Lecture 4 Notes

- Recall the **if-ladder** from last lecture

```
if (income < 30000)
  cout << "Low";
else
{
  if (income >= 30000 && income < 100000)
    cout << "Middle";
  else
  {
    if (income >= 100000 && income < 500000)
      cout << "High";
    else
      cout << "Very high";
}}
```

- The if-ladder can get really messy, so pay extra attention to the number and relative positions of if and else

Switch Statements

- Let's say we have a program that asks the user to make a choice from one to five:

```
int main()
{
  ...
  int choice;
  cin >> choice;
  if (choice == 1)
    cout << "Do thing A" << endl;
  else if (choice == 2 || choice == 4)
    cout << "Do thing B" << endl;
  else if (choice == 3 || choice == 5)
    cout << "Do thing C" << endl;
  else
  {
    cout << "Choice must be 1 through 5." << endl;
    cout << "I'll assume you wanted choice 1" << endl;
    cout << "Do thing A" << endl;
}}
```

- There's a more convenient way to write the code above:

```
int main()
{
  ...
  int choice;
  cin >> choice;

  // Switch Statement
  switch (choice)
  {
    // Equivalent to saying 'if (choice == 1)'
    case 1:
      cout << "Do thing A" << endl;
      break;

    // Equivalent to saying 'if (choice == 2 || choice == 4)'
    case 2:
    case 4:
      cout << "Do thing B" << endl;
      break;

    case 3:
    case 5:
      cout << "Do thing B" << endl;
      break;

    // This is used when your input does not match any of the cases
    default:
      cout << "Choice must be 1 through 5." << endl;
      cout << "I'll assume you wanted choice 1" << endl;
      cout << "Do thing A" << endl;
      break; // This 'break' is optional since it is at the bottom
  }
  ...
}
```

- This is called a **switch statement**
    - Initiate it with `switch (variableName)`, followed by curly braces
        - `variableName` is the name you defined as an int
    - Write out `case`, followed by the specific number you want to assign to it
    - If another number satisfies the same condition, write it as a separate case on the next line
    - `break` tells the program to ignore everything else that comes afterwards within the switch statement
    - If the int does not match any of the cases, it will jump to the `default` section

Frequently Asked Questions

- **What happens if you accidentally forgot to include a break?**
    - The program will run the code for subsequent cases until it sees another break
- **Could you use a range of numbers for cases?**
    - No, you need to type each case separately
- **Could you use `>` or `<` signs for cases?**
    - No, you can only assign a certain case to a specific integer
- **Could you use a switch for strings or doubles?**
    - No, you can only use integers

In summary, **switch statements** are an alternate way to write if-statements. Although they are clearer to read, they are very limited and only work with integers.

While-Loops

- Let's say I want a program that will output the following:

```
How many times do you want to be greeted? 3
Hello
Hello
Hello
```

- We could write plenty of if-statements, but this is not practical for large numbers:

```
cout << "How many times do you want to be greeted> ";
int nTimes;
cin >> nTimes;

if (nTimes >= 1);
  cout << "Hello" << endl;
if (nTimes >= 2);
  cout << "Hello" << endl;
if (nTimes >= 3);
  cout << "Hello" << endl;
if (nTimes >= 4);
  cout << "Hello" << endl;
if (nTimes >= 5);
  cout << "Hello" << endl;
```

- Instead, we can use a while-statement, which is very similar to an if-statement
    - The difference is that if-statements only execute the program once
    - While-statements loop back and check the original condition again until it is no longer true

- Now, the program is as follows:

```
cout << "How many times do you want to be greeted> ";
int nTimes;
cin >> nTimes;  // Let's say we want to be greeted 3 times

int n = 1;
while (n <= nTimes)
{
  cout << "Hello" << endl;
  n = n + 1; // Reassigns n and loops back to beginning of the while-statement
}
```

Shorthands for Assignment Statements

- These assignment statements are particularly useful for loops, and can be condensed as follows
  - `n = n + 1` becomes `n += 1`
  - `m = m * 2` becomes `m *= 2`
  - `k = k / 10` becomes `k /= 10`

- There are even shorter ways to write the same thing
  - `n++` = `++n` = `n += 1` = `n = n + 1`
  - `n--` = `--n` = `n -= 1` = `n = n - 1`

- Pay attention to `<` vs. `<=` and `>` vs. `>=` when writing loops
  - They can cause you calculations to be off by one

- Let's consider an infinite loop, such that there is no assignment statement at the end

```
int n = 1;
while (n <= 10)
{
...
...
...
...
...
}
```

- The variable n will be the same every time, so the loop goes on forever
- If you run an infinite loop, you need to close the window or hit `Ctrl+C` to exit out of the program

For Loops

- There is another type of loop called a **for-loop**

```
for (int n = 1; n <= nTimes; n++)
  cout << "Hello" << endl;
  ...
```

- It has the following syntax:

```
for (initialization; stay_In_Loop_Condition; prepare_For_Next_Iteration)
  statement
  ...
```

- The advantage of a for-loop is that all the necessary components are set up at the top of the command
    - This leaves less room for error

- You can still have an infinite loop
    - In the example below, our “prepare-for-next-iteration” is set so that `k` gets added by 1
    - This will always satisfy the “stay-in-loop” condition
    - The computer will perform this operation until it hits 1 billion, the largest value for an int

```
for (int k = 10; k >= 0; k++)
  cout << k << endl;
  ...
```

Example

- The following program writes out the powers of 2 that are less than 1000, starting at 1

```
for (int n = 1; n < 1000; n *= 2)
  cout << n << endl;
```

- The output will look like

```
1
2
4
8
16
32
64
128
256
512
```