From Java to C++

Basic Program

```
class Simple {
  public static void main(String[] args) {
    System.out.println(“Hello World”);
  }
}
```

```
#include <iostream>
using namespace std;

int main(int argc, char** argv) {
  cout << "Hello World!\n";
}
```

Built-in Types

```
class Main {
  public static void main(String[] args)   {
    int a = 0;
    String greeting = “Hello”;
    greeting += “?”;
    float test = -3.14f;
    boolean isCold = false;
  }
}
```

```
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char** argv) {
    int a = 0;
    string greeting = "hi";
    greeting += "?";
    float test = 3.14f;
    bool isCold = false;
}
```

See: types demo

Conditionals

```
cout << "Do you like jokes? (Y/N) ";
string response;
cin >> response;

cout << endl;
if (response == "Y") {
   cout << "Cool. Me too.\n";
}
else if (response == "N") {
   cout << "Yeah, me neither\n";
}
```

See: if demo

Loops

```
int sum = 0;
for (int i = 0; i < 5; i++) // for loop
{
   sum += i;
}

float values1[] = {2.0f, 3.0f, 5.0f};
for (int i : values1) // for-each semantics loop
{
   cout << i << " ";
}
```

File I/O

```
ifstream file(“grades.txt");
if (!file) // true if the file is valid
{
   cout << "Cannot load file: " << filename << endl;
   return;
}

float sum = 0;
while (file)
{
   int grade;
   file >> grade;
   sum += grade;
}
```

See: fileIO demo

Defining classes

```
class Box {
   protected float mySize = 1.0f;
   public Box(float s) {
      mySize = s;
   }

   public float getSize() {
      return mySize;
   }
}
```

```
class Box {
 public:
   Box(float s) {
      mySize = s;
   }
   float getSize() {
      return mySize;
   }
 protected:
   float mySize = 1.0f;
};
```

See: classes demo

Defining objects

```
class Main {
  public static void main(String[] args) {
      Box box = new Box();
      float v = box.value();
  }
}
```

```
int main(int argc, char** argv) {
  Box box1;
  float v1 = box1.value();

  Box* box2 = new Box();
  box2->value();
  delete box2;
}
```

See: classes demo

Struct

```
struct Point
{
    float x;
    float y;
};

int main(int argc, char** argv)
{
    Point p;
    p.x = 1.0f;
    p.y = 2.0f;
    cout << p.x << " " << p.y << endl;

    Point a{-3.0f, 2.0f};
    cout << a.x << " " << a.y << endl;
}
```

See: struct demo

Polymorphism (virtual functions)

- Subclasses can override virtual methods from their parent

```
class Animal {
 public:
  Animal() {}
  virtual ~Animal() {}
  virtual string say() const = 0;
};

class Cow : public Animal {
 public:
  Cow() {}
  string say() const override { return "Mooo"; }
};

class Cat : public Animal {
 public:
  Cat() {}
  string say() const override { return "Meow"; }
};
```

```
int main(int argc, char** argv) {
  vector<Animal*> animals;
  animals.push_back(new Cow());
  animals.push_back(new Cat());
  animals.push_back(new Cow());

  for (Animal* animal : animals) {
    cout << animal->say() << endl;
  }

  // cleanup
  for (unsigned int i = 0; i < animals.size(); i++)
 {
    delete animals[i];
  }
  animals.clear();
}
```

See: Animals demo

Special class functions

- Override built-in operators
  - stream (for printing), e.g. `ostream& operator<<(const Box& b)`
  - add/subtract/etc
- assignment operator
  - Box& operator=(const Box& b)
  - Called by `b2 = b1;`
- copy constructor
  - Box(const Box& other)
  - Called by `Box b2 = b1;`
- Destructor
  - virtual ~Box()

See: Boxes demo

Namespaces

- Helps avoid naming conflicts
- Like Java packages
- Best practice: Never put `using namespace …` in header files
  - Ok in main cpp file

See: Boxes demo

Standard Template Library (STL)

- Built-in data structures (list, array list, dictionary, string, sorting, …)
- Based on “generic” programming (templates)

std::vector

```
vector<float> values = {1.0, -2.0, 3.0};
values.push_back(-4.0);

for (unsigned int i = 0; i < values.size(); i++) {
   cout << values[i] << endl;
}

values.clear();
values = vector<float>(10);
for (unsigned int i = 0; i < 10; i++) {
  values[i] = i;
}

values[1] *= 10.0;
for (float v : values) {
  cout << v << endl;
}
```

See: vector demo

std::string

```
string phrase = "the quick, brown dog";

if (phrase.find("quick") != string::npos) {
  cout << "Found quick in phrase!\n";
}

cout << "The string length is " << phrase.size() << std::endl;

string newphrase = "";
for (unsigned int i = 0; i < phrase.size(); i++) {
   if (phrase[i] == 'i') newphrase += "1";
   else if (phrase[i] == 'o') newphrase += "0";
   else if (phrase[i] == 'e') newphrase += "3";
   else newphrase += phrase[i];
}

cout << "newphrase: " << newphrase << endl;
```

See: string demo

std::map

```
map<string,int> names2age;
names2age["giles"] = 54;
names2age["buffy"] = 18;
names2age["joyce"] = 38;

cout << "Number of items: " << names2age.size() << endl;

for (auto it = names2age.begin(); it != names2age.end(); ++it) {
  cout << it->first << ", " << it->second << endl;
}

names2age.clear();
names2age = { {"giles", 54}, {"buffy", 18}, {"drusilla", 176} };

for (auto [key, value] : names2age) {
  cout << key << ", " << value << endl;
}
```

See: map demo

Parameters: pass by reference, pass by value

```
void example(const Box& box) {
    int v = box.size;
}

void example(Box& box) {
    box.size = 20;
}


void example(Box box) {
    box.size = 20;
}
```

See: parameters demo

Examples: parameter passing

```
#include <iostream>
using namespace std;

void foo(string& text)
{
    // text CAN be modified!!
}

int main(int argc, char** argv)
{
    string word = “apple”;
    foo(word);
}
```

```
#include <iostream>
using namespace std;

void foo(const string& text)
{
    // text CAN NOT be modified!!
}

int main(int argc, char** argv)
{
    string word = “apple”;
    foo(word);
}
```

Parameter passing (old fashioned way)

```
#include <iostream>
using namespace std;

void foo(string* text)
{
    // text CAN be modified!!
    *text = “apple”;
}

int main(int argc, char** argv)
{
    string word = “apple”;
    foo(&word);
}
```

Return values

```
#include <iostream>
using namespace std;

string foo()
{
    return “apple”;
}

int main(int argc, char** argv)
{
    string word = foo();
}
```

```
#include <iostream>
using namespace std;

string& foo()   // NEVER return reference
{
    return “apple”; // returning an object that will be deleted
}

int main(int argc, char** argv)
{
    string word = foo();
}
```

Exercise: Snacks

- Can you write a program that reads a file into an array of struct Snack?

Exercise: Snacks

```
// Bryn Mawr College, 2021
// Write a program which reads `snacks.txt` and initializes an array of struct Snack

#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
using namespace std;

// TODO: Define struct Snack

int main(int argc, char** argv)
{
   string filename = "../files/snacks.txt";

   // TODO: Your code here
}
```

snacks.txt

```
7
Slurm 5 1 Flourescent_goodness
DietSlurm 0 1 Flourescent_goodnes,_no_calories
SlurmClassic 3 1 Nostalgic_flavor
CherrySlurm 5 2 Flourescent_Cherry
OrangePuffs 8 5 Salty_and_cheesy
CandyCauliflower 2 3 Sticks_to_teeth
MagicBeans 1 100 Are_they_worth_the_hype
```

Development workflow

Git Workshop: What is git and Github?

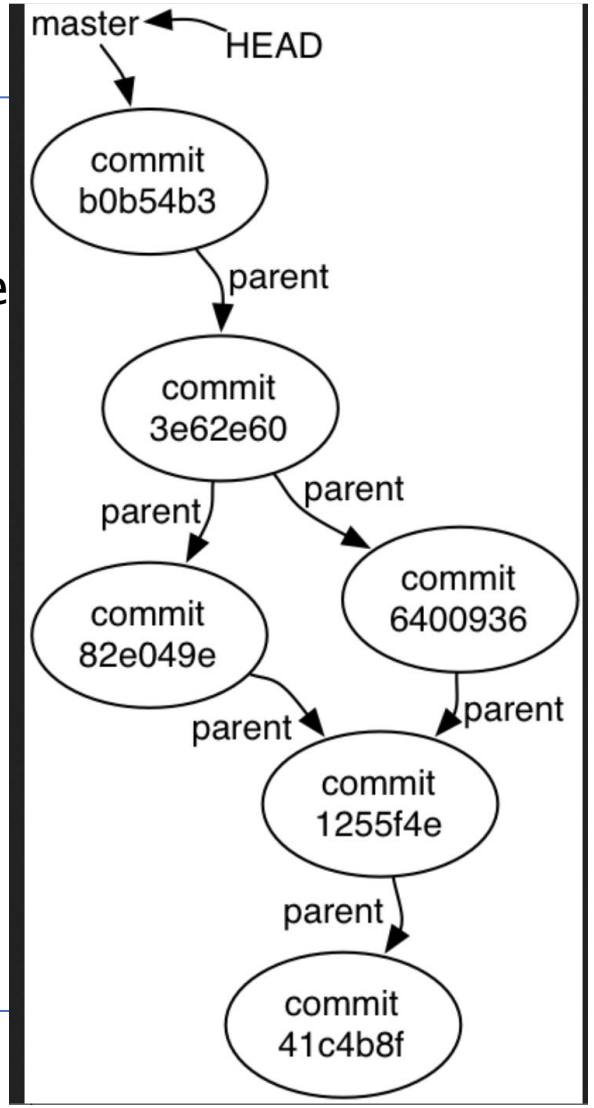
- git is a **Version control system** (aka source control system)
- Github is an online hosting platform for git repositories
  - **repository:** a store of the files in our project

Git Workshop: Let’s learn by doing!

1. Create a Github account (if you don’t have one already)
2. Commit a file
    1. Create repository: git-practice
    2. Commit a file: `hello.txt`
    3. Create a readme on github and answer questions
3. Pulling and merging
4. How does it git work?
5. ..a few odds and ends

Git Workshop: Under the hood

```
alinen@xin MINGW64 hello-git (master)
$ git lol
* b0b54b3 (HEAD -> master, origin/master, origin/HEAD) Gree
*   3e62e60 Merge
|\
| * 6400936 Greeting in Scheme
* | 82e049e Greeting in Ruby
|/
* 1255f4e Change the greeting
* 41c4b8f Initial commit
```



Git: Some odds and ends

- `.gitignore` contains a list of files for git to ignore
  - Build files are listed here!!

- *.md files are markdown files
  - Text files with annotations to control formatting (like html)
  - Github automatically displays these