**Lecture 3: Chapter 2: Introduction to C Programming – Monday Aug 28 or Tuesday Aug 29**

**Announcements**

Reading:

* Chapter 2 and Chapter 3

Assignments:

* Assignment #1 – due on **Wed Aug 30** (MW class) or **Thurs** **Aug 31** (TR)(no late assignments accepted)

Tutor Schedule:

Monday:

             9:30am – 12pm

Tuesday:

             12:30 – 4:30

Wednesday:

             9:30 – 12:00 pm

Thursday:

             12:30 – 4:30

Friday:

            None

**Today’s Goals**

1. Review programs to display information and receive information
2. Identifiers - Variables - Data Types – Assignment Statements
3. Memory Concepts
4. Arithmetic in C
5. Equality and Relational Operators
6. Reserved Words
7. Secure C programming practices

**Today’s Terminology**

**Terminology**

* Preprocessor Directive
  + Lines in a C program that are processed by the preprocessor **before** compilation
  + Any line that begins with a #
  + We will use these when including header files
* Function
  + Functions contain statements that perform a task
  + C programs contain a “special” function called “main
    - **int** **main**(**void**)
  + The “main” function is where a C program starts execution
  + Note, in Java we use the term **method** instead of **function**
* Blocks
  + Code that is between an opening curly brace and closing curly brace - { }
* Statements
  + Statements represent actions
  + Statements must end with a semicolon
* Header File
  + A file with the .h extension
  + Contains C declarations that can be shared with source files
* Source File
  + A file with the .c extension
* String
  + Also called a character string
* Reserved Words (Keywords)
  + Words that have a specific meaning in the language and cannot be used for other purposes
  + int, float, void, if, while, etc.
* Identifier
  + An identifier is **basically a name** chosen by the programmer for variables, constants, functions
  + Follows rules specified by the language
* Variable
  + Used to store a value that may change
  + Variables are given **names** based on the rules of the language
  + Variables have a **type** to indicate the type of values that can be stored in the variable
* Variable Definitions
  + A statement that defines a variable giving it a name and a data type
  + **int** numberOfStudents;
* Data type
  + Specifies the type of data stored in a variable such as integer
* Assignment Statement
  + Way to get a value into a variable
  + variable = expression;
* Arguments
  + The values that are specified when calling a function
  + Values being sent to a function
* Equality Operators
  + == Equal
  + != Not equal
* Relational operators
  + < Less than
  + <= Less than or equal
  + > Greater than
  + >= Greater than or equal
* Boolean expression
  + An expression that evaluates to true or false

**Review Simple C Programs – Input and Output**

**Output Example**

**#include** <stdio.h>

// All functions need to have a comment describing their purpose

**int** **main**(**void**)

{

// This is one way to create a comment in C

**printf** ("This is my first C programming class \n");

/\* This is another way to create a comment in C \*/

**printf** ("And this is my first C program");

}

**Program Output**

This is my first C programming class

And this is my first C program

**Input Example**

**#include** <stdio.h>

// All functions need to have a comment describing their purpose

**int** **main**(**void**) {

// There is a bug in Eclipse when you use scanf function

// The Eclipse console has buffer problems on Windows

// The problem is the printf statements are not executed until after the scanf

// Add the following line of code at the start of your program

**setvbuf** (stdout, NULL, \_IONBF, 0);

// Variable to store the user input

**int** numberOfStudents;

// Prompt the user for the number of students in the class

**printf** ("Enter the number of students in the class\n");

// Read the integer value the user enters

**scanf** ("%d", &numberOfStudents);

// Display the value the user entered

**printf** ("Can you believe %d students showed up for class.\n",

numberOfStudents);

}

**Program Output**

Enter the number of students in the class

32

Can you believe 32 students showed up for class.

**Identifiers - Variables - Data Types – Assignment Statements**

**Identifier**

* An identifier is **basically a name** chosen by the programmer for:
  + Variables
  + Constants
  + Structures
  + Functions
* Identifiers follow certain rules
  + Consists of letters, digits, underscores (\_)
  + Must start with a letter or an underscore – so cannot start with a digit
  + Can be any length
  + Cannot be a **reserved word – (**int, float, void, if, while, etc. – see Fig 2.15)
* Valid
  + sum
  + numberStudents
  + \_thisWorksToo
    - Note: starting identifiers with underscores should be avoided
    - Avoids confusion with complier generated identifiers and library identifiers (they start with underscores)
* Invalid
  + 7\_students
  + #OfStudents
* Notes:
  + C is case sensitive so sum, Sum, and SUM are 3 **different** identifiers
  + Complier will mark invalid identifiers
  + Keep the following in mind with identifiers
    - Use meaningful names
    - With multiple-word identifiers use **camel case**
      * First word starts with lower case letter
      * Each remaining word starts with upper case

**Variable**

* Used to store a value that may change
  + **numberStudents**
  + **dayOfWeek**
  + **miles**
* A variable’s value is stored in the computer’s memory
* Variables have scope
  + Scope defines the part of the program where is variable is accessible
* Variables follow certain rules
  + Must have a name – in this class that must be a meaningful name!
  + Must have a data type
* What we do with variables
  + Declare
  + Put values in
  + Manipulate – modify their value, display their value, use them in formulas

**Data Type**

* Tells compiler **what type** of data that is stored in a variable
* You must **declare** the type of each variable
  + This means assign a specific type to the variable – it’s an integer
* For starters, we are looking at these data types:
  + integer – int
  + floating – float
  + character - char
* Strongly typed language!
  + This means that once you declare a variable to be a certain it will behave as that type
  + Type safety- you can’t put a floating-point value into an integer unless you **explicitly** tell it

**Assignment Statements**

* Way to get a value into a variable
* Syntax for an assignment statement

variable = expression;

* Example:

totalNumberEmployees = fullTimeEmployees + partTimeEmployees + studentEmployees;

* Expression on right hand side is evaluated and *assigned* to variable on left hand side

**Memory Concepts**

**Variables and Computer Memory**

* When you **define** a variable, you are indicating several things:

**int** numberOfStudents;

**Type** of value to **Name** of the

store in memory memory location

location

**Computer Memory**

When variables are defined, a memory location is created for them

**int** numberOfStudents;

**printf** ("Enter the number of students in the class\n");

**scanf** ("%d", &numberOfStudents);

|  |  |
| --- | --- |
| numberOfStudents |  |

When user enters a value – say 32 – the value is placed into the memory location of *numberOfStudents*

Enter the number of students in the class

32

|  |  |
| --- | --- |
| numberOfStudents | 32 |

**Arithmetic in C**

**C Operators**

|  |  |
| --- | --- |
| Addition | + |
| Subtraction | - |
| Multiplication | \* |
| Division | / |
| Remainder | % |

**Notes**

* Division with integers
  + Result is an integer
  + Fraction part is truncated
  + Division examples:
    - ½ is 0
    - 10/6 is 1
  + Reminder examples
    - 1%2 is 1
    - 10%6 is 4
* Remainder operator is useful to determine if a number is odd or even
  + even number % 2 = 0
  + odd number % 2 = 1

**Evaluating Expressions**

* Expressions are evaluated in C just as they are in math
* Recall from math the **order of operators**:
  + Anything in parentheses
  + \* / % (multiplication, division, remainder)
  + + - (addition, subtraction)
  + = (assignment)
* When you have operators with similar precedence what happens?
  + They are applied in order from left to right

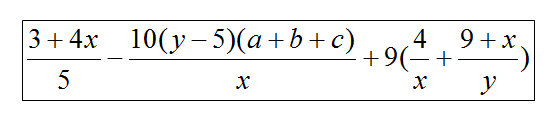
**Evaluating Expression Example**

* Example #1

8 + 16 \* 4 / 2 – 2 \* (3 + 17)

Apply order of operators

* + (3 + 17) = 20 8 + 16 \* 4 / 2 – 2 \* 20
  + 16 \* 4 = 64 8 + 64 / 2 – 2 \*20
  + 64 / 2 = 32 8 + 32 – 2 \* 20
  + 2 \* 20 = 40 8 + 32 – 40
  + 2 + 32 = 34 40 – 40
  + 0
* Example #2



* Figure out the different pieces first

(3 + (4 \*x)) / 5

10 \* (y – 5) \* (a + b + c) / x

9 \* (4 / x + (9 + x) / y)

* Then put together and add parentheses to make it easier to read and understand for humans!

((3 + (4 \*x)) / 5) – (10 \* (y – 5) \* (a + b + c) / x) + (9 \* (4 / x + (9 + x) / y))

**Equality and Relational Operators**

**They talk about if-statement here but I will cover if-statements in next lecture on chapter 3**

**Actions Statements**

* So far, we have seen statements that perform actions
* printf – display information
* scanf – read input

**Decision Statements**

* If statements
* Switch statement

**Purpose**

* Provides ability to select a path of execution **based on a condition**
* “If statements” implement conditional execution - alters flow of control
  + If it is snowing today, then I'm not going to class

**Conditions**

* Create a condition using equality and relational operators
* Conditions evaluate to true if the condition is non-zero
* Conditions evaluate to false if the condition is zero

**Equality Operators**

* == Equal to (yes, that's two equal signs)
* != Not equal to

**Relational Operators**

* < Less than
* <= Less than or equal to
* > Greater than
* >= Greater than or equal to

**Fitting into Order of Operators**

* Anything in parentheses
* \* / % (multiplication, division, remainder)
* + - (addition, subtraction)
* **< <= > >= (relational operators)**
* **== != (equality)**

**Notes**

* Relational operators are used to compare values
* Be aware of the difference between the assignment operator (=) and the equality operator (==)

**Example**

int mathGrade = 82; // This is assignment

if (mathGrade == 3);//This is equality - displays **true**

**Reserved Word s**

Reserved Words (Keywords)

* Words that have a specific meaning in the language and **cannot be used** for other purposes



**Secure C Programming**

**Secure Programming**

* To write code that uses techniques that can stand up to attacks
* This topic is an entire class so we won’t be focusing on this topic
* We will only discuss some of the techniques

**CERT C Secure Coding Standard**

* CERT – Computer Emergency Response Team - [www.cert.org](http://www.cert.org)
* Publishes and promotes secure coding standards
* Standard for C
  + <https://www.securecoding.cert.org/confluence/display/c/SEI+CERT+C+Coding+Standard>
* Standard for other lanaguages:
  + <https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards>

**Practice to avoid leaving systems open to attacks - printf**

* Avoid using **printf** with a single string argument. Use the **puts** function instead, it adds in the newline automatically.

**printf** ("This is my first C programming class \n");

rewrite as

**puts** ("This is my first C programming class");

* If you want to display a string without a newline:

**printf** ("Enter the number of students in the class");

rewrite as

**printf** ("%s", "Enter the number of students in the class");

**Class Time for Exercises**

Any questions about getting your environment setup or the first programming assignment?