Pre-lab: Function Pointers
Week 15

Introduction to Void Pointers

In C the **void pointer** is called the General Purpose Pointer. It does not have any data type associated with it, and can be stored as address of any type of variable. A void pointer is a C convention for a raw address. The compiler has no idea what type of object a void Pointer really points to. Sample code for void pointers;

```
void *_ptr; // declaration of a void pointer

char c_var;
int i_var;
float f_var;

ptr = &c_var;  // ptr has address of character data
ptr = &i_var;  // ptr has address of integer data
ptr = &f_var;  // ptr has address of float data
```

The three variables above are of the data type character, integer and float, respectively. When we assign the address of integer to the void pointer, pointer will *become an Integer Pointe*r. When we assign the address of Character Data type to void pointer it will *become a Character Pointer*. Similarly we can assign the address of any data type to the void pointer. It is capable of storing address of any data type.

To de-reference this void pointer following syntax is to be used;

```
int _ptri   = *((int*)_ptr);
float _ptrf = *((float*)_ptr);
char _ptrc  =  *((char*)_ptr);
```

Introduction to Function Pointers

Following is a declaration of a function pointer that takes in two integer parameters and returns an integer;

```
int (*myFunction)(int, int);
```

Below is a function that takes in two integers and returns an int:

```
int addnum (int a, int b);
```

You can store the address of ***addnum*** to ***myFunction***, as follows:

```
myFunction  = &addnum;
```

And, then call **addnum** through the function pointer **myFunction** by explicitly dereferencing it using the * operator as shown below;

```
int result = *myFunction (2, 4);
```

You can also call the function through the function pointer without dereferencing as shown below:

```
int result = myFunction( 2,4);
```

Result will have 6 after the function call is made.