Introduction to C Programming
C Functions

- All C programming must be part of a C function.
- Example Declaration:

```c
void MyFunc(int a, int b)
{
    int c;

    c = a + b;
}
```
Your First Function

int main(int argc, char *argv[])
{
    ... Your Code Goes Here!
}

Enter the first line
Just Like This!

Braces Must Appear
On A Line By Themselves

Braces Must Line up
With One Another
Your First Program

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

int main(int argc, char *argv[])
{
    printf("Hello World\n");
    // Add a Line Here to print your name
    return 0;
}
```
Formal Syntax

- A function is declared as follows
- The `<Type>` is return value type and function characteristics

```
<Type>    <Function Name> ( <Argument List> )
{
    <Local Variable Declarations>
    <Executable Code>
}
```
## Types

- The most common types in C are the following:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>16-bit integer</td>
</tr>
<tr>
<td>long</td>
<td>32-bit integer</td>
</tr>
<tr>
<td>short</td>
<td>16-bit integer</td>
</tr>
<tr>
<td>char</td>
<td>8-bit integer or character</td>
</tr>
<tr>
<td>float</td>
<td>32-bit floating point</td>
</tr>
<tr>
<td>double</td>
<td>64-bit floating point</td>
</tr>
</tbody>
</table>
Type Declarations

- Type Declarations declare simple variables as well as pointers and arrays
- `int a;` -- defines `a` to be a 16-bit integer.
- `long b, c, d;` -- defines `b`, `c`, and `d` to be 32-bit integers.
- `char *xyz;` -- `xyz` is a pointer to a `char`.
- `int Totals[15];` -- `Totals` is an array of 15 `ints`. 
Function Headers

- The type `void` is used to indicate no return value, or no argument list.
- Example: `void Func1(void)`
- Each argument must have a declared type preceding its name
- Example: `int F2(int a, int b, char c)`
A function body consists of two parts:
- Declaration of Local Variables
- Executable code

Example:
```c
int F2(int a, int b)
{
    int c;

    c = a*a;
    c += b;
    return c;
}
```
Global Variables

- Arguments and Local Variables are accessible only inside the function where they are declared.
- Variable declarations that are placed outside of any function are accessible to all functions, and retain their values for the life of the program.
Globals: An Example

```c
int a;  // a global variable

void f1(int b, int c)
{
    int k;  // local k
    k = b*b;
    a = k + c;
}

// a different b and c
void f2(int b, int c)
{
    int k;  // a different k
    k = b + 2;
    // the same a as before
    a = k * c;
}
```
Assignments

- The equals sign is the assignment operator.
  - \( a = b + c; \)
- All common arithmetic operators, *except exponentiation*, can be used.
- Examples:
  - \( a=b+c; \quad a=b-c; \)
  - \( c=d*e; \quad c=d/e; \)
Assignment Statements

- Multiplication and Division Have Precedence over Addition and Subtraction.
- The Following Are the same
  - $a = b*c + e*d$;
  - $a = (b*c) + (e*d)$;
- Parentheses can be used to over-ride precedence.
New Operators

- % is used for remainders
  - This statement assigns 2 to a
  - \( a = 17 \% 3; \)

- & is used for bit-wise AND
  - This statement assigns 4 to a
  - \( a = 5 \& 6; \)
New Operators

- | is used for bit-wise OR
- ~ is used for bit-wise NOT
- ^ is used for bit-wise Exclusive-OR
- The expression $A << k$ can be used to shift $A$ to the left by $k$ bits.
- The operator $A >> k$ is used for right shift.
Short-Cut Operators

- $a=a+1$; can be replaced by $a++$;
- $b=b-1$; can be replaced by $b++$;
- DO NOT USE $++a$; or $--a$; even though they are available.
- DO NOT USE $a++$ in an expression, even though it is legal to do so.
- $a++$ and $b--$ must always appear on a line by themselves!
More Short-Cuts

- Any Binary Operator can be combined with the equals sign.
- A = A + 2; can be shortened to A += 2;
- B = B - 2; can be shortened to B -= 2;
- Also works for multiplication, division, remainder, bit-wise operations, and shifts
- A += (B * C) + (A * D); is legal.
Another Short-Cut

- All assignment expressions have a value
- $A = B = C = D = 1$, sets $A$, $B$, $C$, and $D$ to 1.
- **DO NOT DO STUFF LIKE**
  $A = B + C = D / E = Q*R$; Even though it is legal.
- Use multiple assigns ONLY to assign the same value to several variables.
Comparisons

The Comparison Operators are as follows

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>Equals</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>Not Equals</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>Less Than</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater Than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less than or equal</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater than or equal</td>
</tr>
</tbody>
</table>
WARNING!!!!!!!

- A = B is an assignment of B to A
- A == B is a comparison of B and A
- An assignment is legal any place where a comparison is legal!
- An assignment produces a TRUE/FALSE result and a comparison produces an arithmetic result. **BE CAREFUL!**
Comparison Results

- All Comparison Operators Produce a Numeric value: False produces zero, while True produces One.
- Complex Tests can be created using AND, OR and NOT operators.  (True is 1, False is 0)
  - && logical AND  (DO NOT USE SINGLE & )
  - || logical OR   (DO NOT USE SINGLE | )
  - ! logical NOT   (DO NOT USE ~ )
Boolean Values

- THERE AREN’T ANY!
- Integers (long, short) or characters are used instead.
- A zero value is considered false.
- ANY non-zero value is considered true.
- Formal comparison operators use 1 for true, 0 for false.
If Statements

- The format of the *if* statement is as follows.

```plaintext
if ( <Numeric Expression> )
{
    <True-Body>
}
else
{
    <False-Body>
}
```
If Evaluation

- If the numeric expression is zero, it is considered to be False, otherwise it is considered to be True.
- If the expression is True, the True-Body is executed, otherwise the False-Body is executed.
- The False-Body may be omitted, along with the *else* keyword and the enclosing braces.
While Statements

- The format of the *while* statement is as follows.

```plaintext
while (<Numeric Expression>)
{
    <While Body>
}
```
While Execution

- If the Numeric Expression is zero, it is considered to be False, otherwise it is considered to be True.
- The Loop-Body is executed until the Numeric Expression becomes False.
- The loop body will be skipped entirely if the expression is initially false.
For Loops

- In C, the *for* statement is used for most loops. The syntax is as follows.

```c
for (<Start-Body> ; <Condition> ; <Continue-Body>)
{
    <For-Body>
}
```
For Execution

- The C for statement is a special case of the while.
- The Start-Body is executed before the loop begins.
- The Condition is tested before executing the Loop-Body.
- The Continue-Body is executed after the Loop-Body.
More *For* Execution

- The loop-body continues to execute until the condition becomes false.
- If the condition is initially false, the Loop-Body will be skipped entirely.
- The Start-Body, and Continue-Body may consist of several statements separated by commas.
For Details

- Any part, Start-Body, Continue-Body, or Condition may be omitted. The semi-colons are required.
For Example 1

- Processing an Array

```c
for (i = 0 ; i < ArraySize ; i++)
{
    A[i] += 10;
}
```
For Example 2

- Processing a Singly-Linked List with Previous-Element Pointer

```c
for (Curr=Start, Prev=NULL ;
     Curr != NULL && Curr->Type != Red ;
     Prev=Curr, Curr=Curr->Next)
{
    Curr->Size += 3;
}
```
Break and Continue

Early termination of a loop is accomplished using the *break* and *continue* statements.

*Break* terminates the current loop immediately. The current-loop is the most deeply nested loop containing the *break* statement.

*Continue* is similar to *break*, but goes on to the next iteration of the loop.
Case Statements

- The Case statement is actually called Switch, and has the following format.

```java
switch (<numeric expression>)
{
    case <value-1>:
        {
        }
    break;
    case <value-2>:
        {
            ...
        }
}
```
Case Details

- The Numeric-Expression must be something that evaluates to an integer.
- `<value-1>, <value-2>, …` must be integer constants.
- Don’t forget the *break* statements, or you will be sorry.
Case Variations

- If you want to do the same thing for two different values, say 5, and 17, you can place case labels one after the other as follows.

```plaintext
case 5:  
case 17:  
{  
   <case-body>  
}
break;
```
The equivalent of the *else* keyword is the Case *default* label, which is used as follows.

```python
default:
{
   <Default-Body>
}
break;
```
Passing Data to Functions

- All arguments are passed by value.
- Arrays are passed by passing the address of the array to the function. Access is identical to accessing the array directly.
- Structures are copied and passed by value.
- All floats are converted to doubles, and converted back inside the function.
Passing by Reference

- Declare the function argument as a pointer to the desired type.
- When passing a variable, precede it by the & operator, which extracts the address of the variable.
- Reference the variable through the pointer.
- Use this to avoid copying massive structures to the argument stack.
Header Files

- If you have many declarations that are used in many different programs,
  - Place all declarations in one file with a .h extension, such as externs.h
  - Place the statement \#include "externs.h" at the beginning of each file

- Recall:
  - \#include <stdio.h> and \#include <stdlib.h>
Accessing Arrays

- Arrays are accessed as in other languages, but the first index is always zero.
- Square Brackets are Used for Array Indices.
  - \( A[3,4] \) is a reference to Array A
  - \( A(3,4) \) is a call to function A