C++ Part 4

Recap

- Memory allocation and pointers
- User-defined functions
- Pairs
- Pass by reference
- Recursion

Header files

- Header (*.h) files are useful for factorizing code
  - Declarations/interfaces in header and definitions/implementations in source code
- Include user-defined headers as #include "header.h"
  - Can use relative paths with "": #include "../dir/header.h"
- Good practice: include header files (and libraries) at the lowest-level possible
  - Included files are passed on with subsequent #include statements
- If using multiple source code files, include all in compile command
  - Package managers and makefiles can handle this for you

Classes: intro

- Classes and object are the main aspects of object-oriented programming
- Class: a template for objects with various attributes and functions (methods)
- Object: an instantiation of a class with defined values for attributes

Class: country	USA	France	Japan
int year_established	1776	843	-660
std::string continent	"North America"	"Europe"	"Asia"
float population	3.33e8	6.8e7	1.25e8

Classes: declaration and member attributes

- Class must be declared before any instances can be created
- Class name, attributes and methods are declared together
- Attributes can be objects of any type
- Access attribute using . or -> followed by attribute name

```
class MyClass {
 public:
   int var1;
   float var2;
};
```

```
MyClass obj;
obj.var1 = 3;
std::cout << obj.var1 << std::endl;
```

Classes: member methods

- Classes can have dedicated methods that operate on class attributes
- Methods can be defined in-line (in declaration) or separately
    - Class namespace needed when defining separately

```
class MyClass {
  public:
    int var;
    int getvar(){return var;}
    void printvar();
};
```

```
void MyClass::printvar() {
  std::cout << var << std::endl;
}
```

Classes: access specifiers

- Class attributes and methods are given access keywords
  - public: accessible from outside the class
  - private: cannot be accessed outside the class
  - protected: cannot be accessed outside the class, but can be accessed by derived classes
- Good practice: keep attributes private and use public accessors

```
class MyClass {
  public:
    void setvar(int);
    int getvar();
  private:
    int var;
};
```

```
void MyClass::setvar(int newvar) {
  var = newvar;
}
int MyClass::getvar() {
  return var;
}
```

Classes: constructor

- Constructors generally defined to perform functions that are needed
  - Instantiate and initialize member attributes
  - Allocate memory for pointers
- Default constructor creates object but doesn't initialize anything
- Called whenever a new instance of the class is created

```
class MyClass {
  public:
    MyClass(int);
    MyClass() = default;
  private:
    int var;
    float * pointer;
};
```

```
MyClass::MyClass(int newvar) {
  std::cout << "Making MyClass" << std::endl;
  var = newvar;
  pointer = new float;
}
```

Classes: initializer list

- Class attributes can be initialized with initializer list
- Can only be done for objects with a default constructor
- Executed before constructor
- Considered better practice

```
class MyClass {
  public:
    MyClass(int);
  private:
    int var;
    float * pointer;
};
```

```
MyClass::MyClass(int newvar) :
  var(newvar), pointer(nullptr)
{
  std::cout << "Making MyClass" << std::endl;
  pointer = new float;
}
```

Classes: destructor

- Destructors are used to perform functions needed when object is deleted
  - When object goes out of scope or when pointer is deleted

```
class MyClass {
  public:
    MyClass(int);
    ~MyClass();
    void setvar(int);
    int getvar();
  private:
    int var;
    float * pointer;
};
```

```
MyClass::~MyClass() {
  std::cout << "Bye from MyClass!" << std::endl;
  delete pointer;
}
```

Classes: inheritance

- Classes can inherit structures from one another
  - Useful to minimize redundant code
- Derived classes gain all public or protected members of base class

```
class Vehicle {
  public:
    int size;
  protected:
    std::string fuelType;
    void start();
};
```

```
class Car : public Vehicle {
  public:
    int getSize() {return size;}
    void ignition() {start();}
  protected:
    std::string make = "Ferrari";
};
```

Maps

- A std::map (map library) holds a variable length set of key/value pairs
- Useful for storing information associated with list of names
- Easiest access (read or write) uses mymap[<key>]
    - If element doesn’t exist, it has default value or is assigned
    - If element exists, value is read or value is overwritten
- Key and value accessed with first and second when looping over elements

```
std::map<std::string,int> mymap;
mymap.clear();
mymap["a"] = 3;
std::cout << mymap.size() << std::endl;
std::cout << mymap["a"] << std::endl;
```

```
for(auto const& x : mymap) {
  std::cout << x.first << std::endl; // key
  std::cout << x.second << std::endl; // value
}
```

Default argument values

- Functions arguments can be given default values
    - If no argument is given, default value is used
    - Arguments with optional values must be at the end of list
- Define default value in declaration

```
int sum(int x, int y = 0) {
  return x + y;
}
int main() {
  std::cout << sum(8,5) << std::endl;
  std::cout << sum(8) << std::endl;
  return 0;
}
```

Resources

- https://www.w3schools.com/ - Great online learning resource
- https://www.youtube.com/@codebreakthrough - Excellent tutorial videos
- https://en.cppreference.com/w/ - Thorough documentation
- https://stackoverflow.com/ - Ask questions to experts