**Lecture 7 - Chapter 4: C Program Control – Wed Sept 13 or Thurs Sept 14**

**Announcements**

Reading:

* Chapter 4

Assignments:

* Graded: Assignment #2 graded
* Due: Assignment #3
* Assign: Assignment #4 (While and For Loops)- due on **Sept 20** (MW class) or **Sept 21** (TR class)
* No late assignments accepted

Exam:

* Exam #1 on Sept 20th (MW - class) or Sept 21th (TR - class)

**Today’s Goals**

1. Assignment #2 Feedback
2. Do-While Loop
3. Loop Design
4. Break and continue
5. Logic Operators
6. Secure C Programming
7. Exam #1 Review (Hand out review sheet today – might not have time to cover in lecture)

**Assignment #2 Feedback**

1. Reading instructions
   1. Not reading specifications
   2. Did not handle backorder situation
2. Comments
   1. Some people didn’t have any comments in their code
   2. Add comments to sections to explain what that section does
   3. For description of the program don’t leave default description in place – your program is not the “hello world” program
3. Use of constants
   1. Lots of people had the numbers for the book prices and tax all over the place!
   2. Create constants for values like this.
   3. What would happen if the price of the math book changed? You or the person updating your code would have to track down all those $37.50 numbers.
   4. Define constants either as
      1. #define MATH\_COST 37.50
      2. const int MATH\_COST = 37.50;
4. Indentation and spacing
   1. Watch indentation when dealing with if-statements
   2. Some code could use better use of space - white space helps readability
5. Using if-statements properly
   1. Many people wrote code like this:

If (menu == 1) {

}

If (menu == 2) {

}

If (menu == 3) {

}

This should be written

If (menu == 1) {

}

else if (menu == 2) {

}

else if (menu == 3) {

}

The 1st approach is less efficient. For example, if menu option is 1, it will execute body of 1st if statement but then it will test the other cases as well when there is no need to.

With the 2nd approach, once it is done with the first if-statement, it’s done!

1. Redundant code
   1. Biggest issue!
   2. Many people repeated code over and over.
   3. This is a very bad practice! You are creating a maintenance nightmare!
   4. Pull the common bits out. Use if statements and variables to figure out what should be printed.
   5. I will post my solution on blackboard. Please review if you had this issue.
   6. You were not marked off for this but will be on future assignments.

**Today’s Terminology**

**Terminology**

* Iteration - Looping
  + Way to repeatedly execute code
  + While loops
  + Do-while loops
  + For loops
* Counter-controlled Iteration
  + Loop for a known number of iterations
  + Also called *definite iteration*
* Sentinel-controlled Iteration
  + Loop until a certain value is reached
  + Also called *indefinite iteration*
* Loop Body
  + Statements that are repeated within the loop
* Loop Iteration
  + One complete execution of the loop body
* Infinite Loop
  + A loop that runs forever - stuck executing the body over and over because the condition tested never becomes false
* Sentinel Value
  + A value that signifies the end of a loop
* Pretest Loop
  + Loops where the continuation condition is checked **before** the loop body is executed
* Posttest Loop
  + Loops where the continuation condition is checked **after** the loop body is executed
* Coding Incrementally
  + A problem-solving approach
  + Write one part of the code - get it working and tested - then add a little more
  + Helpful with loops
* Logical operators
  + != Not
  + && And
  + || Or
  + ^ Exclusive Or (Not in C – but can do with != or the bitwise ^ operator)
* Boolean data type
  + Boolean is used to store the values - **true** or **false**

**Do…While Loop**

**Purpose**

* Same purpose as a while loop => a programming structure that performs repeated execution (looping)

**General Form**

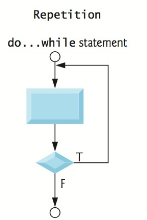
* A do-while loop has the following general form:

do {

statements(s);

} while (boolean expression);

* Flowchart:



**How It Works**

* The loop body is executed
* The loop condition - boolean expression - is evaluated
* If the loop condition is true, then loop body is executed again
* If the loop condition is false, control is transferred to the 1st statement following the loop

**Rules for Do While Loops**

* These are same as while loop
* The loop condition must be a boolean expression
  + Boolean expression must be in parentheses
  + Boolean expression is formed using relational and logical operators
  + Generally, some statement before the while loop "initializes" the loop condition to true
* Some statement within the loop body must eventually change the condition to false
* If the condition is never changed to false, the program will be forever stuck in the loop
  + This is called an "infinite loop"
* Curly braces are not necessary if only one statement in loop but best practice is to always include curly braces

**Example**

**int** counter = 1; // Initializes the loop control

**do** {

counter++; // Changes Loop control variable

**printf** ("The value of count is %d\n", counter);

} **while** (counter <= 5);

The value of count is 2

The value of count is 3

The value of count is 4

The value of count is 5

The value of count is 6

**Notes**

* Recall how placing a semicolon at the end of the while-clause creates an infinite loop!

**int** iteration = 1;

**while** (iteration <= 10)**;** { // Unnecessary semicolon

**printf** ("Iteration = %d", iteration);

iteration = iteration + 1;

}

* In the case of do-while you must include the semicolon since it ends the loop!

**int** iteration = 1;

**do** {

iteration++;

**printf** ("Iteration = %d", iteration);

} **while** (iteration <= 5); // Necessary semicolon

**Which Iteration Statement Should Be Used**

**Review**

* While loop
  + Loops while some condition is true
  + The body of a while-loop may or may not be executed - loop can be skipped
  + Does not have built in initialization or iteration
  + Useful when unsure how many times a loop will execute (indefinite loop)
* Do-while
  + Loops while some condition is true
  + The body of a do-while loop always executes once
  + Does not have built in initialization or iteration
  + Useful when unsure how many times a loop will execute (indefinite loop)
* For loop
  + Loops a preset number of times
  + Comes with built in initialization and iteration
  + Useful when certain how many times a loop will execute (definite loop)

**Which One**

* The 3 loops are equivalent
  + Generally, while loop can be converted into a for-loop and a for-loop can be converted into while
* Use a While-loop if
  + You **don't** know how many times the loop will execute
* Use a For-loop if
  + You **know explicitly** how many times the loop will be executed
  + Loop will not terminate abnormally or before we have completed specified number of iteration

**Loop Design**

**Strategies for Writing Loops**

* 1st - figure out what part of your code needs to be repeated
* 2nd - place that part of you code into a loop
* 3rd - determine what condition the loop checks
* 4th - make sure you have code in the body that eventually causes the condition to become false

**Break and Continue**

**Break in Loops**

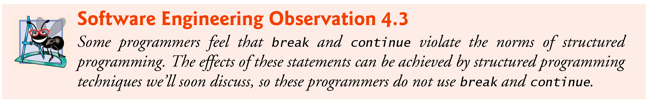
* Used "break" in switch statements to end a case
* Can be used in a loop to terminate a loop
* Breaks out of loop
* **Don't use breaks in a loop in this class!**

**Continue in Loops**

* Used to end current iteration of loop
* Program control goes to end of loop body
* Breaks out of current iteration
* **Don't use continue in a loop in this class!**

**Note**

* There are cases when the use of a **break** or **continue** in a loop is necessary – (i.e. performance situations)
* Overusing or improper use can make programs hard to read, maintain, and debug.
* Best practice is to never use these unless you have a specific situation
* **IN THIS CLASS,** 
  + **WE WILL NOT HAVE A SITUATION THAT REQUIRES THE USE OF A BREAK OR CONTINUE IN WHILE OR FOR LOOPS**
  + **THE GOAL IS TO 1st LEARN PROPER STRUCTURED PROGRAMMING**
  + **Read the 3 tips on page 132.**



**Boolean Data Type**

**Data Type**

* Tells the compiler **what type of data** is stored in a variable
* For integer variables, we have
  + short, int, long
* For floating-point variables, we have
  + float, double

**Boolean Data Type**

* The C Standard
  + includes a boolean type represented by the keyword **\_Bool**
  + **true** is represented by non-zero values
  + **false** is represented by zero

\_Bool snowing = 1;

\_Bool sunny = 0;

* + also includes <stdbool.h> which defines the following replacements
    - bool --- \_Bool
    - true --- non-zero values
    - false --- zero Preferred approach!

bool snowing = true;

bool sunny = false;

**Notes**

* In Java, you cannot cast from **ints** to **boolean** or **boolean** to **int**
* **In C, you can do stuff like this:**

bool snowing = true;

**if** (snowing) {

**int** someIntegerValue = snowing;

**printf** ("someIntegerValue = %d\n", someIntegerValue);

}

**Displays**

someIntegerValue = 1

* Some implementations my not have \_Bool -> book mentions Visual Studio did not implement \_Bool

**Logic Operators**

**Simple and Complex Conditions**

* We have learned how to create simple conditions
  + Using relational and equality operators

if (numberStudents <= 10)

* Want ability to create complex condition statements
  + Use logical operators to combine conditions into complex statements

**Logical Operators (also called Boolean Operators)**

* ! not
* && and
* || or

**Fitting into Order of Operators**

* Anything in parentheses
* expr++ expr-- (postfix)
* ++expr --expr (prefix)
* **! (not)**
* \* / % (multiplication, division, remainder)
* + - (addition, subtraction)
* < <= > >= (relational operators)
* == != (equality)
* **&& (and)**
* **|| (or)**
* = += -= \*= /= %= (assignment, augmented assignment)

**Notes**

* Logical operators are used to compare values and create compound boolean expressions
* C evaluates all expression that include relational operators, equality operators, and/or logical operators to 0 or 1

**Truth Tables**

* **Not operator (!)**
  + Reverses the logical value of the expression

|  |  |
| --- | --- |
| **Expr1** | **!Expr1** |
| true | false |
| false | true |

bool snowing = true;

if (!snowing) {

puts ("Go to class");

}

* **AND operator (&&)**
  + True if and only if both boolean operands are true, otherwise false

|  |  |  |
| --- | --- | --- |
| **Expr1** | **Expr2** | **Expr1 && Expr2** |
| False | false | false |
| False | true | false |
| True | false | false |
| True | true | true |

bool snowing = true;

bool icyRoads = true;

if (snowing && icyRoads) {

puts ("Stay home");

}

int mathGrade = 82;

if ((mathGrade >= 80) && (mathGrade < 90)) {

puts ("You got an B");

}

* **OR operator (||)**
  + True if at least one of the boolean operands is true, otherwise false

|  |  |  |
| --- | --- | --- |
| **Expr1** | **Expr2** | **Expr1 || Expr3** |
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | true |

int height = 72;

int weight = 190;

if ((height > 65) || (weight < 140)) {

puts ("Move to next step");

}

**Short Circuit Evaluation**

* Means that we stop evaluating a **boolean expression** as soon as its value can be determined
* This happens when have a compound expression involving && and ||
* && - stop once an expression that evaluated to false is found
* || - stops once an expression that evaluates to true is found

Review

* Integers
* Data Type: short, int, long
* Operators: \*, /, %, +, -, equality, relational
* Floating Point Data Type
* Data Type: float, double
* Operators: \*, /, %, +, -, equality, relational
* Booleans
* Data Type: bool
* Values: true and false
* Operators: equality, relational, logical

**Confusing Equality and Assignment**

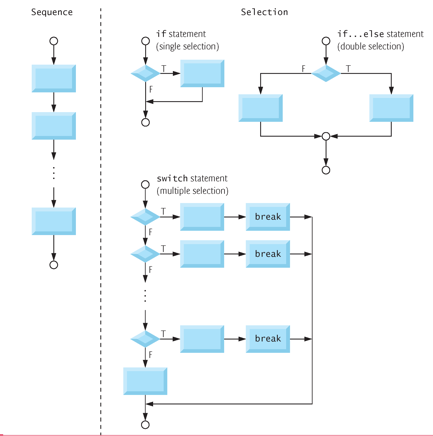
**Common Error**

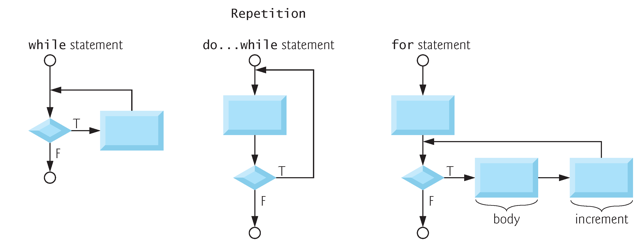
* Using = when == is expected!
* Very hard to spot!
* Leads to logical errors!

**Structured Programming Summary**

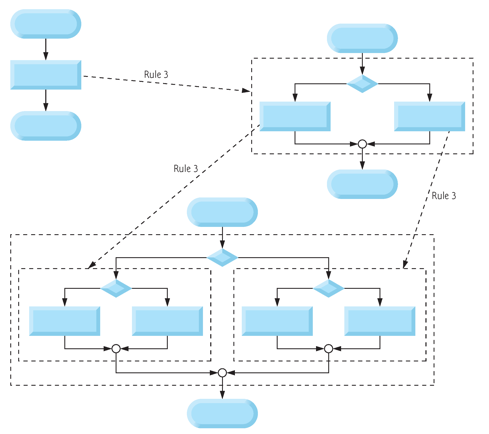
**Structured Programming**

* Basically, a set of rules to produces programs that are easier to understand, debug, and modify.
* Structures using in programming
  + if statement
  + switch statements
  + while statements
  + do…while statement
  + for statement
* Flowcharts for these structures





* We build structured programs by combining these structures
* Rules for **properly** combining these structures
  + **Exit point** of one structure may be connected to the **entry poin**t of another
  + Any rectangle can be replaced by two rectangles in sequence – stacking rule
  + Any rectangle can be replaced by any control statement – nesting rule



**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 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>

**Checking Function scanf’s Return Value**

* scanf returns an integer indicating if operation was successful
  + If it fails, it returns the value EOF
  + Otherwise, it returns number of items that were read
* You can make your input processing more robust by
  + Checking scanf’s return value and ensuring the number of inputs read match number of inputs expected!
  + This will help prevent errors, program crashes and attacks.

**Range Checking**

* To make programs more robust, good practice to ensure the values a user enters is in expected range

**Review for Exam #1**

We’ve covered a lot of terminology and I will **NOT** be asking you to define any terms **BUT** you need to know the meaning of the terminology we’ve covered.

**Chapter 1 - Introduction to Computers, the Internet and the Web**

* Questions about programming style and documentation
  + Based off homework grade sheet
  + What are meaningful names, proper indentation, camel case, constants
* Difference between compiler, logical and runtime errors
  + What causes each type of error?
  + When or how they occur

**Chapter 2 – Introduction to C**

* Know how to print output to console and obtain input from keyboard
  + printf and scanf
* Know the different data types and what a "data type" tells the compiler
  + int, float, double, char
* Know when and how to declare variables, assign values to, use in statements
* Know when and how to declare constants, assign values to, use in statements
* Be able to use numeric operations (+, - , \*, /, %)
  + Integer division and the issues you can have with integer division
* Know how to evaluate numeric expressions
* Understand operator precedence when it comes to evaluating numeric expressions
* Know what the **relational operators** and **equality operators** are
* Know how to use relational operators to create boolean expressions
* Know how to evaluate boolean expressions

**Chapter 3 – Structured Program Development in C**

* Understand these different types of **if statements**
  + One way if statement
  + Two way if statement
  + Nested if statement
* Understand how a **while loop** works
* Simple type conversions - explicit casting – we did this with integer division example
* Be able to use the increment and decrement operators
* Be able to understand code and/or write code snippets with **if statements**
* Be able to understand code and/or write code snippets with **while loops**

**Chapter 4 – C Program Control**

* Understand the difference between counter controlled loops and sentinel controlled loops
* Understand how a ***for loop*** works
* Understand how a ***do-while loop*** works
* Know when to use one loop structure over the other
* Understand what causes infinite loops
* Understand how a **switch statement** works
  + Understand how fall-through behavior works
  + Understand how the default case works
* Know the difference between an **if statement** and a **switch statement**
* Understand char data types
  + What ASCII code is and how it is used at a high level
* Know how to use logical operators together with the relational/equality operators to create boolean expressions
  + Know the truth tables
* Know the operator precedence for all the operators
* Be able to understand code and/or write code snippets with **for loops** and **do-while loops**
* Be able to understand code and/or write code snippets with **switch statements**