Introduction to Software Development – CS 6010

Lecture 3 – Control Flow

Master of Software Development (MSD) Program

Varun Shankar

Fall 2022

Miscellaneous

- Starting today you need to name your labs/homework exactly as specified in the assignment. They should be placed in a “Day#” folder. So for today, you should create “Day3” inside your repo, and place all labs/HW in that sub-folder. Note “Day 3” and “Day_3” and “day3” are all NOT “Day3”!
- Are you using Slack to speak with the TAs / other students?
- Any questions?

Are these the same:

- 3.1415926
- 3.14159265358979323846264338327 9

- To a computer they are… sort of…
  - *at the standard level of precision (float)*
- float (aka single) vs. double precision float
  - 32 bits vs 64 bits
  - ~7 vs ~16 digits of precision

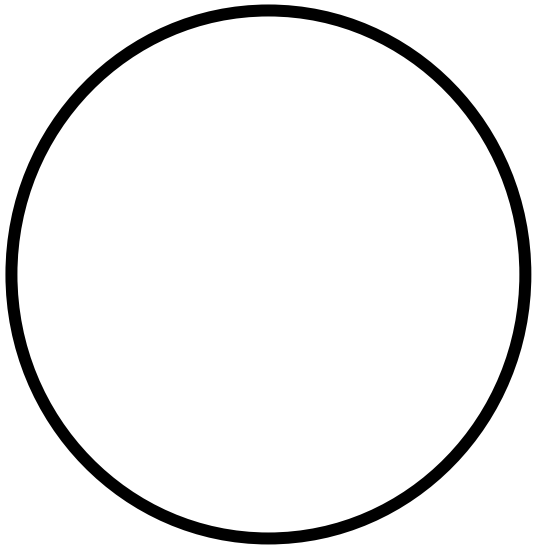
What about this?

pi

- This is a *Constant*. If the programming language has one, use it!
  - #include <cmath> // C++ library that defines pi for us.
  - M_PI // The pi constant.
- Is it the same as just writing pi out yourself?
  - It's better as we don't have to remember the exact value. Really, who wants to remember the digits of pi? ☺
- A *Constant* is a symbolic name for a value stored in the computer that does not change. We use it in a program just like a variable, but CANNOT change its value.
  - const double pi = 3.1415926535897932;

Computers Approximate

- Computers Approximate the World
  - Computer programs represent *approximations* of the reality they are modeling.
  - Variables are used to store the information we use to *approximate* the world.
  - What information (data / variables) would we need to create this approximation of a circle?



A better approximation, but in this case, using the same data…

‘If’ statement (Branching)

- Allows computer to make a decision based on its current state (i.e., its data / variables)
- This is the only way to ask a question!
  - The question can only be a “yes” or “no” question!
- Example: (Notice the indentation!)

```
if( age >= 18 )
    printf( ‘Stay out late.\n’ ); // printf is oldschool cout
```

If, If-Else

```
// Basic 'if' statement (Only want to do something if a condition is true):
if( condition ) {
     A - lines of code to execute if condition is true
}

// Use an 'else' section if you need to do something different when the
condition is false.
// With 'else, the computer will do one thing if true, something else if
false...
if( condition ) {
   A - things to do if condition is true
}
else {
   B - things to do if condition is false
}
// What is the 'type' of 'condition'?
```

if / else if / else

```
if( condition ) {
    A - things to do if condition is true
}
else if( another_condition ) {
   B – things to do if condition is false and another_condition is true
}
else {
   C - things to do if both 1st and 2nd conditions are false
}
// If condition and another_condition are true, will B happen?  How about C?
```

Conditionals

- A "Conditional" is a statement that evaluates to either `true` or `false` – a boolean.
  - You can think of it as a true/false question
- For example

```
( 5 <= 10 )    // Written as it is said, ie: 5 is less than or equal to 10
( 10 < 5 )
( x < y )
( x == y )     // Equality
( x != y )     // Not equal
( length(array) > 100 )  // Note: length() is not a built in function.
```

- What *type* is the result of each of the above?
  - Generically the results are all Booleans.

Conditionals – more complex questions

- This is a single *question* with 2 parts:
  - (x < y && x < z)
- Note: && means AND
  - With AND, BOTH sides most be true for the entire statement to be true.
  - Used in order to have “two parts” to one question.
- (x < y || x < z)
- || (double pipe signs) means OR
  - ONE OR THE OTHER side (or both) must be true for the entire *question* to be true.

Truth Tables (And, Or) [What is TT for not?]

AND: B1 && B2	B1	B2
True	True	True
False	True	False
False	False	True
False	False	False

```
B1 = false;
B2 = true;
result = B1 && B2;
// What value does result have?

result = B1 || B2;
// Now what is the value of result?
```

OR: B1 || B2	B1	B2
True	True	True
True	True	False
True	False	True
False	False	False

*Two variables (in this case B1 and B2) -> four possible permutations

Examples of Conditionals

- if(!time_to_go) // What *type* is time_to_go?
  - What is the "!" (exclamation mark)?
    - Means "not"
  - We could also write the above as:
- if(time_to_go == false) // or
- if(time_to_go != true) // all 3 are equivalent
- if(has_ticket == true)
- if(has_ticket) // same as above
- if(age > 16 && has_ticket)
- if(age >= 18 || with_parent)
- if(strcmp(answer, 'yes') == 0)

  // To compare two strings (*actually, character arrays), you cannot use ==, you must use the strcmp()

  // (string compare) function.

DeMorgan's Law (Boolean Logic)

- **Not (A OR B) == (Not A) AND (Not B)**

  !(A || B) == !A && !B

  !(A && B)

  !A || !B

  !(!A && !B)

  A || B

- !(num < 10) // This and the next example are not DeMorgan's Law

  num >= 10 // But are similar and important to know.

- !(num == 10)

  num ~= 10

Programming Example

- In class example of determining a student’s letter grade.

Reminder: Use {}

```
if( grade > 90 )
        cout << “A”
        cout << “Well Done!”
```

- Which of the above lines are *inside (belong to)* the *if* statement?

```
if( grade > 90 ) { // What is (might be) the logic error with this line?
        cout << “A”;
        cout << “Well Done!”;
}  // Because of the {} both lines now belong to the if statement.
```

Looping

- A loop repeatedly executes code over and over again until some condition is met.
  - At that point, the loop *terminates*
- There are two types of loops
  - `for` Loop (usually repeat a “known” number of times)
  - `while` Loop (usually repeat an “unknown” number of times)

While Loop – Design Pattern

- Syntax: Memorize this!

```
while ( condition is true ) {
    do these
    lines of
    code
}
```

While Loop In Action

Example: (What does this code do?)

```
int age = 1;
int doubles = 0;          // Assume age is set to 50.
while( age < 100 ){
    doubles = doubles + 1;
    age = age * 2;
}
```

At the “end” of a loop, you ALWAYS

1) go back to the top of the loop
2) then check the condition to determine if you repeat the loop, or drop out

While Loop

```
// Remember: One line at a time!
// And only one line.
```

```
// Executing code by hand to
// understand what it does:
 age = 1;
 doubles = 0;
 while ( age < 100 ) {
    doubles = doubles + 1;
    age = age * 2;
 }
 printf( "result is...
```

While Loop

- Example:

```
age = 1;
```

When debugging or trying to understand what codes does as you execute it by hand, don’t look at the rest of the code, only look at the current line that is being executed!

While Loop

- Example:

```
age = 1;
doubles = 0;
while ( age < 100 ) {
    doubles = doubles + 1;
    age = age * 2;
}
printf( ‘result is...
```

While Loop

- Example:

```
doubles = 0;
```

While Loop

- Example:

// Think of *while* as a
// repeating *if* statement.

```
age = 1;
doubles = 0;
while ( age < 100 ) {
    doubles = doubles + 1;
    age = age * 2;
}
printf( ‘result is...
```

While Loop

- Example:

```
while ( age < 100 ) { // What is the
 value of age?
```

While Loop

- Example:

```
age = 1;
doubles = 0;
while ( age < 100 ) {
    doubles = doubles + 1;
    age = age * 2;
}
printf( ‘result is...
```

While Loop

- Example:

```
age = 1;
doubles = 0;
while ( age < 100 ) {
    doubles = doubles + 1;
    age = age * 2;
}
printf( ‘result is...
```

While Loop

- Example:

```
age = 1;
doubles = 0;
while ( age < 100 ) {
    doubles = doubles + 1;
    age = age * 2;
}
printf( ‘result is...
```

While Loop

- Example:

```
age = 1;
doubles = 0;
while ( age < 100 ) {
    doubles = doubles + 1;
    age = age * 2;
}
printf( ‘result is...
```

While Loop

- Example:

```
age = 1;
doubles = 0;
while ( age < 100 ) {
    doubles = doubles + 1;
    age = age * 2;
}
printf( ‘result is...
```

While Loop

- Example:

```
age = 1;
doubles = 0;
while ( age < 100 ) {
    doubles = doubles + 1;
    age = age * 2;
}
printf( ‘result is...
```

While Loop

- Example:

// What is the next line of code that will be looked at?

```
age = 1;
doubles = 0;
while ( age < 100 ) {
    doubles = doubles + 1;
    age = age * 2;
}
printf( ‘result is...
```

While Loop

- Example:

// Assume ‘age’ is 128 now.
// What is the next line of code?

```
age = 1;
doubles = 0;
while ( age < 100 ) {
    doubles = doubles + 1;
    age = age * 2;
}
printf( ‘result is...
```

While Loop

- Example:

```
age = 1;
doubles = 0;
while ( age < 100 ) {
    doubles = doubles + 1;
    age = age * 2;
}
printf( ‘result is...
```

While – UNKNOWN repetitions

- Use a while loop when you don’t know how many times something will happen.
- Allows the computer to “keep going” until a condition becomes false, for example:

```
while( grade < 90 ){
    // repeat quiz
}
```

For Loops

- For loops are typically used when we “know” how many times we want the computer to repeat.
  - For example:
    - A thousand times
    - Once for every element (piece of data) in an array

For Loop – Design Pattern

- Syntax: Memorize this!

```
for( int i = start; i < finish; i++ ) { // initialize, test, increment
    do something
}
```

For Loop – Example

```
for( int x = 0; x < 10; x = x + 2 ) {
        cout << “x is: “ << x << “ and twice x is: “ << x * 2 << “\n”;
}

// How many times does this loop execute?
// 5 times
```

Today’s Assignment(s)

- Lab – If Statements
  - ~10:40 – ~11:10 AM
- Lab – Loops
  - ~11.10 AM - ~Noon