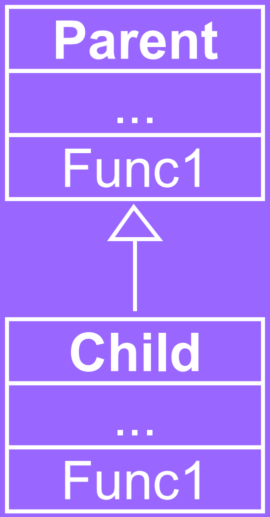
Object-Oriented Programming (OOP)
Lecture No. 25

Overriding Member Functions of Base Class

- Derived class can override the member functions of its base class
- To override a function the derived class simply provides a function with the same signature as that of its base class

Overriding



Overriding

```
class Parent {
public:
    voidFunc1();
    voidFunc1(int);
};

class Child: public Parent {
public:
    void Func1();
};
```

Overloading vs. Overriding

- Overloading is done within the scope of one class
- Overriding is done in scope of parent and child
- Overriding within the scope of single class is error due to duplicate declaration

Overriding

```
class Parent {
public:
   void      Func1();
   void      Func1();   //Error
};
```

Overriding Member Functions of Base Class

- Derive class can override member function of base class such that the working of function is totally changed

Example

```
class Person{
public:
   void Walk();
};
class ParalyzedPerson: public Person{
public:
   void Walk();
};
```

Overriding Member Functions of Base Class

- Derive class can override member function of base class such that the working of function is similar to former implementation

Example

```
class Person{
  char *name;
public:
  Person(char *=NULL);
  const char *GetName() const;
  void Print(){
     cout << "Name: " << name
            << endl;
  }
};
```

Example

```
class Student : public Person{
  char * major;
public:
  Student(char * aName, char* aMajor);


  void Print(){
      cout  <<“Name: ”<< GetName()<<endl
            << “Major:” << major<< endl;
  }
...
};
```

Example

```
int main(){
  Student a(“Ahmad”,   “Computer
  Science”);
  a.Print();
  return 0;
}
```

Output

Output:

Name: Ahmed

Major: Computer Science

Overriding Member Functions of Base Class

- Derive class can override member function of base class such that the working of function is based on former implementation

Example

```
class Student : public Person{
   char * major;
public:
   Student(char * aName, char* m);


   void Print(){
        Print();//Print of Person
        cout<<“Major:” << major <<endl;
   }
...
};
```

Example

```
int main(){
  Student a(“Ahmad”, “Computer
  Science”);
  a.Print();
  return 0;
}
```

Output

- There will be no output as the compiler will call the print of the child class from print of child class recursively
- There is no ending condition

Example

```
class Student : public Person{
   char * major;
public:
   Student(char * aName, char* m);

   void Print(){
        Person::Print();
        cout<<"Major:" << major <<endl;
   }
...
};
```

Example

```
int main(){
  Student a("Ahmad", "Computer
  Science");
  a.Print();
  return 0;
}
```

Output

Output:

```
Name: Ahmed
Major: Computer Science
```

Overriding Member Functions of Base Class

- The pointer must be used with care when working with overridden member functions

Example

```
int main(){
  Student a("Ahmad", "Computer
          Scuence");
  Student *sPtr = &a;
  sPtr->Print();

  Person *pPtr = sPtr;
  pPtr->Print();
  return 0;
}
```

Example

Output:

```
Name: Ahmed
Major: Computer Science

Name: Ahmed
```

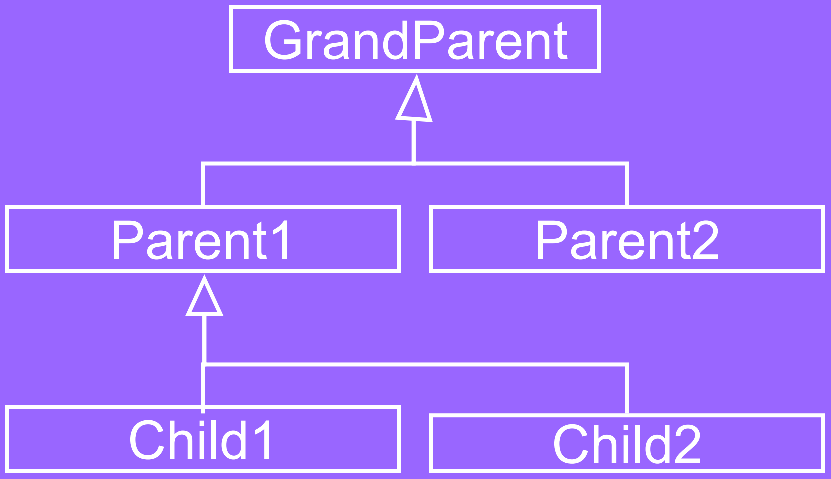
Overriding Member Functions of Base Class

- The member function is called according to static type
- The static type of pPtr is Person
- The static type of sPtr is Student

Hierarchy of Inheritance

- We represent the classes involved in inheritance relation in tree like hierarchy

Example



Direct Base Class

- A direct base class is explicitly listed in a derived class's header with a colon (:)

```
class Child1:public Parent1
...
```

Indirect Base Class

- An indirect base class is not explicitly listed in a derived class's header with a colon (:)
- It is inherited from two or more levels up the hierarchy of inheritance

```
class GrandParent{};
class Parent1:
          public GrandParent {};
class Child1:public Parent1{};
```