Additional Loop Types and Control

CS 10A – LOOPING PART 2

Do While Loops

- The third type of loop is known as a do-while loop.
- It’s similar to a while loop, but has one key difference: it will always run at least once.
- A do while loop checks it’s loop condition at the end of the loop instead of the beginning.
- Less commonly used, but useful for certain applications such as input validation, because those commands need to run at least once.
- A semicolon will be needed at the end of the statement where the while logic is defined.

Do While Syntax

```
char select = ‘y’;
int main()
{
        do
        {       // Brackets are optional if the loop contains only one command
                cout << “Here is your drink.” << endl;

                cout << “Would you like another? (y/n) ”;
                cin >> select;
                cin.ignore();
        } while (select == ‘y’ || select == ‘Y’); // Notice the semicolon at the end

        return 0;
}
```

Do While Example

Program

```
int i = 0;
int main()
{
        do
                cout << “Hello World!\n”;
        while(i < 0);

        // This statement is false, so the loop
        // executes once then moves on

        return 0;
}
```

Console

```
➢ ./a.exe
Hello World!
```

Loop Breaking

- The same break command that you use in switch-case statements can also be used in loops.
- Breaking a loop stops program flow right at where you use the break statement, exits the loop, and then proceeds with the rest of the code.
- While your loop logic should be written such that you won't need to use break statements, they can be useful for breaking out of nested loops.
- Each break statement breaks exactly one loop.
- Do NOT break loops. It's a bad practice, just like goto.

Loop Break Example

Program

```
int main()
{
        for(int i = 0; i < 3; i++)
        {
                for(int j = 0; j < 10; j++)
                {
                        cout << ‘*’;
                        if(j == 5)
                                break;
                }
                cout << endl;
        }
        return 0;
}
```

Console

```
➢ ./a.exe
******
******
******
```

Loop Continuing

- In contrast to loop breaking, there's also loop continuing.
- Continuing is the act of skipping the rest of the commands in the loop and immediately jumping to the next loop iteration.
- Also limited in use, but it still has it's applications, such as when you want to ignore a huge swath of commands without having to resort to encompassing all of the remainder commands in an if statement.
- Just like breaking, continuing can only affect one loop per each use of continue.
- Just like breaking, you should try to avoid using continue.

Loop Continue Example

Program

```
int main()
{
        for(int i = 0; i < 3; i++)
        {
                for(int j = 0; j < 10; j++)
                {
                        cout << ‘*’;
                        if(j > 4)
                                continue;
                        cout << ‘#’;
                }
                cout << endl;
        }
        return 0;
}
```

Console

- ./a.exe

```
*#*#*#*#*#*****
*#*#*#*#*#*****
*#*#*#*#*#*****
```