## Introduction to C & IDEs

- See online notes on Visual Studio and Xcode

### Hello, World!

- First intro program.
- The program will print the text "Hello, World!" onto the screen.
- That's it.

#### hello_world.c

```c
#include <stdio.h>

int main(void)
{
  printf("Hello, World!\n");
  return 0;
}
```

- Let's break this down:

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

- This is a *preprocessor directive*. We'll talk more about the preprocessor later.
- For now, think of this as a step that happens before *compilation*
  - Compilation is the process of converting your C program into machine code)
- Here, we are using the `#include` directive to tell the preprocessor to get the `stdio.h` file and include in our program
  - "Include" in this sense means bring the source code of `stdio.h` into this project
  - This prevents us from having to re-invent the wheel - we can easily bring in existing code into our projects and use it.
  - For this example, we only need to `printf` function from `stdio.h`
  - Files that end in `.h` are called *header files*. We'll talk more about headers later.
- The `main` *function* is the entry point of all C programs. The computer needs to know where to start your program. `main` is the starting point.
- `printf("Hello, World!");` prints the string "Hello, World!" to the console
- `return 0;` terminates the program to let the operating system whether or not the program terminated successfully.
  - `0` indicates success in this case

## Preliminaries

### Comments

1. Single-line comments start with `//`
    - Example:

```c
// this is a single-line comment
```

2. Variable-line comments start with `/*` and end with `*/`
    - If you have a multi-line comment, each line beings with `*` after starting the comment

```c
/*
 * This is a multi-line comment
 */
```

## Block Structure, Statements, Whitespace, Scope

### Statement

- A **statement** is a command given to the computer that instructs the computer to take a specific action
    - Think of statements as the most atomic unit of our programs.
    - A program is made up of some sequence of statements
    - Statements terminate with the the semicolon `;` character
- An example statement: `int x = 1;`
    - This statement **delcares** a **variable** named `x` and **initializes** `x` to have the value `1`.
    - This value of `1` can be **accessed** or **modified** with the **identifier** `x`
- Understanding check:
    - What does it mean to declare a variable?
    - What does a variable store?
    - How can we access the value of a particular variable?

### Blocks

- **Blocks** consists of a group of executable **statements**
- Blocks begin with `{` and end with `}`
- Example:

```c
int main(void)
{
  /* this is a 'block' */
  int i = 5;
  {
    /* this is also a 'block', nested inside the outer block */
    int i = 6;
  }
  return 0;
}
```

### Whitespace

- Whitespace in a C program refers to tabs, spaces, and newline characters that separate text in the source code.
- Whitespace is ignored in many instances in C programs.
- The following are equivalent to a C compiler

```c
printf("Hello world"); return 0;
```

```c
printf("Hello world");
return 0;
```

```c
printf(
"Hello world");


return 0;
```

- When does the compiler not ignore whitespace?
  - Whitespace is important when using any **keyword** in C, such as `return`, `int` and others.

### Scope

- Two types of scope: **global** and **local**
- Global indicates something can be seen or manipulated from anywhere in the program
- Local indicates something can be seen or manipulated from anywhere in the program
- Example:

```c
int i = 5; /* this is a 'global' variable, it can be accessed from
    anywhere in the program */

```

```c
/* this is a function, all variables inside of it are "local" to the
   function. */
int main(void)
{
    int i = 6; /* 'i' now equals 6 */
    printf("%d\n", i); /* prints a '6' to the screen, instead of the
       global variable of 'i', which is 5 */

    return 0;
}
```

- What do we see from this example?
  - Local scope supersedes global
- A more complicated example:

```c
/* the main function */
int main(void)
{
    /* this is the beginning of a 'block', you read about those above
       */

    int i = 6; /* this is the first variable of this 'block', 'i' */

    {
        /* this is a new 'block', and because it's a different block,
           it has its own scope */

        /* this is also a variable called 'i', but in a different '
           block',
            because it's in a different 'block' than the old 'i', it
               doesn't affect the old one! */
        int i = 5;
        printf("%d\n", i); /* prints a '5' onto the screen */
    }
    /* now we're back into the old block */

    printf("%d\n", i); /* prints a '6' onto the screen */

    return 0;
}
```

## Basic Function Use

- We will take extensively about functions later in the course, but we need to have a basic introduction now - we need to know the basics to succeed in this course
- A **function** is a special kind of block that performs a well-defined task
- It enables programmers to perform a task without knowing how the function works
  - A form of *information hiding*
- When you **call** a function, you are telling the computer to execute the entire function code block in a single statement
  - The function invoking the function is called the **caller**
  - The function being called is called the **callee**
- Many functions require data as input
  - This data is passed to the function as **arguments**
- Many functions return a value to the caller
  - This is called a **return value**
- What you should know before calling a function:
  - What the function does
  - The data type of the arguments are and what they mean
  - The data type of the return value and what it means

## The Standard Library

- A collection of standard functions provided to you as the programmer to make programming easier, more secure, more robust, and more standardized
- `#include <stdio.h>` includes the standard library file `stdio.h` which stands for *standard IO*
  - We used this header to bring in the `printf` function, which is a part of IO