

CS 32 Week 2

Discussion 2C

UCLA CS
Yiyu Chen

About us

TA: Yiyou Chen. Nickname: Gerry. Email: gerry99@ucla.edu.

Discussion Zoom: see Bruin Learn.

Discussion location (if in person): Rolfe 3126.

Office hours: TW 8:30-9:30 am, R 2:30-3:30 pm, F 4:30-5:30 pm.

OH Zoom: see Bruin Learn.

LA: Katie Chang

Email: katiechang@ucla.edu

OH: Monday 1:30-3:30

 lcpc interview workshops on Thurs 6:00-8:00 

Icebreaking? . !

- Preferred name?
- Year?
- Where from?
- Favorite programming language?
- Favorite season of the year?

Topics

- Order of Construction and Destruction, Const Member Functions for class.
- Compiling and Linking, #include guard, Circular Dependency.
- Copy Constructor and Assignment Operator.

Order of Construction

1. -----
2. Initialize the data members (built-in: uninitialized; class: default constructor) in order
3. Body of the constructor

Order of Construction

```
1 class Characters {
2     public:
3         Characters(double x, double y, string name);
4     private:
5         double m_x;
6         double m_y;
7         string m_name;
8 };
9
10 Characters::Characters(double x, double y, string name) {
11     m_x = x;
12     m_y = y;
13     m_name = name;
14 }
15
16 class Game {
17     public:
18         Game(const double& size, const double& x, const double& y, const string& name);
19     private:
20         Characters m_character;
21         double m_size;
22 };
23
24 Game::Game(const double& size, const double& x, const double& y, const string& name) {
25     m_character = Characters(x, y, name);
26     m_size = size;
27 }
```

Will it compile?

Order of Construction

```
1 class Characters {
2     public:
3         Characters(double x, double y, string name);
4     private:
5         double m_x;
6         double m_y;
7         string m_name;
8 };
9
10 Characters::Characters(double x, double y, string name) {
11     m_x = x;
12     m_y = y;
13     m_name = name;
14 }
15
16 class Game {
17     public:
18         Game(const double& size, const double& x, const double& y, const string& name);
19     private:
20         Characters m_character;
21         double m_size;
22 };
23
24 Game::Game(const double& size, const double& x, const double& y, const string& name) {
25     m_character = Characters(x, y, name);
26     m_size = size;
27 }
```

Wrong!
Characters has no default constructor
Characters::Characters();

Order of Construction

```
3 class Characters {
4     public:
5         Characters(double x, double y, string name);
6     private:
7         double m_x;
8         double m_y;
9         string m_name;
10 };
11
12 Characters::Characters(double x, double y, string name) {
13     m_x = x;
14     m_y = y;
15     m_name = name;
16 }
17
18 class Game {
19     public:
20         Game(const double& size, const double& x, const double& y, const string& name);
21     private:
22         Characters m_character;
23         double m_size;
24 };
25
26 Game::Game(const double& size, const double& x, const double& y, const string& name)
27     : m_character(x, y, name)
28 {
29     m_size = size;
30 }
```

Correct!

Order of Construction

```
1 class Characters {
2     public:
3         Characters(double x, double y, string name);
4     private:
5         double m_x;
6         double m_y;
7         string m_name;
8 };
9
10 Characters::Characters(double x, double y, string name) {
11     m_x = x;
12     m_y = y;
13     m_name = name;
14 }
15
16 class Game {
17     public:
18         Game(const double& size, const double& x, const double& y, const string& name);
19     private:
20         Characters* m_character;
21         double m_size;
22 };
23
24 Game::Game(const double& size, const double& x, const double& y, const string& name)
25 {
26     m_character = new Characters(x, y, name);
27     m_size = size;
28 }
```

Correct!

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

```
32 class User
33 {
34     public:
35         User(const double* tasks, const int& len);
36         ~User();
37     private:
38         string m_name;
39         int m_age;
40         double* m_tasks;
41         int m_len;
42 }
43 User::User(const double* tasks, const int len) {
44     m_len = len;
45     m_tasks = new double[len];
46     for (int i = 0; i < len; ++i) {
47         m_tasks[i] = tasks[i];
48     }
49 }
50 User::~~User() {
51     delete [] m_tasks;
52 }
```

Const Member Functions

```
56 class Characters {
57     public:
58         Characters(double x, double y, string name);
59         void Move(const double& movex, const double& movey);
60     private:
61         double m_x;
62         double m_y;
63         string m_name;
64 };
65 Characters::Characters(double x, double y, string name) {
66     m_x = x;
67     m_y = y;
68     m_name = name;
69 }
70
71 void Characters::Move(const double& movex, const double& movey) {
72     cout << "hi" << endl;
73 }
74
75 void Play(const Characters* character, const double& movex, const double& movey) {
76     character->Move(movex, movey);
77 }
```

Will it compile?

Const Member Functions

```
56 class Characters {
57     public:
58         Characters(double x, double y, string name);
59         void Move(const double& movex, const double& movey);
60     private:
61         double m_x;
62         double m_y;
63         string m_name;
64 };
65 Characters::Characters(double x, double y, string name) {
66     m_x = x;
67     m_y = y;
68     m_name = name;
69 }
70
71 void Characters::Move(const double& movex, const double& movey) {
72     cout << "hi" << endl;
73 }
74
75 void Play(const Characters* character, const double& movex, const double& movey) {
76     character->Move(movex, movey);
77 }
```

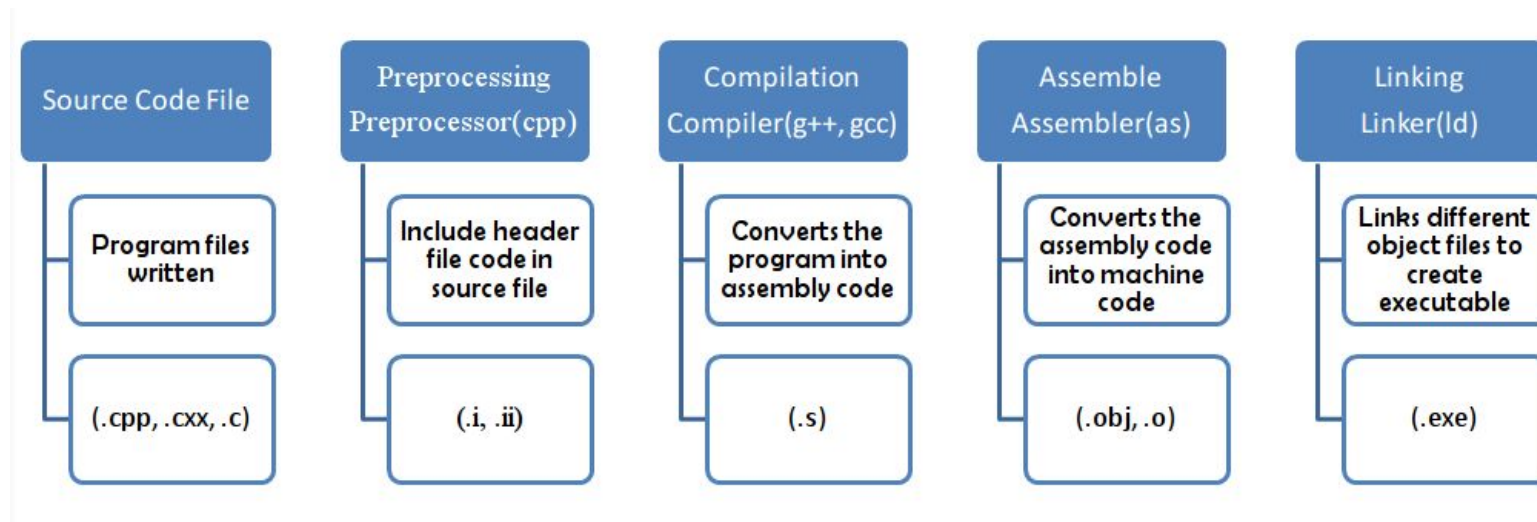
Wrong! Move is not defined to be a constant member function.

Const Member Functions

```
56 class Characters {
57     public:
58         Characters(double x, double y, string name);
59         void Move(const double& movex, const double& movey) const;
60     private:
61         double m_x;
62         double m_y;
63         string m_name;
64 };
65 Characters::Characters(double x, double y, string name) {
66     m_x = x;
67     m_y = y;
68     m_name = name;
69 }
70
71 void Characters::Move(const double& movex, const double& movey) const {
72     cout << "hi" << endl;
73 }
74
75 void Play(const Characters* character, const double& movex, const double& movey) {
76     character->Move(movex, movey);
77 }
```

Correct!

Compiling and Linking



#include guard

To make sure each header file is included once for each source file.

```
"XXX.h"  
#ifndef XXX_INCLUDED  
#define XXX_INCLUDED  
class XXX{  
    ....  
};  
...  
#endif
```


Circular Dependency

```
80 "Characters.h"
81 #ifndef Characters_INCLUDED
82 #define Characters_INCLUDED
83 #include <cstring>
84 #include "Game.h"
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 #include "Characters.h"
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"
117
```

Will it compile?

Circular Dependency

```
80 "Characters.h"
81 #ifndef Characters_INCLUDED
82 #define Characters_INCLUDED
83 #include <cstring>
84 #include "Game.h"
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 #include "Characters.h"
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"
117
```

Wrong!

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?

Circular Dependency

```
80 "Characters.h"
81 #ifndef Characters_INCLUDED
82 #define Characters_INCLUDED
83 #include <cstring>
84 #include "Game.h"
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"
```

Correct!

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;
```

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;
```