Constructors and Destructors

Today

- Investigate when constructor & destructors are called
  - In different type of objects
    - local variable, array, dynamic allocated object
  - In compound data, where an object is
    - a member variable of another object
    - the base class section of an children object

Local variable object

- Example class

```
6  class A
7  {
8      int id; // private by default
9  public:
10     A(int id) {
11         this->id = id; // must use this->
12         cout << "A(int) " << id << endl;
13     }
14     A(const A & a) { // alway use const
15         id = a.id;
16         cout << "A(const A &) " << id << endl;
17     }
18     ~A() { // destructor
19         cout << "~A() " << id << endl;
20     }
21 };
```

Local variable object

- Constructor: at the line of object definition
- Destructor: when the memory of the object memory is freed

```
28 int main() {
29     cout << "the first line" << endl;
30     A a1(100);
31     cout << "the last line" << endl;
32 }
```

```
the first line
A(int) 100
the last line
~A() 100
```

Local variable object

- Constructed first, destructed last
  - Why?

```
28 int main() {
29     cout << "the first line" << endl;
30     A a1(100);
31     A a2(200);
32     cout << "the last line" << endl;
33 }
```

```
A(int) 100
A(int) 200
the last line
~A() 200
~A() 100
```

Local variable object

- If an object is not constructed, it will not be destructed

```
35 int main() {
36     int id;
37     cin >> id;
38     if (id < 0) return 0;
39     A a(id);
40 }
```

```
-100
```

```
42 int main() {
43     int id;
44     cin >> id;
45     if (id > 0) {
46         A a(id);
47     }
48 }
```

```
100
A(int) 100
~A() 100
```

Local variable object

- Object in blocks
  - Optimization: memory reused

```
50 int main() {
51     A a1(100);
52     {
53         A a2(200);
54         cout << &a2 << endl;
55     }
56     {
57         A a3(300);
58         cout << &a3 << endl;
59     }
60 }
```

```
[home@me Desktop % g++ -O0 7.cpp
[home@me Desktop % ./a.out
A(int) 100
A(int) 200
0x7ffee3fa5bc0
~A() 200
A(int) 300
0x7ffee3fa5bb0
~A() 300
~A() 100
```

```
[home@me Desktop % g++ -O2 7.cpp
[home@me Desktop % ./a.out
A(int) 100
A(int) 200
0x7ffee66babb0
~A() 200
A(int) 300
0x7ffee66babb0
~A() 300
~A() 100
```

Local variable object

- Object in loops

```
home@me Desktop % g++ -O0 7.cpp
home@me Desktop % ./a.out
A(int) 100
A(int) 200
0x7ffee380ebc0
~A() 200
A(int) 200
0x7ffee380ebc0
~A() 200
~A() 100
```

```
62 int main() {
63     A a1(100);
64     for (int i = 0; i < 2; ++ i) {
65         A a2(200);
66         cout << &a2 << endl;
67     }
68 }
```

Objects in arrays

- Example class

```
4  int next_id = 0;
5  class A
6  {
7      int id;
8  public:
9      A() {
10         id = ++ next_id;
11         cout << "A() " << id << endl;
12     }
13     A(int x) {
14         id = x;
15         cout << "A(int) " << id << endl;
16     }
17     A(const A & a) {
18         id = a.id;
19         cout << "A(const A &) " << id << endl;
20     }
21     ~A() {
22         cout << "~A() " << id << endl;
23     }
24 };
```

Objects in arrays

```
94 int main() {
95     A array[3];
96 }
```

```
A() 1
A() 2
A() 3
~A() 3
~A() 2
~A() 1
```

```
94 int main() {
95     A * array1 = new A[2];
96     delete [] array1;
97     A * array2 = new A[2];
98 }
```

```
A() 1
A() 2
~A() 2
~A() 1
A() 3
A() 4
```

Objects in arrays

- 返回值优化
  - 如果一个临时对象仅仅被用来初始化另一个同类型的对象，那么则可以省掉这种临时对象（而直接构造目标对象）

```
104 int main() {
105     A a = A(100); // A a(A(100));
106 }
```

```
home@me Desktop % g++ 7.cpp
home@me Desktop % ./a.out
A(int) 100
~A() 100
```

```
home@me Desktop % g++ -fno-elide-constructors 7.cpp
home@me Desktop % ./a.out
A(int) 100
A(const A &) 100
~A() 100
~A() 100
```

Objects in arrays

- 返回值优化

```
4  int next_id = 0;
5  class A
6  {
7      int id;
8  public:
9      A() {
10         id = ++ next_id;
11         cout << "A() " << id << endl;
12     }
13     A(int x) {
14         id = x;
15         cout << "A(int) " << id << endl;
16     }
17     A(const A & a) {
18         id = 1000 + a.id;
19         cout << "A(const A &) " << id << endl;
20     }
21     ~A() {
22         cout << "~A() " << id << endl;
23     }
24 };
```

Objects in arrays

- 返回值优化
  - 如果一个临时对象仅仅被用来初始化另一个同类型的对象，那么则可以省掉这种临时对象（而直接构造目标对象）

```
26 A fun(A x) {
27     cout << "fun" << endl;
28     return A(200);
29 }
30 int main() {
31     A y = fun(A(100));
32     cout << "end" << endl;
33 }
```

```
home@me Desktop % g++ -O0 test.cpp
home@me Desktop % ./a.out
A(int) 100
fun
A(int) 200
~A() 100
end
~A() 200
```

```
home@me Desktop % g++ -fno-elide-constructors test.cpp
home@me Desktop % ./a.out
A(int) 100
A(const A &) 1100
fun
A(int) 200
A(const A &) 1200
~A() 200
A(const A &) 2200
~A() 1200
~A() 1100
~A() 100
end
~A() 2200
```

Objects in arrays

- There will be an error if const is removed, even if the constructor is

```
84  A(const A & b) {
85      id = b.id;
86      cout << "A(const A &) " << id << endl;
87  }
```

```
84  A(A & b) {
85      id = b.id;
86      cout << "A(const A &) " << id << endl;
87  }
```

```
104 int main() {
105     A a = A(100); // A a(A(100));
106 }
```

```
7.cpp:105:4: error: no matching constructor for initialization of 'A'
        A a = A(100); // A a(A(100));
          ^   ~~~~~~
```

```
7.cpp:84:2: note: candidate constructor not viable: expects an l-value for 1st
      argument
    A(A & b) {
    ^
```

Objects in arrays

- How to make an array if class A has no default constructor ?

```
104 int main() {
105     A a[3];
106 }
```

```
7.cpp:105:4: error: no matching constructor for initialization of 'A [3]'
        A a[3];
          ^
```

Objects in arrays

- How to make an array if class A has no default constructor ?

```
104 int main() {
105     A a[2] = { A(100), A(200) };
106 }
```

```
home@me Desktop % g++ 7.cpp
home@me Desktop % ./a.out
A(int) 100
A(int) 200
~A() 200
~A() 100
```

```
home@me Desktop % g++ -fno-elide-constructors 7.cpp
home@me Desktop % ./a.out
A(int) 100
A(const A &) 100
A(int) 200
A(const A &) 200
~A() 200
~A() 100
~A() 200
~A() 100
```

Object as a member variable

- How to select a constructor for a member variable object ?

```
class B
{
    A a(100);
public:
    B() {}
};
```

```
7.cpp:237:6: error: expected parameter declarator
        A a(100);
             ^
```

```
class B
{
    A a;
public:
    B() {
        a(100);
    }
};
```

```
7.cpp:240:3: error: type 'A' does not provide a call operator
                a(100);
                ^
```

Object as a member variable

```
120 class C
121 {
122     A a;
123     int id;
124 public:
125     C(int id) {
126         this->id = id;
127         cout << "C(int) " << id << endl;
128     }
129     C(const C & b) {
130         id = b.id;
131         cout << "C(const C &) " << id << endl;
132     }
133     ~C() {
134         cout << "~C() " << id << endl;
135     }
136 };
```

```
138 int main() {
139     C c(100);
140 }
```

```
A() 1
C(int) 100
~C() 100
~A() 1
```

Object as a member variable

- What if the member has no default constructor or we do not what to use the default constructor ?

```
120 class C
121 {
122     A a;
123     int id;
124 public:
125     C(int id) : a(200) {
126         this->id = id;
127         cout << "C(int) " << id << endl;
128     }
129     C(const C & b) : a(b.a) {
130         id = b.id;
131         cout << "C(const C &) " << id << endl;
132     }
133     ~C() {
134         cout << "~C() " << id << endl;
135     }
136 };
```

```
138 int main() {
139     C c1(100);
140 }
```

```
A(int) 200
C(int) 100
~C() 100
~A() 200
```

```
138 int main() {
139     C c1(100);
140     C c2(c1);
141 }
```

```
A(int) 200
C(int) 100
A(const A &) 200
C(const C &) 100
~C() 100
~A() 200
~C() 100
~A() 200
```

Object as a member variable

```
120 class C
121 {
122     A a1;
123     A a2;
124     int id;
125 public:
126     C(int id) : a2(200), a1(300) {
127         this->id = id;
128         cout << "C(int) " << id << endl;
129     }
130     C(const C & b) : a1(b.a1), a2(b.a2) {
131         id = b.id;
132         cout << "C(const C &) " << id << endl;
133     }
134     ~C() {
135         cout << "~C() " << id << endl;
136     }
137 };
```

This list is to specifig the constructor use by each member, order in the list does not matter.

```
139 int main() {
140     C c1(100);
141 }
```

```
A(int) 300
A(int) 200
C(int) 100
~C() 100
~A() 200
~A() 300
```

Object as a member variable

Each constructor is free to choose its members' constructors.

```
A() 1
A(int) 200
C(int) 100
~C() 100
~A() 200
~A() 1
```

```
120 class C
121 {
122     A a1;
123     A a2;
124     int id;
125 public:
126     C(int id) : a2(200), a1() {
127         this->id = id;
128         cout << "C(int) " << id << endl;
129     }
130     C(const C & b) : a1(b.a1), a2(b.a2) {
131         id = b.id;
132         cout << "C(const C &) " << id << endl;
133     }
134     ~C() {
135         cout << "~C() " << id << endl;
136     }
137 };
```

By omitting, the default constructor is chosen

```
120 class C
121 {
122     A a1;
123     A a2;
124     int id;
125 public:
126     C(int id) : a2(200) {
127         this->id = id;
128         cout << "C(int) " << id << endl;
129     }
130     C(const C & b) : a1(b.a1), a2(b.a2) {
131         id = b.id;
132         cout << "C(const C &) " << id << endl;
133     }
134     ~C() {
135         cout << "~C() " << id << endl;
136     }
137 };
```

Constructor/destructor chaining

```
103 class B
104 {
105     int id;
106 public:
107     B(int id) {
108         this->id = id;
109         cout << "B(int) " << id << endl;
110     }
111     B(const B & b) {
112         id = b.id;
113         cout << "B(const B &) " << id << endl;
114     }
115     ~B() {
116         cout << "~B() " << id << endl;
117     }
118 };
```

```
120 class C : public B
121 {
122     int id;
123 public:
124     C(int id) : B(id/10) {
125         this->id = id;
126         cout << "C(int) " << id << endl;
127     }
128     C(const C & c) : B(c) { // down-casting
129         id = c.id;
130         cout << "C(const C &) " << id << endl;
131     }
132     ~C() { // calls ~B()
133         cout << "~C() " << id << endl;
134     }
135 };
```

```
137 int main() {
138     C c(100);
139 }
```

```
B(int) 10
C(int) 100
~C() 100
~B() 10
```

Constructor/destructor chaining

```
137 class D : public C
138 {
139     int id;
140 public:
141     D(int id) : C(id/10) {
142         this->id = id;
143         cout << "D(int) " << id << endl;
144     }
145     D(const D & d) : C(d) { // down-casting
146         id = d.id;
147         cout << "D(const D &) " << id << endl;
148     }
149     ~D() { // calls ~C()
150         cout << "~D() " << id << endl;
151     }
152 };
```

```
155 int main() {
156     D d(100);
157 }
```

```
B(int) 1
C(int) 10
D(int) 100
~D() 100
~C() 10
~B() 1
```

Auto-composed member functions

```
168 class C : public B
169 {
170     int id;
171     A a;
172 public:
173     C() : B(), a() {
174         id = 0;
175         cout << "C() " << id << endl;
176     }
177     C(int id) : B(id/100), a(id/10) {
178         this->id = id;
179         cout << "C(int) " << id << endl;
180     }
181     C(const C & c) : B(c), a(c.a) {
182         cout << "C(const C &) " << id << endl;
183     }
184     ~C() { // calls ~B(), a.~A()
185         cout << "~C() " << id << endl;
186     }
187 };
```

```
189 int main() {
190     C c(100);
191 }
```

```
B(int) 1
A(int) 10
C(int) 100
~C() 100
~A() 10
~B() 1
```

Auto-composed member functions

```
196 class A
197 {
198     ...
199     A & operator = (const A & a) {
200         id = a.id;
201         cout << "A::operator= " << id << endl;
202         return *this;
203     }
204 };
205
206 class B
207 {
208     ...
209     B & operator = (const B & b) {
210         id = b.id;
211         cout << "B::operator= " << id << endl;
212         return *this;
213     }
214 };
```

```
216 class C : public B
217 {
218     ...
219     C & operator = (const C & c) {
220         (*this).B::operator=(c);
221         a = c.a;
222         cout << "C::operator= " << id << endl;
223         return *this;
224     }
225 };
226
227
228 int main() {
229     C c1;
230     C c2(100);
231     c1 = c2;
232 }
```

```
B(int) 0
A() 1
C() 0
B(int) 1
A(int) 10
C(int) 100
B::operator= 1
A::operator= 10
C::operator= 0
~C() 100
~A() 10
~B() 1
~C() 0
~A() 10
~B() 1
```

Auto-composed member functions

```
168 class C : public B
169 {
170     int id;
171     A a;
172 public:
173     C() : B(), a(), id() {
174     }
175     C(const C & c) : B(c), a(c.a) {
176     }
177     ~C() { // calls ~B(), a.~A()
178     }
179     C & operator = (const C & c) {
180         (*this).B::operator=(c);
181         a = c.a;
182         return *this;
183     }
184 };
```

If removed, the compiler can compose all these four functons correctly.