

C++ Classes

Outline

- Objects in Libraries
- Constructors
- Destructors
- What compiler creates for you
- friends
- Dynamic memory and objects
- RAII

Objects in Libraries

- See `Library_classes_demo`

The big picture before the details

- Provide a constructor (or the compiler will, and it will be wrong!)
- When your object holds dynamic data (pointers and new memory) you want to ensure deep copies so....
- **YOU MUST IMPLEMENT**
 - Destructor
 - Copy constructor
 - Assignment operator (one of many operators)
- If you are lazy make the copy and assignment operator private, then they cannot be copied or assigned
- Use the destructor to deallocate your dynamic memory

Constructors

- Default constructor (no arguments)
 - `Classname::classname()`
 - Overloaded constructors (with arguments)
 - `Classname::classname(type varName,...)`
 - Copy constructor
 - Assignment operator
- } Get to these in a bit
- Constructors set up the object for use

Constructors- Member Initialization

- If you do NO initialization
 - For Objects – their default constructor is called
 - Primitives (ints, bools, doubles, longs, char etc) – **NO INITIALIZATION AT ALLBest to initialize...**

```
class NoMemberInitilization {  
private:  
    std::string myString;  
    int         myInt;  
  
public:  
    NoMemberInitilization();  
    virtual ~NoMemberInitilization();  
};  
  
NoMemberInitilization::NoMemberInitilization() {  
    //myStrings no argument (default) constructor called set to 0  
    //myInt has garbage in it  
}
```

Constructors- Member Initialization

- 2 ways
 - Initialize in constructor body
 - Initializer list (uses copy constructor)
- Which is better?
- Construction of objects proceeds in 2 phases
 - 1. Initialization of Data members
 - 2. Execution of the body of the constructor that was called
- So if you initialize in constructor body, you initialize an object with default constructor then assign in constructor body. 2 calls
- **Also what if 1 of data members is const? Must use initializer list!**
- **TLDR always use Initializer list, faster and handles const data members.**
- **See `library_classdemo` - `class_initializeinconstructor` and `classs_initlist`**

Destructor

- A function that gets called when an object is destroyed.
- Called when object goes out of scope (whether statically or dynamically allocated)
- Its purpose is to clean up after object
 - dynamically allocated memory that the object has pointers to
 - Close open filestreams
 - Close database connections
 - Close network connections
- Syntax:
`classname::~~classname();`

See **library_classdemo** – `class_destructor`

Destructor

Objects with dynamic memory

- Already seen this
- If dynamic memory allocated deallocate it.

```
HoldsWithDynamicData::~~HoldsWithDynamicData() {  
    if (ps)  
        delete[] ps;  
}
```

- **Must** write yourself if object has dynamically allocated members, or object members that cannot make a copy of themselves.
- Otherwise let compiler handle it

Default IDE Class Creation

- Creates .h and .cpp file (header and implementation)
- With same name as class
- With a no argument constructor and a destructor
- Optional: Can define namespace to protect against collisions (keith::string will not conflict with std::string)
- **Do namespaces after class is working**

```
#pragma once

namespace keith{
    class defaultClass {
    private:
        int i;
    public:
        defaultClass(int i);
        virtual ~defaultClass();
    };
}
```

```
#include "defaultClass.h"

using namespace keith;

defaultClass::defaultClass(int i) {
    this->i=i;
}

defaultClass::~~defaultClass() {
}
```

Compiler created Functions

- Given this class

* defaultClass.cpp

```
#include "defaultClass.h"
defaultClass::defaultClass(int i) {
    this->i=i;
}

defaultClass::~defaultClass() {
}
```

* defaultClass.h

```
#ifndef DEFAULTCLASS_H_
#define DEFAULTCLASS_H_
class defaultClass {
private:
    int i;
public:
    defaultClass(int i);
    virtual ~defaultClass();
};
#endif /* DEFAULTCLASS_H_ */
```

- Why does this work?

//why does this work

```
defaultClass d11(1); //1 arg constructor
defaultClass d22(d11); //copy constructor
defaultClass d33(2);
d33 = d11; //assignment operator
```

Where is the copy constructor and assignment operator? I did not write it.

- Compiler did.
- It will invisibly write copy constructor, assignment operator destructor for you and others if needed
- Does 'shallow' copy (variable to variable)
 1. Fine if class has no dynamically allocated memory
 2. all member variables know how to make copy of themselves
- What if you have dynamic data, or ignorant variables?
 - Show demo (Class project - class.cpp)
- Need a 'deep' copy (dynamic mem to new dynamic mem)
- Must write these 3 functions if have dynamic member vars
 - Copy constructor
 - Assignment operator
 - Destructor

Copy Constructor

Objects with dynamic memory

- A special constructor that is used to make a copy of an existing instance

```
//copy constructor  
HoldsDynamicData(const HoldsDynamicData& other);
```

- Where is it used?
 - Initializer lists (Constructing a new instance from another)
 - Pass by value to a function
 - **Show demo (Class project - class.cpp – copy constructor)**
- **Must** write yourself if object has dynamically allocated members, or object members that cannot make a copy of themselves.
- Otherwise let compiler handle it

Assignment Operator

Objects with dynamic memory

- One of many operators

```
//assignment operator  
HoldsDynamicData & operator= (const HoldsDynamicData & other);
```

- Used with =
- Show demo (Class project - class.cpp - HoldDynamicData)
- **Must** write yourself if object has dynamically allocated members, or object members that cannot make a copy of themselves.
- Otherwise let compiler handle it

Don't want to implement?

- If you don't want to implement copy constructor or assignment operator (or others as well)
- Why not just not write them?
 - Because compiler will if you don't
- Solution: **Mark them as private**
- But friend functions, and class methods can still access
- Solution: Do not define them, then they are declared so compiler will not generate them, but if you try to call them anywhere, You get a linker error.

Automatic memory Management

- RAII (Resource Acquisition Is Initialization)
- Object manages dynamic memory allocation
- As well as deletion, (you cant lose)

```
RAII::RAII(const char *ps) {
    if (ps) {
        int len = strlen(ps) + SPACE_FOR_SLASH0;    //how much?
        pmyString = new char[len];                  //allocate it
        strncpy(pmyString, ps, len);                //copy, include '/0'
    }
    else
    {
        pmyString = new char[1];
        *pmyString = '\0';
    }
}

RAII::~~RAII() {
    delete[] pmyString;
}
```


Friends

- Break encapsulation (but not as bad as getters and setters)
- .h file
 - Friend in class
 - Friend declaration
- .cpp file
 - Friend func definition
- See `Class_Stuff`

Classes – what happens before read()?

- Object members are initialized to default state.
- How to verify that they have valid data?
- Add isValid() function

.h file

```
class StudentInfo {

private:
    std::string name;
    double midterm;
    double final;
    bool bIsValid;

public:
    //default constructor
    StudentInfo();           // construct an empty
    bool isValid() const {return bIsValid;}
    void read():             //initialize name midt
```

.cpp file

```
void StudentInfo::read()
{
    cout<<"enter name";
    cin>>name;

    cout<<"enter midterm";
    cin>>midterm;

    cout<<"enter final";
    cin>>final;
    bIsValid = true;
}

StudentInfo::StudentInfo(): midterm(0), final(0), bIsValid(false) {
```

Summary

- When your object holds dynamic data – YOU MUST IMPLEMENT
 - Destructor
 - Copy constructor
 - Assignment operator (one of many operators)