Discussion Session Week 3

Conditionals, Loops, Tips and Tricks

Before we get started

Your first homework will likely be released soon, here are some tips

- **Start early**
  - You will typically be given a week for each assignment, use it
  - Late work is not accepted in this class
- Read the assignment a few times before starting
  - “Weeks of coding can save you an hour of planning”
  - Seriously, don’t just try to code without understanding what the assignment does
- Utilize your resources
  - Come to office hours, read the slides, look online

Context

- Conditionals and loops are what makes up a bulk of programming
- A clever use of them can solve just about any problem you encounter
- Mastering these skills is essential for you to become successful as a programmer
- Assignments will require them

The basics

Different formatting

Sometimes we can chain statements to get them formatted the way we want them to be

There are a few ways to achieve the same outcome (as with most programming principles)

```
int input;

std::cout << "enter a number: ";
std::cin >> input;
```

```
~/CSC 211/discussions/basicsPlus/ $ g++ format.cpp -o format && ./format
enter a number: 12
Even
```

If / else

- Works the same as it does in Python
- The only real difference is that instead of ‘elif’ we use ‘else if’
- Conditions *always* need to be in parentheses and followed by a set of braces

```
if (condition1) {
  // block of code to be executed if condition1 is true
}

else if (condition2) {
  // block of code to be executed if condition1 is false and condition2 is true
}

else {
  // catch-all for any other case
}
```

Switch statements

- Think of these as a really condensed set of if/else statements
- Like a set of if/else statements, switch statements only evaluate once
- Optional, but important (use them for now)
  - *break* will get you out of the statement when it's done. If there is no *break* statement, it will also run all of the cases below the triggered case.
  - *default* acts the same as your catch-all 'else' statement
- We can only use *chars* or *ints* as our expression
  - This has to do with internal representation of the expression as well as the cases needing a constant value
  - I encourage you to look into this!

```
switch(expression) {
  case 1:
    // do something if case 1 is true
    break;

  case 2:
    // do something if case 2 is true
    break;

  default:
    // catch-all for anything else
}
```

For loops

We've already gone over for loops, but here's another example

```cpp
#include <iostream>

int main(int argc, char* argv[]){

    for(int i = 5 ; i < 10; i++){

        std::cout << i << std::endl;

    }
}
```

```
~/CSC 211/discussions/ $ g++ for.cpp -o for && ./for
5
6
7
8
9
```

While loops

Again, it's very similar to how it works in Python

- Variable declaration, followed by the loop with a boolean expression that evaluates to true. This process continues until the condition is false.

```cpp
#include <iostream>

int main(int argc, char*argv[]){

    int i = 0;

    while(i < 5){

        std::cout << i << std::endl;
        i++;


    }
}
```

```
~/CSC 211/discussions/ $ g++ while.cpp -o while && ./while
0
1
2
3
4
```

do/while loops

This is a variation of a while loop in which the body of the loop will be executed at least once

Rather than executing the body of the loop only if the condition is true, the body is executed prior to the condition being checked

- If the condition is true, the body will execute again like a normal while loop

```cpp
#include <iostream>

int main(int argc, char* argv[]){

    int i = 6;

    do{

        std::cout << i << std::endl;
        i++;
    }
    while(i < 5);
}
```

```
~/CSC 211/discussions/ $ g++ dowhile.cpp -o dowhile && ./dowhile
6
```

Nesting

We can nest pretty much any conditional or loop inside of another conditional or loop

- Try not to nest too many loops at once, it'll slow down your program quite a bit

A short example

```cpp
#include <iostream>

int main(int argc, char* argv[]){

    for(int i = 0; i < 10 ; i++)
    {

            if( i % 2 == 0){
                std::cout << i << " is even" << std::endl;
            }

            else{
                std::cout << i << " is odd" << std::endl;
            }
    }

}
```

```
~/CSC 211/discussions/ $ g++ nest.cpp -o nest && ./nest
0 is even
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
8 is even
9 is odd
```

Exercise 1

```
int main(int argc, char* argv []){

    char op;
    float a, b;

    // operators should only be *, /, +, -
    std::cout << "enter an operator: " << std::endl;
    std::cin >> op;
```

Complete the following tasks for a simple calculator

1. create two functions that handle the logic in different ways
    - One that handles the calculator logic using if/else statements
    - One function that handles the same logic with a switch statement
2. Print the equation and the result
    - This should be in the form <num1> <op> <num2> = <output>

Note: You only need to handle the four basic operators (*, /, +, -) but you can do others if you finish early

- Keep in mind that modulo (%) is different from division (/)

Beyond the basics

Useful Things to Remember

- In C/C++, **everything** can be evaluated to either true or false
- Examples:
- if(1)
- if(myCharStr)
- if (!varName)
- while(str[i])
- etc

```cpp
#include <iostream>

// a program that returns the grade in a given range
int main(int argc, char* argv[]){

    int grade;

    std::cout << "enter a grade" << std::endl;
    std::cin >> grade;


    switch(grade) {
      case 90 ... 100:
         std::cout << "A" << std::endl;
        break;

      case 80 ... 89:
         std::cout << "B" << std::endl;
        break;

      case 70 ... 79:
        std::cout << "C" << std::endl;
        break;

      case 60 ... 69:
        std::cout << "D" << std::endl;
        break;

      case 0 ... 59:
        std::cout << "F" << std::endl;
        break;

      default:
        // catch-all for anything else
        std::cout << "That's not a grade!" << std::endl;
    }
```

Ranged Switch Statements

Recall the range keyword from Python and how we used it

C++ range statements have a similar functionality that allows us to evaluate an expression between a minimum and maximum value

- We identify these ranged by three periods with a space on either side (…)

We can find ranges between both ints and chars

- 1 … 99
- ‘A’ … ‘Z’

```
~/CSC 211/discussions/basicsPlus/ $ g++ rangeswitch.cpp -o range && ./range
enter a grade: 67
D
```

Conditional formatting

Don’t see any brackets around a conditional or loop? Assume the next line is iterated over/ run on the logic above it

- We call these kinds of statements one-liners

```
if(input % 2 == 0)
    std::cout << "Even" << std::endl;

else
    std::cout << "Odd" << std::endl;
```

Or

```
if(input % 2 == 0) std::cout << "Even" << std::endl;

else std::cout << "Odd" << std::endl;
```

More formatting

Recall the following snippet from the std::min we looked at last week

```
template<class T>
const T& min(const T& a, const T& b)
{
    return (b < a) ? b : a;
}
```

This is called a ternary operator

- result = (condition) ? true_branch : false_branch

```
#include <iostream>
#include <string>

int main(int argc, char* argv[]){

    int input;

    std::cout << "enter a number: ";
    std::cin >> input;

    // storing the result in a string
    std::string result = (input % 2 == 0) ? "Even" : "Odd";

    std::cout << result << std::endl;
}
```

```
~/CSC 211/discussions/basicsPlus/ $ g++ branch.cpp -o branch && ./branch
enter a number: 15
Odd
```

Pre-increment and post-increment operators

Pre-increment

- Before assigning a value to the variable, the value is incremented by one
- ++i

Post-increment

- After assigning a value to the variable, the value is incremented by one

  i++

```
int main() {

    int i = 5;

    std::cout << "The post-incremented value" << std::endl;
    while(++i < 10 )
    std::cout << i << std::endl;

    std::cout << "\n" << std::endl;

    int j  = 5;

    std::cout << "The pre-incremented value" << std::endl;
    while(j++ < 10 )
    std::cout << j << std::endl;
```

```
The post-incremented value
6
7
8
9


The pre-incremented value
6
7
8
9
10
```

More on Using For Loops

- The 3 sections can be anything you want
- They are always executed in the same order
- ++i could be replaced with i+=2, or some other expression too

Exercise 2

Using a pair of nested loops, do the following

1.) Print out the following statement
   - X + Y = Z
     i.) Where x is all even numbers up to and including 10
     ii.) Y are powers of 3 starting at 1 and including 81
     iii.) Z is the sum of X and Y

Arrays

The basic setup

Arrays are analogous to lists in Python

Same general idea, store a bunch of things in one continuous structure

The difference lies in how we're able to utilize them

- Only things of the same type can be in a C++ array
- We need to tell the compiler how much space we need as we initialize the array

```
int main(int argc, char* argv[]){

    // type      // contents
    int arr[4] = {10, 20, 30, 40};
        //size

    std::cout << arr[3] << std::endl;
```

```
int main(int argc, char* argv[]){

    // type      // contents
    std::string arr[5] = {"volvo", "ford", "bmw", "subrau", "toyota" };
        //size

    std::cout << arr[1] << std::endl;
```

Iterating through an array

Keep in mind that we can treat strings similarly to how we do in Python - as an array of characters

Note that when setting equal to a string, we don't need to explicitly state the array size

```
int main(int argc, char* argv[]){

    // type     // contents
    char arr[] = "Hello";
        //size

    int i = 0;
    while(arr[i]){
        std::cout << arr[i] << std::endl;
        i++;
    }
```

```
~/CSC 211/discussions/arrays/ $ g++ iter.cpp -o iter && ./iter
H
e
l
l
o
```

One last thing

Survey

Here is a google survey that I made to rate discussion sessions

Please fill this out so I can better understand how you'd like these discussions to be conducted

- If you do not fill it out I won't know what to change
- There are some things I will need to go over/ go through quickly, but I can do my best to accomidate