**Lecture 5 Notes**

---

- Recall while-loops and for-loops from last lecture

```
initialization;
while (stay-in-loop condition)
  {
  statement
  prepare-for-next-iteration;
  }

for (initialization; stay-in-loop condition; prepare-for-next-iteration)
  statement
```

- They both serve the same purpose, but are formatted differently
    - Choose the one that's simpler for you
- We want to print out three rows of four asterisks
    - Any time there is repetition, you should always think of loops

```
****
****
****
```

```
for (int r = 1; r <=3; r++)
{
  for (int c = 1; c <= 4; c++)
  {
      cout << "*";
  }
  cout << endl; // This is to start a new line after there are 4 asterisks
}
```

- We want to visit every character in a string and cout it out, followed by an endline

```
string s = "Hello";
  for (int k = 0; k != s.size(); k++) // The kth character of "Hello"
    cout << s[k] << endl; // "s sub k"
```

- `s.size()` is the **number of characters** of the string, where `s` is the name of the string
    - You might get a warning if you use a less than sign (`<`), so it is better to use `=!`
- `s[k]` is the kth character of the string **starting at 0**
- Sometimes, `s.length()` is used instead of `s.size()`, but they mean the same thing
    - Since these are numbers, you can initialize another variable and set it to the same value

```
int n = s.size();
cout << s.length() << endl;
```

- `s[k]` and `s.at(k)` performs similar functions, but `s.at(k)` is safer since it checks the validity of the value
    - Try to use `s.at(k)` since it is more correct
    - However, `s[k]` is faster since it does not check

```
s[k]      // Does not check whether k is a valid number
s.at(k)   // The program will terminate if k is out of bounds
          // (i.e greater or less than the string size)
```

- We want to count the number of “E’s” or “e’s” in some text
- The program is shown in the next page

```
cout << "Enter some text: "; // Text is "Everyone, hello!"
string t;
getline(cin, t);
int numberOfEs = 0;
for (int k = 0; k != t.size(); k++)
{
  if (t.at(k) == 'e' || t.at(k) == 'E') // Must be single quotes since this is a
character !!!
    numberOfEs++;
}
cout << "The number of Es (upper and lower case) is " << numberOfEs << endl;
```

- A character variable `char` holds one character only
    - Strings use double quotes `“string”`
    - Chars use single quotes `‘c’`

```
string s = "Hello";
char c = s.at(1); // c is a char (initialized to lower case e)
```

- Certain characters are written with a backslash and do not follow the norm
    - `‘\t’` represents the tab character
    - `‘\n’` represents the newline character
- These characters can also be placed in strings
    - For instance, a string could be written as `“ab\ncd\n”`
- Now, we want to write a program that checks if a phone number is a valid US phone number
    - There’s many formats to phone numbers, so we need to take this into account
        - 3108254321
        - (310) 825-4321
        - 310 825 4321
        - 310.825.4321
- The program is shown on the next page

```
#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;
}
```

- `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
- If you want the opposite (i.e. something that is not a digit), you can use the not (`!`) operator for the if-statement

```
if ( ! is digit(s.at(k)) )
   // will be executed only if s.at(k) is not a digit
```

- Let's say we want a to be 1 through 10 (inclusive)

```
if ( ! (a < 1 || a > 10) )

if (a >= 1 && a <= 10)
```

- Let's say we want to repeatedly read a string until it is a "yes" or "no"
- First approach:

```
cout << "Enter yes or no: ";
string response;
getline(cin, response);

while (response != "yes" && response != "no")
{
  cout << "Please respond yes or no: ";
  getline(cin, response);
}
cout << "Thank you" << endl;
```

- Second approach:

```
cout << "Enter yes or no: ";
string response;
  while (42 == 42) // We want something that is always true to enter loop
  {
    getline(cin, response);
    if (response == "yes" || response == "no")
      break; // break gets you out of a loop or switch statement
    cout << "Please response yes or no: ";
  }
cout << "Thank you" << endl;
```

- Third approach:

```
cout << "Enter yes or no: ";
string response;
  for ( ; ; ) // If you leave nothing between semicolons, it will be always true
  {
    getline(cin, response);
    if (response == "yes" || response == "no")
      break; // break gets you out of a loop or switch statement
    cout << "Please response yes or no: ";
  }
cout << "Thank you" << endl;
```