IT206 Data Structures Lab with OOP

Lecture 4

Rachit Chhaya

DA-IICT

May 2, 2022

Constructors in Derived Classes

- As long as base class constructors does not have arguments, derived class need not have a constructor function.

Constructors in Derived Classes

- As long as base class constructors does not have arguments, derived class need not have a constructor function.
- If any base class has a constructor with one or more arguments, mandatory for derived class to have constructor and pass arguments to the base class constructor.

Constructors in Derived Classes

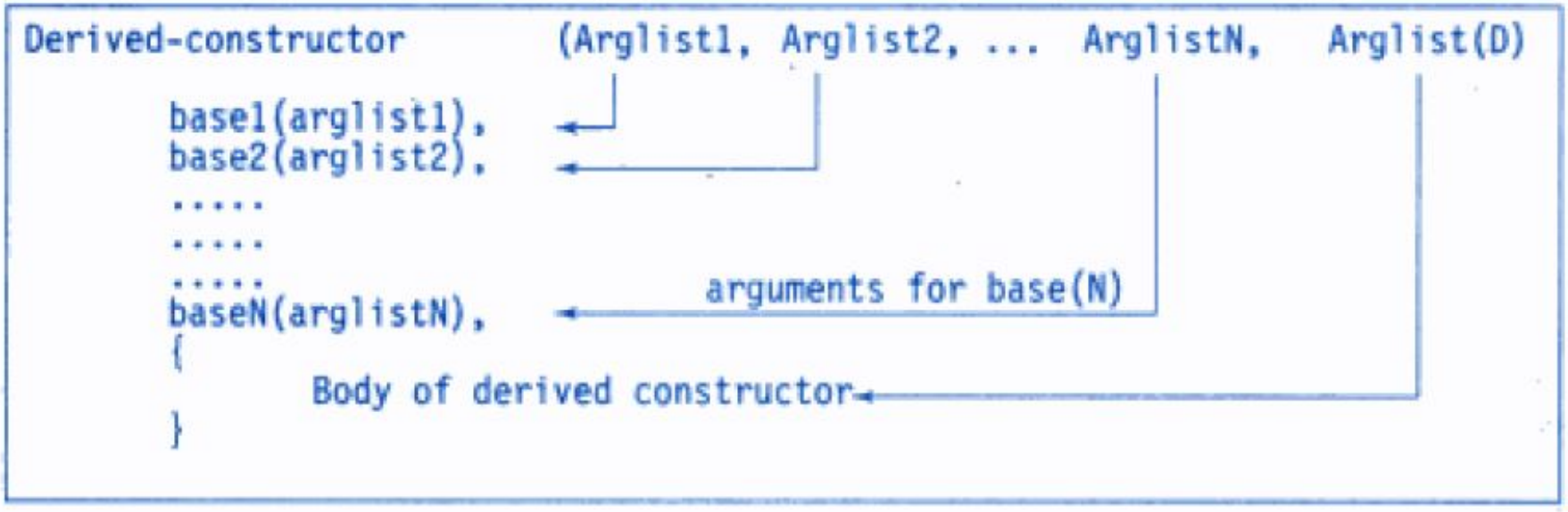
- As long as base class constructors does not have arguments, derived class need not have a constructor function.
- If any base class has a constructor with one or more arguments, mandatory for derived class to have constructor and pass arguments to the base class constructor.
- If both derived and base class have constructors, base -first , derived executed after.

- Multiple Inheritance: Base classes constructed in order in which they appear in declaration of derived class.

- Multiple Inheritance: Base classes constructed in order in which they appear in declaration of derived class.
- Multilevel Inheritance: Constructors executed in order of inheritance.

Passing Arguments using Derived Constructors

The general form of defining a derived constructor is:



Example:

```
D(int a1, int a2, float b1, float b2, int d1):
A(a1, a2),          /* call to constructor A */
B(b1, b2)          /* call to constructor B */
{
     d = d1;      // executes its own body
}
```

Another method of Initialization

```
constructor (arglist) : intialization-section
{
      assignment-section
}
```

```
class XYZ
{
      int a;
      int b;
  public:
      XYZ(int i, int j) : a(i), b(2 * j) { }
};

main()
{
      XYZ x(2, 3);
}
```

Data members are initialized in order of declaration, independent of order in the initialization list.

Ambiguity in Inheritance

```
#include <iostream>

class ClassA {
  public:
    int a;
};

class ClassB : public ClassA {
  public:
    int b;
};
```

```
class ClassC : public ClassA {
  public:
    int c;
};

class ClassD : public ClassB, public ClassC {
  public:
    int d;
};
  int main()
{
    ClassD obj;

    // obj.a = 10;                    // Statement 1, Error
    // obj.a = 100;                   // Statement 2, Error
```

```
    obj.ClassB::a = 10;  // Statement 3
    obj.ClassC::a = 100; // Statement 4

    obj.b = 20;
    obj.c = 30;
    obj.d = 40;

    cout << " a from ClassB  : " << obj.ClassB::a;
    cout << "\n a from ClassC  : " << obj.ClassC::a;

    cout << "\n b : " << obj.b;
    cout << "\n c : " << obj.c;
    cout << "\n d : " << obj.d << '\n';
    return 0;
}
```

Virtual Base Class

- Duplication of inherited members can be avoided using virtual base class.

Virtual Base Class

- Duplication of inherited members can be avoided using virtual base class.
- When a base class is specified as a virtual base, it can act as an indirect base more than once without duplication of its data members. A single copy of its data members is shared by all the base classes that use virtual base.

Virtual Base Class

- Duplication of inherited members can be avoided using virtual base class.
- When a base class is specified as a virtual base, it can act as an indirect base more than once without duplication of its data members. A single copy of its data members is shared by all the base classes that use virtual base.
- `class B : public virtual A {};`

Virtual Base Class

- Duplication of inherited members can be avoided using virtual base class.
- When a base class is specified as a virtual base, it can act as an indirect base more than once without duplication of its data members. A single copy of its data members is shared by all the base classes that use virtual base.
- `class B : public virtual A {};`
- The keywords *public* and *virtual* can be in any order.
- Constructors of virtual base class are invoked before any non-virtual base class

```
#include <iostream>

class ClassA
{
  public:
    int a;
};

class ClassB : virtual public ClassA
{
  public:
    int b;
};
```

```
  class ClassC : virtual public ClassA
{
  public:
    int c;
};

class ClassD : public ClassB, public ClassC
{
  public:
    int d;
};

int main()
{
    ClassD obj;

    obj.a = 10;         // Statement 3
    obj.a = 100;        // Statement 4
```

```
 obj.b = 20;
    obj.c = 30;
    obj.d = 40;

    cout << "\n a : " << obj.a;
    cout << "\n b : " << obj.b;
    cout << "\n c : " << obj.c;
    cout << "\n d : " << obj.d << '\n';
    return 0;
}
```

Abstract Classes

- Class not used to create objects.

Abstract Classes

- Class not used to create objects.
- Designed only to be inherited by other classes and act as base class
- More in next lecture (Pure Virtual functions)

Nesting of Classes

```
class alpha {....};
class beta {....};
class gamma
{
        alpha a;
        beta b;
        .....
};
```

```
class gamma
{
      .....
      alpha a;          // a is object of alpha
      beta b;           // b is object of beta
  public:
      gamma(arglist): a(arglist1), b(arglist2)
      {
            // constructor body
      }
};
```

Exercises

```
class Student {
        char* name;
        int rollNumber;
private:
        Student() {
                name = "AlanKay";
                rollNumber = 1025;
        }
        void setNumber(int no) {
                rollNumber = no;
        }
        int getRollNumber() {
                return rollNumber;
        }
};
```

```
class AnualTest: Student {
        int mark1, mark2;
public:
        AnualTest(int m1, int m2)
                :mark1(m1), mark2(m2) {
        }
        int getRollNumber() {
                return Student::getRollNumber();
        }
};

void main()
{
        AnualTest test1(92, 85);
        cout << test1.getRollNumber();
}
```

```
class A
{
        int i;
};

class AB: virtual A
{
        int j;
};
```

```
class AC: A, ABAC
{
        int k;
};
class ABAC: AB, AC
{
        int l;
};
void main()
{
        ABAC abac;
        cout << "sizeof ABAC:" << sizeof(abac);
}
```

```
// Test program
#include <iostream.h>

class B1
{
        int b1;
  public:
        void display();
        {
                cout << b1 <<"\n";
        }
};

class B2
{
        int b2;
  public:
        void display();
        {
                cout << b2 <<"\n";
        }
};
class D: public B1, public B2
{
        // nothing here
};
main()
{
        D d;
        d.display()
        d.B1::display();
        d.B2::display();
}
```