Today's Plan

Upcoming:

- Labs next week

Last time:

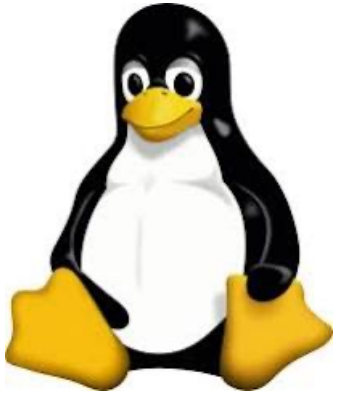
- Course outline

Today's topics:

- Getting Started with C++
- C++ Review
  - Basic Types
  - Conditional Statements: if and switch
  - Loops: while, for, for-each
  - Function Calls: Pass by Value/Reference

Getting Started with C++

- Your assignments will be compiled, run, and marked using g++ on Ubuntu Linux
- So, you should use the same environment for writing your programs
  - On Windows, installing Windows Subsystem for Linux (WSL) is the easiest way to get Linux
  - On a Mac, you can probably use the built-in terminal
- We recommend you VS Code for this course
  - Free, popular, high-quality programming editor
  - Supports C++ and lots of other languages
  - Works well with WSL



C++ Version

- We'll be using C++17
- Later version of C++ are not yet fully supported by all compilers
- If you use features beyond C++17, your programs might not compile when marked!
- We'll provide exact compiler options to make sure you use the correct version
  - in a makefile

Review of C++

- Sample code to review
  - Hello, world
  - Basic types
  - If statements and switch statements
  - Loops: while, for, for-each

```
// hello_world.cpp

#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World!\n";
    return 0;
}
```

int type (whole numbers)

```
int a;

int b = 5;

int b2(5);

int b3{5};

int c = 5.5;

auto d = 5;
```

For 32-bit ints:

min int = -2147483648
max int = 2147483647

double type (decimal numbers)

```
double e;

double f = 5;

double g = 5.5;

double g2(5.5);

double g2{5.5};

auto h = 5.5;
```

char type (single character)

```
char i; // unknown initial value
char j = 'a';
char k = 97;  // ASCII value of 'a'
auto l = 'a';
cout << j;
cout << k;
cout << int(j);
cout << int(k);
cout << int('a');
cout << char(97);
```

unsigned int type (non-negative ints)

```
unsigned int m;
unsigned int n = 5;
unsigned int p = -5;
auto q = 5u;

cout << p; // prints 4294967291
cout << q;
```

For 32-bit ints:

min unsigned int = 0
max unsigned int = 4294967295

unsigned int type (non-negative ints)

```
typedef unsigned int uint;

uint m;
uint n = 5;
uint p = -5;
```

typedef lets you give any C++ type another name. This lets you make names that are shorter or more descriptive.

typedef does **not** create a new type: it just gives another name to an existing type.

C++ string type (sequence of chars)

```
string r;
string s = "Hello World!";
string t("Hello World!");
string u{"Hello World!"};
string v(5, '!');

cout << "r = \"" << r << "\"";

cout << "v = \"" << v << "\"";
```

Important!

string is the name of the standard C++ string type, and it's what you should use in this course

A **C-style string** is an array of characters ending with a '\0'. They are used as strings in C, but in C++ we will usually only use them for string literals like "hello world!".

IF vs. SWITCH Conditionals

```
char d = 'm';
if (d == 'm' || d == 'M')
{
    cout << "Monday";
}
else if (d == 'w')
{
    cout << "Wednesday";
}
else if (d == 'f')
{
    cout << "Friday";
}
else
{
    cout << "Invalid day";
}
```

```
char d = 'm';
switch (d)
{
    case 'm':
    case 'M':
        cout << "Monday";
        break;
    case 'w':
        cout << "Wednesday";
        break;
    case 'f':
        cout << "Friday";
        break;
    default:
        cout << "Invalid day: '";
        break;
    }
```

WHILE vs. FOR LOOP

```
int total = 0;
int i = 0;
while (i < 100)
{
    total += i;
    i++;
}


cout << total; // 4950
```

```
int total = 0;



for(int i = 0; i < 100; i++)
{
    total += i;
}
cout << total; // 4950
```

Infinite Loop using WHILE and FOR

```
vector<int> v;
for (int i = 0; i < 100; i++) {
    v.push_back(i);
}
```

Make sure to **#include <vector>** and **using namespace std;**

for loop

```
int total = 0;
for(int i = 0; i < v.size(); i++)
{
    total += v[i];
}
cout << total;
```

for-each loop

```
int total = 0;
for(int n : v)
{
    total += n;
}
cout << total;
```

Function Calls: Pass by Value

```
int count(string s) {
    int num = 0;
    for(char c : s) {
        if (c == ' ') num++;
    }
    return num;
}
```

Makes a new copy of the passed-in string **s**. Slow, and uses extra memory.

Function Calls: Pass by Reference

```
int count(string& s) {
    int num = 0;
    for(char c : s) {
        if (c == ' ') num++;
    }
    return num;
}
```

Does **not** make a copy of **s**. Works on the actual passed-in string. Fast!

Pass-by-reference vs. Pass-by-constant-reference

```
int count(string& s) {
    int num = 0;
    for(char c : s) {
        if (c == ' ') num++;
    }
    return num;
}
```

Does **not** make a copy of **s**. Works on the actual passed-in string. Fast!

```
int count(const string& s) {
    int n = 0;
    for(char c : s) {
        if (c == ' ') n++;
    }
    return n;
}
```

As fast as pass-by-reference, but **guarantees** to compiler that **count** does **not** modify **s**.

Functions: Local Variables

```
int count(const string& s) {
    int n = 0;
    for(char c : s) {
        if (c == ' ') n++;
    }

    return n;
}
```

Local variables exist only when the function is active. They are **automatically** created when the function is called, and then **automatically** deleted when the function ends.

Return values are returned (passed) by value

```
int count(const string& s) {
    int n = 0;
    for(char c : s) {
        if (c == ' ') n++;
    }
    return n;
}
```

When **n** is returned, a **copy** is returned. So **be careful** returning large values: they could use a lot of time and memory.

The returned **int** must be a copy because **n** is a local variable that's automatically deleted when the function ends.