C++ Lesson 1

A First Program
Print Statements
Functions
If/Switch Statements
For/While Loops

C++ Lesson 1: Summary

- No need to always use OOP
- int main() {}
- cout << “Some Text” << x << endl;
  - #include <iostream>
- printf(“Some text %d more text %s\n”, x, str.c_str());
  - #include <stdio.h>
- Variable declaration, functions, comments syntax is the same
- If statement, For/While Loop syntax is the same

C++ Lesson 2

Header Files

C++ Lesson 2: Summary

In C++ forward declare functions, classes in a header file.

Then include the header file in the cpp file.

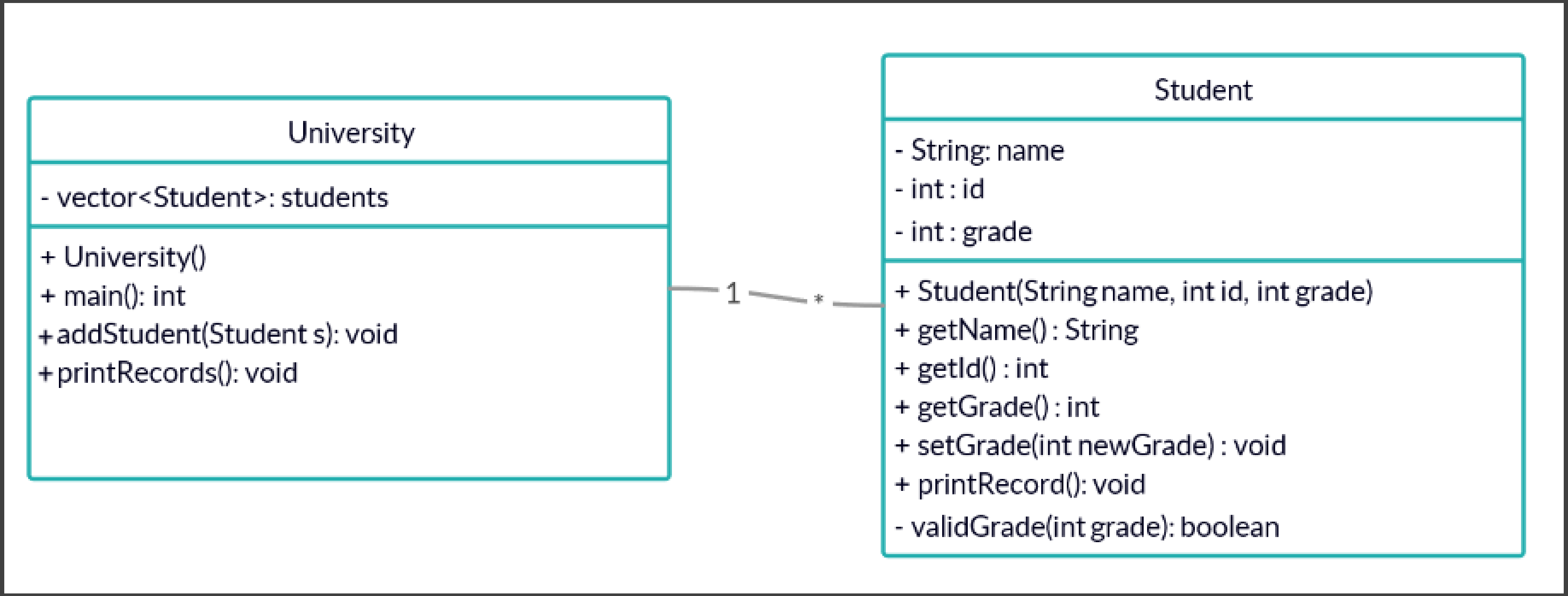
```
#ifndef FIRSTPROGRAM_H
#define FIRSTPROGRAM_H

Import Statements
Definitions of Constants
Class, Function Declarations, Etc.

#endif
```

```
#include “FirstProgram.h"
```

C++ Lesson 3

Classes and Objects
Vectors
Structs

UML Diagram



creately.com

validGrade: return true if the grade is between 0 and 100

C++ Lesson 3: Summary

Class Declaration

```
class Name {
    private:
    public:
};
```

Method Implementation

```
returnType ClassName::methodName(int x, int y) {
}
```

C++ Lesson 3: Summary

Constructor

```
ClassName::ClassName(int x, int y) {
    this->x = x;
    this->y = y;
}
```

Calling the Constructor

```
ClassName varName(x, y);
```

C++ Lesson 3: Summary

Working with Vectors

```
myVector = vector<Type>();

myVector.push_back(x);

for (int i = 0; i <  myVector.size(); i++) {

        myVector[i] ...

}
```

C++ Lesson 3: Summary

A struct is the same thing as a class

- In a struct everything is **public** by default
- In a class everything is **private** by default

[structs are typically used to bundle together smaller bits of data]

C++ Lesson 4

Pointers

Pointer Basics

- Regular variable declaration and assignment
  - int x = 1;
- A pointer is a variable that stores an address of another variable
- “Address of” operator fetches the address of some variable
  - int *a = &b;
- Dereference operator “goes to” the address stored in the pointer
  - int val = *a;

Pointers with Functions

- Does Nothing
  - void add(int x, int y)
- Updates Variables
  - void add(int *x, int *y)
- Updates Pointers
  - void add(int **x, int **y)
- Do not return addresses to local variables!
- Always initialize your pointers!
- Do not deference null pointers!

Pointers with Classes

- New Constructor
  - SampleClass(string name, int id) : name(name), id(id)
- Deference Operator
  - (*p).getName() = p->getName()

C++ Lesson 5

References
Const Qualifier

References

- Are variables that hold direct, fixed links to other variables
- Cannot be re-assigned
- Does Nothing
  - void add(int x, int y)
- Updates Variables
  - void add(int &x, int &y)
- Updates Pointers
  - void add(int *&x, int *&y)

References and Classes

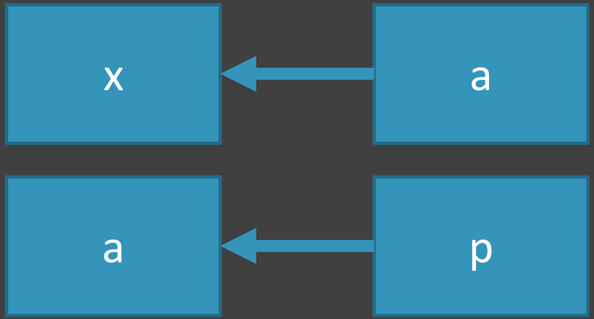
- Does Nothing
  - void foo1(SampleClass s)
- Updates Variables
  - void foo2(SampleClass &s)
  - void foo3(SampleClass *s)

Const Qualifier

- Pass in by constant reference
  - void foo4(const SampleClass &s)
- Const qualify a method
  - int getId() const

Star [*] and Ampersand [&] in C++

- Use in the variable declaration
  - int *a: declaration of a POINTER
  - int &a: declaration of a REFERENCE [C++ only]
- Anywhere else before a variable
  - *a: dereference operator
  - &a: "address of" operator
- The following "cancel out" in regular code
  - &*a = a [a is a pointer]
  - *&a = a [a is a variable]



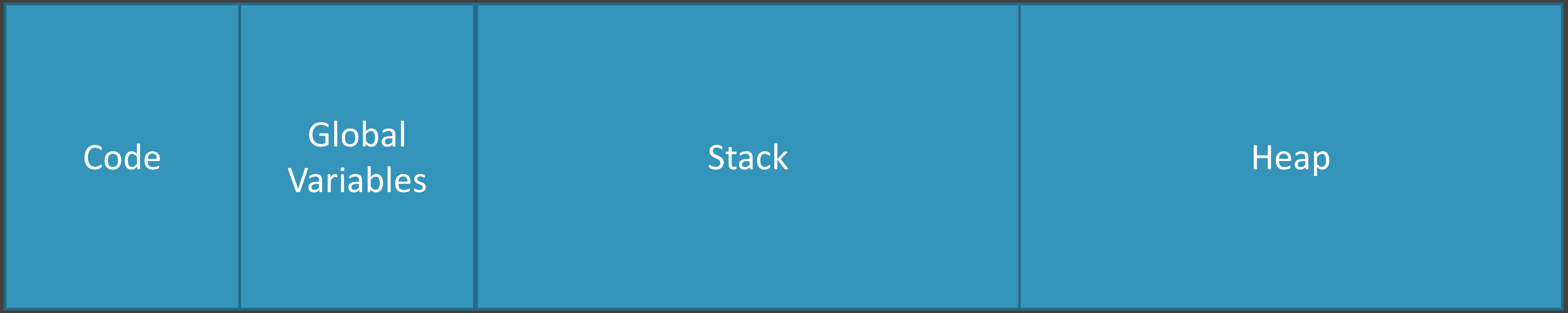
C++ Lesson 6

Additional Exercises (on pointers and references)

C++ Lesson 7

Memory Management
Dynamic Constructors
Destructors
Memory Leaks

Memory Layout



Stack: Stores local variables, etc. statically in the chronological order at *compile time*

Heap: **G**eneral purpose memory, allocated dynamically on request at *runtime* [memory allocated using the NEW keyword]

Memory Leaks

new -> delete

Constructor -> Destructor

# [new] = # [deletes]

```
ClassName name(..., ..., ...);
ClassName *name = new  ClassName(..., ..., ...);
~ ClassName();
```

```
valgrind ./programName
```

Types of Memory Issues

- Segmentation Fault / Dangling Pointer:

  trying to use a pointer that went out of scope

- Memory Leaks

- Double Free Errors

  trying to delete the same pointer twice

  or trying to delete a pointer which never been allocated