CS 32 Week 2
Discussion 1I

UCLA CS

Srinath

Some of the slides are taken from Yiyou Chen

Clarification : Circular Dependency

```
80 "Characters.h"
81 #ifndef Characters_INCLUDED
82 #define Characters_INCLUDED
83 #include <cstring>
84 class Game;
85 using namespace std;
86 class Characters {
87   public:
88     Characters(double x, double y);
89   private:
90     double m_x;
91     double m_y;
92     string m_name;
93     Game m_game;
94 };
95 #endif
```

```
 98 "Game.h"
 99 #ifndef Game_INCLUDED
100 #define Game_INCLUDED
101 #include <cstring>
102 #include <iostream>
103 class Characters;
104 using namespace std;
105 class Game {
106   public:
107     Game(double size, double x, double y);
108   private:
109     Characters* m_character;
110     double m_size;
111 };
112 #endif
113
114 "main.cpp"
115 #include "Game.h"
116 #include "Characters.h"
```

Will it compile?

Clarification : Circular Dependency

```
"Characters.h"
#ifndef Characters_INCLUDED
#define Characters_INCLUDED
#include <cstring>
class Game;
using namespace std;
class Characters {
  public:
    Characters(double x, double y);
  private:
    double m_x;
    double m_y;
    string m_name;
    Game m_game;
};
#endif
```

```
"Game.h"
#ifndef Game_INCLUDED
#define Game_INCLUDED
#include <cstring>
#include <iostream>
class Characters;
using namespace std;
class Game {
  public:
    Game(double size, double x, double y);
  private:
    Characters* m_character;
    double m_size;
};
#endif

"main.cpp"
#include "Game.h"
#include "Characters.h"
```

Will it compile?

In this case, Yes!

Clarification : Circular Dependency

```
80 "Characters.h"
81 #ifndef Characters_INCLUDED
82 #define Characters_INCLUDED
83 #include <cstring>
84 class Game;
85 using namespace std;
86 class Characters {
87   public:
88     Characters(double x, double y);
89   private:
90     double m_x;
91     double m_y;
92     string m_name;
93     Game m_game;
94 };
95 #endif
```

```
 98 "Game.h"
 99 #ifndef Game_INCLUDED
100 #define Game_INCLUDED
101 #include <cstring>
102 #include <iostream>
103 class Characters;
104 using namespace std;
105 class Game {
106   public:
107     Game(double size, double x, double y);
108   private:
109     Characters* m_character;
110     double m_size;
111 };
112 #endif
113
114 "main.cpp"
115 #include "Game.h"
116 #include "Characters.h"
```

Will it compile?

However, it will not compile if main.cpp has the following code instead(notice the reverse ordering of header files).

“main.cpp”
#include “Characters.h”
#include “Game.h”

Better ways of Testing

Better ways of Testing

- Using “nano” to edit file on server

Better ways of Testing

- Using “nano” to edit file on server
- Open remote server files on your local and modify
  - Using sshfs
  - https://sbgrid.org/corewiki/faq-sshfs.md
  - Command:
    - sshfs yourUsername@cs32.seas.ucla.edu: yourLocalFolderName
    - Enter the password and your local directory will mounted to the remote one.

Topics

- Destructor
- Copy Constructor
- Assignment Operator

Destructor

Destructor

Destructor

Destructor

Why do we need a destructor?

- To free up resources used by our class
- Our code should not have Memory leaks, dangling pointers

Destructor : How to define?

```
class String{
    public:
        int m_len;
        cher* m_text;
    String();
    ~String();
}

String::String(){
    // constructor code
    ...
    m_text = new char[m_len+1];
    ...
}

String::~String(){
    // Destructor code
    delete [] m_text;
}
```

Destructor : Definition

What if we don’t define one?

Destructor : Definition

What if we don’t define one?

- A default destructor with empty body is created

Order of Destruction

Reverse the order of construction:

Order of Destruction

Reverse the order of construction:

1. Run body of the destructor
2. Data members (class: default destructor) are destructed in reverse order.
3. ---

Order of Destruction

```
class Geometry{
    public:
        Circle circle;
        Triangle triangle;
        Rectangle rectangle;
}
```

In which order the default destructors are called?

Order of Destruction

```
class Geometry{
    public:
        Circle circle;
        Triangle triangle;
        Rectangle rectangle;
}
```

In which member order the default destructors are called?

1. rectangle
2. triangle
3. circle

Destructor

When is a destructor called?

Destructor

When is a destructor called?

1. When object goes out of scope

Destructor

When is a destructor called?

1. When object goes out of scope

```
1.  Void h(...){
2.      ….
3.      Rectangle rectangle(width, height);
4.      ….
5.  }
```

Destructor

When is a destructor called?

1. When object goes out of scope

```
1.  Void h(...){
2.      ….
3.      {
4.          ……//some code
5.          Rectangle rectangle(width, height);
6.          ….. //some code
7.      }
8.      ….//some other code
9.  }
```

Destructor

When is a destructor called?

1. When object goes out of scope

```
1.  Void h(...){
2.      Rectangle rectangles[100];
3.  }
```

How many times destructor is called?

Destructor

When is a destructor called?

1. When object goes out of scope

```
1. Void h(...){
2.     Rectangle rectangles[100];
3. }
```

How many times destructor is called?

- 100 times (1 for each object)

Destructor

When is a destructor called?

1. When object goes out of scope
2. When “delete” is called

Destructor

When is a destructor called?

1. When object goes out of scope
2. When “delete” is called

```
Void h(){
        Circle* c = new Circle(cx, cy, r);
        delete c;
}
```

Destructor

When is a destructor called?

1. When object goes out of scope
2. When “delete” is called

```
Void h(){
        Circle* circles = new Circle[101];
        delete [] circles;
}
```

Destructor

When is a destructor called?

1. When object goes out of scope
2. When “delete” is called

Will this call destructor on out of scope?

```
Void h(){
        /….
        Rectangle* rectangle = new Rectangle(w, h);
        /…..
}
```

Destructor

When is a destructor called?

1. When object goes out of scope
2. When “delete” is called

Will this call destructor on out of scope? - NO

```
Void h(){
        /….
        Rectangle* rectangle = new Rectangle(w, h);
        /…..
}
```

Copy Constructor

Copy Constructor

Pass by Reference vs Pass by Value

Copy Constructor

```
145 class User
146 {
147   public:
148     User(const double* tasks, const int& len);
149     User(const User& other);
150     User& operator=(const User& rhs);
151     void swap(User& other);
152     ~User();
153   private:
154     string m_name;
155     int m_age;
156     int m_len;
157     double* m_tasks;
158 };
159 User::User(const User& other)
160 {
161   m_len = other.m_len;
162   m_age = other.m_age;
163   m_name = other.m_name;
164   m_tasks = new double[m_len];
165   for (int i = 0; i < m_len; ++i)
166     m_tasks[i] = other.m_tasks[i];
167 }
```

Most of the time, a copy constructor passes by constant reference.

```
211 User b(...);
212 User a(b);
213 User a = b;
```

Copy Constructor

When is it invoked?

Copy Constructor

When is it invoked?

- String a(b)
- String a = b
- Pass by value : void area(Circle c)
- Return by value : return circle

Assignment Operator

Assignment Operator

By default, "=" copies everything, which may cause memory leak and malfunctioning when the class has pointers as its data member. We define "=" as a member function.

```
30 class User
31 {
32   public:
33     User(const double* tasks, const int& len);
34     User(const User& other);
35     User& operator=(const User& rhs);
36     ~User();
37   private:
38     string m_name;
39     int m_age;
40     int m_len;
41     double* m_tasks;
42 }
43
44 User& User::operator=(const User& rhs) {
45   //check if assign u to u: u = u
46   if (this != & rhs) {
47     m_age = rhs.m_age;
48     m_name = rhs.m_name;
49     m_len = rhs.m_len;
50     delete [] m_tasks;
51     m_tasks = new double[m_len];
52     for (int i = 0; i < m_len; ++i) {
53       m_tasks[i] = rhs.m_tasks[i];
54     }
55   }
56 }
```

Aliasing: two different variables have the same reference (e.g. u = u). Always be cautious of aliasing!

This is the traditional way. Not widely used. Why?

```
216 User a(...);
217 User b(...);
218 a = b;
```

Assignment Operator

```
173 class User
174 {
175   public:
176     User(const double* tasks, const int& len); //constructor
177     User(const User& other); //copy constructor
178     User& operator=(const User& rhs); //assignment operator
179     void swap(User& other); //swap
180     ~User();
181   private:
182     string m_name;
183     int m_age;
184     int m_len;
185     double* m_tasks;
186 };
187 void User::swap(User& other) {
188   std::swap(m_name, other.m_name);
189   std::swap(m_age, other.m_age);
190   std::swap(other.m_tasks, m_tasks);
191   std::swap(m_len, other.m_len);
192 }
193 User& User::operator=(const User& rhs) {
194   //check if assign u to u: u=u
195   if (this != &rhs) {
196     User temp(rhs); //copy
197     swap(temp);
198   }
199   return *this;
200 }
```

This is the modern way to assign.
It makes sure there's enough resource for assignment by creating a copy of the rhs and swap it with lhs.

```
216 User a(...);
217 User b(...);
218 a = b;
```