Lecture 6 Notes

Recap of Last Lecture

```
#include <iostream>
#include <string> // Make sure you include this when you're using strings
#include <cctype> // Include this for isdigit
using namespace std;

int main()
{
  cout << "Enter a phone number: ";
  string phoneNumber;
  getline(cin, phoneNumber);

  int numberOfDigits = 0;
  for (int k = 0; k != phone number.size(); k++)
  {
    if (isdigit(phoneNumber.at(k)))
      numberOfDigits++;
  }
  if (numberOfDigits == 10)
    cout << "That's a valid phone number" << endl;
  else
    cout << "A phone number must have 10 digits" << endl;
}
```

- `#include <cctype>` allows us to use the following:
    - `if (isdigit(some character))` tests whether a character is a digit, and it will store `true` if it is indeed a digit
    - `if (isupper(some character))` tests whether a character is an uppercase letter, and it will store `true` if it is indeed uppercase
    - `if (islower(some character))` tests whether a character is a lowercase letter, and it will store `true` if it is indeed lowercase
    - `if (isalpha(some character))` tests whether a character is an uppercase or lowercase letter, and it will store `true` if it is indeed any letter

- It is important to make sure strings are written in double quotes, while characters are written in single quotes

```
string s = "Hello"; // s is a string; "Hello" is a string literal
char c = s.at(1); // c is a char (initialized to e) and written as single quotes

char c = 'x'; // OK
char c = "x"; // Error! Won't compile
string s = "x"; // OK
string s = 'x' // Error! Won't compile
```

- Remember that the first character in a string corresponds to the 0th character
- The `tolower` feature turns an uppercase letter to a lowercase letter
    - If you use it on a lowercase letter, it will give back the same character
    - If you use it on a symbol (i.e. #, $, %), it will give back the same character

```
string s;
getline(cin, s); // Hello there! (H is character 0)

s.at(0) = tolower(s.at(0)); // The 'tolower' function turns 'H' to 'h'
char c = tolower(s.at(0)); // Can be stored as a character too
```

- The `toupper` feature turns a lowercase letter to a uppercase letter
    - If you use it on a uppercase letter, it will give back the same character
    - If you use it on a symbol (i.e. #, $, %), it will give back the same character
- When you call `toupper` or `tolower`, you need to do something with it, like initialize a new character
    - If you just call it, it will not do anything useful
- Common mistakes

```
toupper(s.at(0)); // Mistake: It doesn't do anything useful
toupper(s); // Error if s is a string; toupper takes a char, not a string
s = toupper(s); // Error if s is a string; toupper takes a char, not a string
```

- If `s` is the empty string, the `tolower` or `toupper` function below does not work

```
string s;
getline(cin, s);
s.at(0) = tolower(s.at(0));
```

- To fix this, you can use an if-statement

```
string s;
getline(cin, s);
if (s != "") // or you can say if (s.size() != 0)
  s.at(0) = tolower(s.at(0));
```

Functions

- Consider this analogy of a cookbook

```
p. 47
...
...
...
Make the icing
...
...

Icing:
...
...
```

- Once you get to **`Make the icing`**, the program will jump to the **`Icing`** section
    - Once done with the icing section, it will return to the line after **`Make the icing`**

- The following program could be condensed by defining a function

```
int main()
{
  ...
  ...
  for (int k = 0; k < 3; k++)
    cout << "Hello" << endl;
  ...
  ...
  for (int k = 0; k < 3; k++)
    cout << "Hello" << endl;
  ...
  ...
  ...
  for (int k = 0; k < 3; k++)
    cout << "Hello" << endl;
  ...
}
```

- Program starts executing the `main` function first, and it refers to the `greet` function when referred

```
void greet()  // Defines a function called greet
{
  for (int k = 0; k < 3; k++)
    cout << "Hello" << endl;
}

int main() // Program starts here
{
  ...
  ...
  greet(); // Refer to greet function
  ...
  ...
  greet();
  ...
  ...
  ...
  greet();
  ...
}
```

- You can list the statements of `greet` after the main function, but you need to define that `greet` is a function before the main function

```
void greet();  // Defines greet (include a semicolon!)
int main() // Program starts here
{
  ...
  ...
  greet(); // Refer to greet function
  ...
  ...
  greet();
  ...
  ...
  ...
  greet();
  ...
}

void greet()  // Statements associated with greet
{
  for (int k = 0; k < 3; k++)
    cout << "Hello" << endl;
}
```

Another Version of the Cookbook Analogy

- Now, there are two different flavors, but the directions to making the icing is almost same

```
p. 47                                          p. 56
...                                            ...
...                                            ...
...                                            ...
Make the icing (lemon flavored)                ...
...                                            Make the icing (orange flavored)
...                                            ...

Icing:
...
Add the flavoring (parameter)
...
```

- Lemon and orange are the arguments, and the general flavoring is the parameter
- An example of the actual program is shown below:

```
void greet(int nTimeS);
int main()
{
  ...
  greet(3); // 3 is the argument
  ...
  ...
  int n;
  cin >> n; // Suppose user types 5
  greet(n+2); // Call greet(7), which is the argument
  ...
  ...
  greet(1); // 1 is the argument
  ...
}

void greet(int nTimes)  // nTimes is a parameter
{
  for (int k = 0; k < nTimes; k++)
    cout << "Hello" << endl;
}
```

- Let's take this a step further an add a custom message that you want to be greeted with

```
void greet(int nTimeS, string msg);
int main(){
  ...
  /*If I type in a string and integer ("Salaam", 2); it won't compile*/
  greet(3, "Hello");
  ...
  ...
  int n;
  cin >> n; // Suppose user types 5
  greet(n+2, "Ni hao"); // Call greet(7), which is the argument
  ...
  ...
  string s;
  getline(cin, s);
  greet(1, s);
  ...}

void greet(int nTimes, string msg)  // nTimes and msg are parameters
{
  for (int k = 0; k < nTimes; k++)
    cout << msg << endl;}
```

- A mathematical version of a function that takes the square is shown below:

```
int square(int k); // Defines the function square

int main(){
  int n = 3;
  cout << square(n) << endl; // writes 9
  int m = square(n+1) * 3; // m is 48
  ...}

int square(int k){
  return k * k; // Tells you the value you want to return to the main function
}
```

- This function takes the absolute value

```
int absoluteValue(int x);
int square(int k); // Defines the function square

int main(){
  int n = 3;
  cout << square(n) << endl; // writes 9
  int m = square(n+1) * 3; // m is 48
  cout << absoluteValue(m-50) << endl; // writes 2
  ...}

int square(int k) {
  return k * k; // Tells you the value you want to return to the main function
}

int absoluteValue(int x){
  if (x >= 0)
    return x;
  else
    return -x;}
```

- Notice that `int` is used when the function **returns** an integer to the main function, and `void` is used when the function just does its job

- For other languages, you are technically supposed to type `return 0;` after the main function
    - This is optional in C++