Data Structures and Object Oriented Programming

Lecture 17

Dr. Naveed Anwar Bhatti

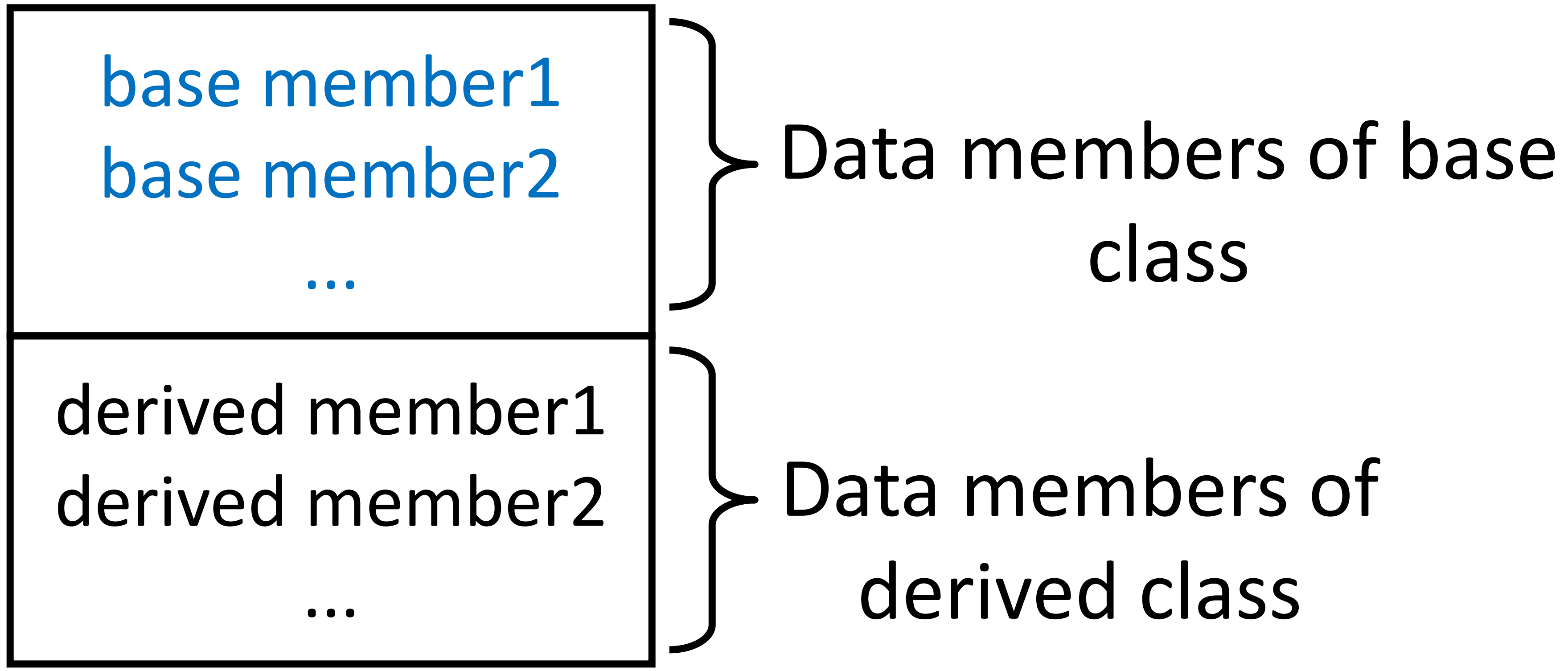
Webpage: naveedanwarbhatti.github.io

Allocation in Memory

- Every object of derived class has an anonymous object of base class

Allocation in Memory

- The object of derived class is represented in memory as follows

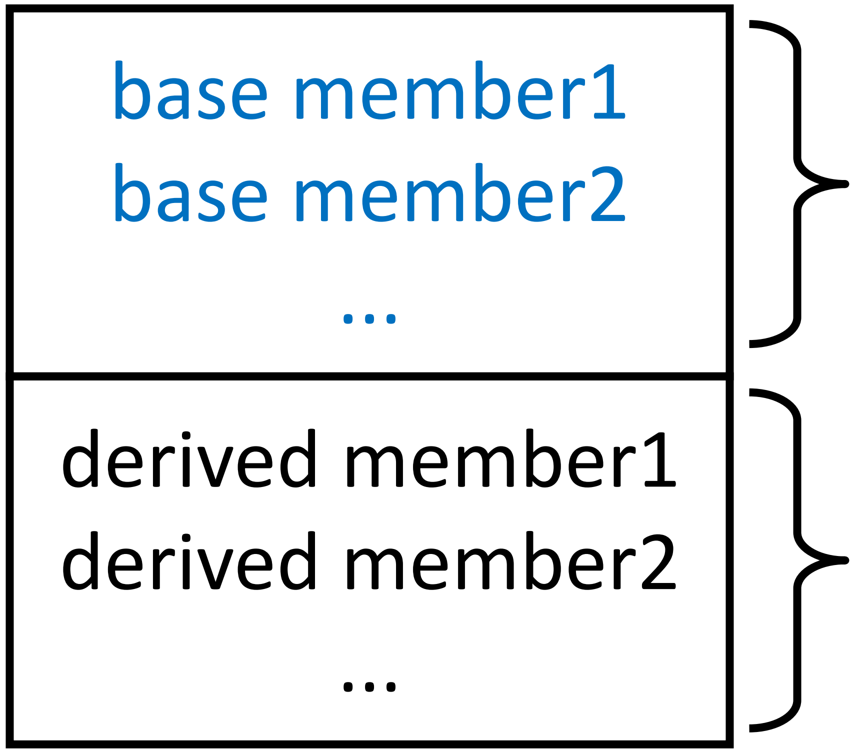


Constructors

- The anonymous object of base class must be initialized using constructor of base class

- When a derived class object is created the constructor of base class is executed before the constructor of derived class

Constructors



Base class constructor initializes the anonymous object

Derived class constructor initializes the derived class object

Example

```cpp
class Parent
{
public:
    Parent()
    {
        cout << "Parent Constructor...";
    }
};

class Child : public Parent
{
public:
    Child()
    {
        cout << "Child Constructor...";
    }
};
```

Example

```
int main()
{
    Child cobj;
    return 0;
}
```

Output:

```
Parent Constructor...
Child Constructor...
```

Constructor

- If **default constructor** of base class does not exist then the compiler will try to generate a default constructor for base class and execute it before executing constructor of derived class

Constructor

- If the user has given only an **argument based constructor** for base class, the compiler will not generate **default constructor** for base class

Example

```
class Parent
{
public:
    Parent(int i)
    {
        cout << "Parent Constructor...";
    }
};

class Child : public Parent
{
public:
    Child()
    {
        cout << "Child Constructor...";
    }
};
```

Example

```
int main()
{
    Child cobj;
    return 0;
}
```

Output:

Error

Base Class Initializer

- C++ has provided a mechanism to explicitly call a constructor of base class from derived class

- The syntax is similar to member initializer and is referred as base-class initialization

Example

```cpp
class Parent
{
public:
    Parent(int i)
    {
        cout << "Parent Constructor...";
    }
};

class Child : public Parent
{
public:
    Child(int i): Parent(i)
    {
        cout << "Child Constructor...";
    }
};
```

Base Class Initializer

- User can provide base class initializer (through constructor) and member initializer simultaneously

Example

```cpp
class Parent
{
public:
    Parent(int i)
    {
        cout << "Parent Constructor...";
    }
};

class Child : public Parent
{
    int member;
public:
    Child(int i, int x) : member(x), Parent(i)
    {
        cout << "Child Constructor...";
    }
};
```

```cpp
int main()
{
    Child rec1(1,2);

    system("pause");
    return 0;
}
```

Base Class Initializer

- The base class initializer can be written after member initializer for derived class
- The base class constructor is executed before the initialization of data members of derived class.

Initializing Members

- Derived class can only initialize members of base class using base class constructor constructors
  - Derived class can not initialize the public data member of base class using member initialization list

Example

```
class Parent
{
public:
   int member;
    Parent()
    {
        cout << "Parent Constructor...";
    }
};

class Child : public Parent
{
public:
    Child(int i): member(i)
    {
        cout << "Child Constructor...";
    }
};
```

Error

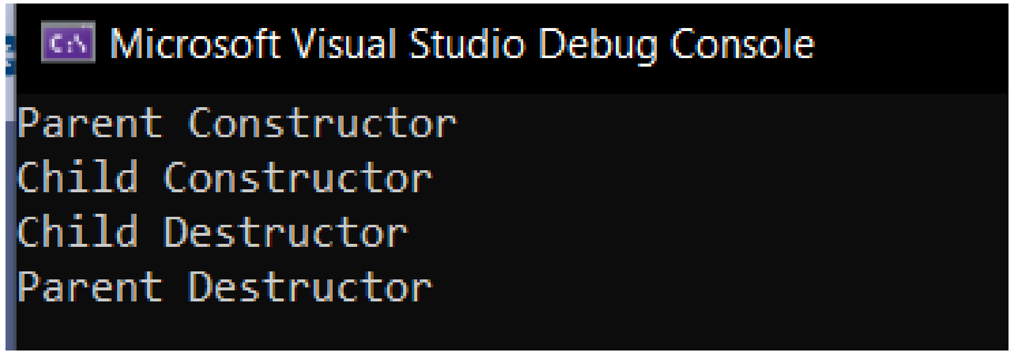
Reason:
It will be an assignment not an initialization

Destructors

- Destructors are called in reverse order of constructor called
- Derived class destructor is called before the base class destructor is called

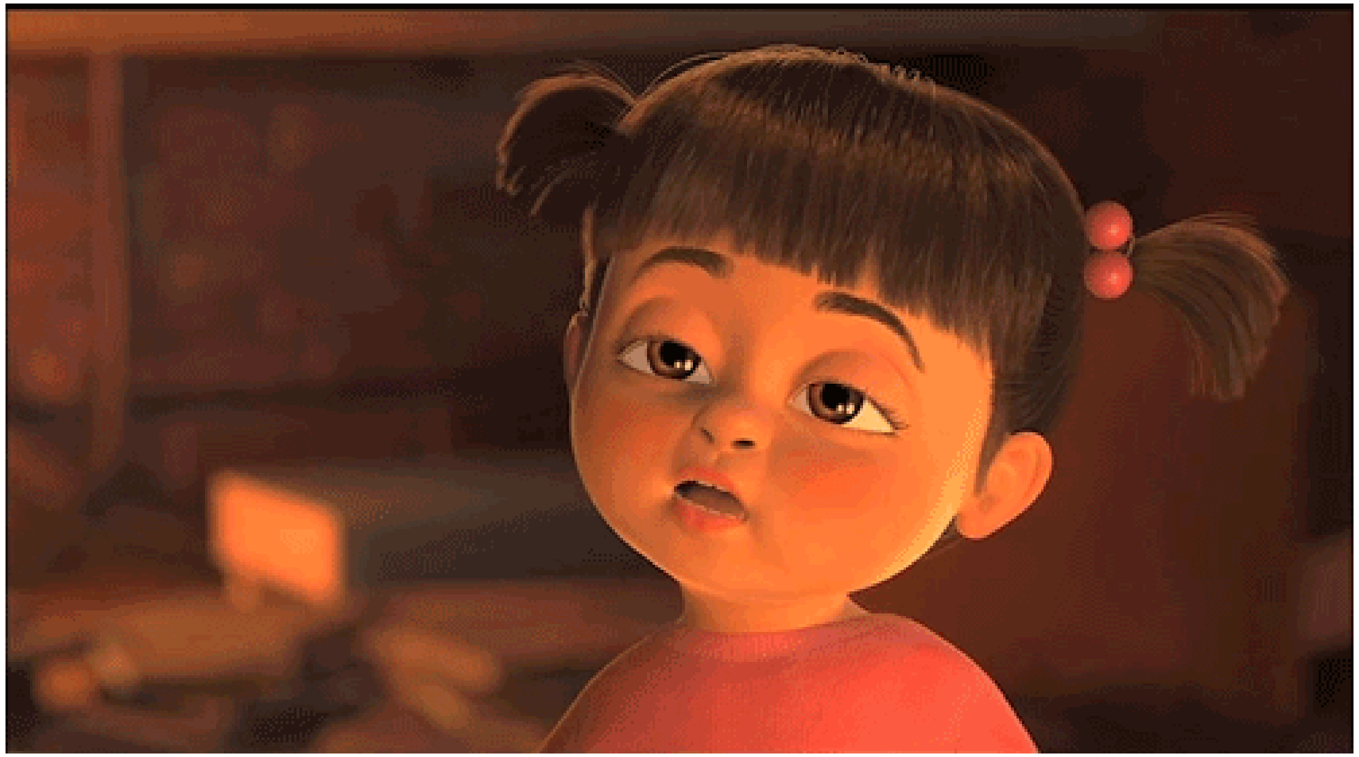
Example

```cpp
class Parent
{
public:
    Parent() { cout << "Parent Constructor"; }
    ~Parent() { cout << "Parent Destructor"; }
};

class Child : public Parent
{
public:
    Child() { cout << "Child Constructor"; }
    ~Child() { cout << "Child Destructor"; }
};

int main()
{
    Child var;
    return 0;
}
```



```
Parent Constructor
Child Constructor
Child Destructor
Parent Destructor
```

Thanks a lot



If you are taking a Nap, **wake up**.......Lecture Over