Void print\_bar(void);  
void print\_middle(void);

Int main (void)  
{

Print\_bar();

For(int k = 0; k < 3; ++k)  
{

Print\_middle();  
}

}  
void print\_bar(void)  
{  
 printf(“+---------+”)  
}

Void print\_middle(void)  
{  
 printf(“| |”)  
}