First Year First Semester Course
M.Tech. (CS) [Batch 2021-23]

Lecture #17

Introduction to Programming

C++: Type Conversion, Function Overriding, Logical Programming

Course Instructor:

Dr. Monidipa Das

DST-INSPIRE Faculty

Machine Intelligence Unit (MIU), Centre for Artificial Intelligence and Machine Learning (CAIML)

Indian Statistical Institute (ISI) Kolkata, India

Type Conversion

- For the built-in (basic) data types the assignment operations causes automatic type conversion between the operand as per certain rules.

- The type of data to the right of an assignment operator is automatically converted to the data type of variable on the left.

```
int val;
float p=3.14159;
val= p;
```

Can we do this for user-defined data types?

Three situations might arise:
1) Conversion from basic data type to class type
2) Conversion from class type to basic type
3) Conversion from one class type to another class type

Basic to Class Type

- Constructors may be used to perform a defacto type conversion from the argument's type to the constructor's class type

```
String :: String (char*a)
{
    len = strlen (a);
    name=new char[len+1];
    strcpy (name,a);
}
```

```
String s1, s2;
char* name1 = “Good Morning”;
char* name2 = “ STUDENTS” ;
s1 = string(name1);
s2 = name2; //implicit call to constructor 2
```

Basic to Class Type [contd.]

```
class time{
        int hours;
        int minutes;
   public:
        time()
        {cout<<"Constructor-1 called";}
        time (int t) // constructor
        {
          hours = t / 60; //t is inputted in minutes
          minutes = t % 60;
         cout<<"\nConstructor-2 called";
        }
};
```

```
int main(){
  time Tl; //object Tl created
  int period = 160;
  Tl = period; //int to class type
  return 0;
}
```

Class to Basic Type

- The constructor functions do not support conversion from a class to basic type.
- C++ allows us to define a overloaded casting operator that convert a class type data to basic type.
- **General Form**

```
operator typename ( )
{
 //Function statements.
}
```

```
vector:: operator double ( )
{
   double sum = 0 ;
   for(int i = 0; i<size; i++)
        sum = sum + v[i] * v[i ] ;
   return sqrt(sum);
}
```

`double length=double(V1);`

`or`

`double length=V1;`

One Class to another Class Type

- Obj1 = Obj2 ; //Obj1 and Obj2 are objects of different classes.
  //Can we do it?
- Can be carried out by ***either a constructor*** or a ***conversion function***.
- Which form to use, depends upon whether the type conversion is to be done in the source class or in the destination class.

Conversion Required	**Conversion takes place in**	
	Source Class	Destination class
Basic → Class	Not applicable	Constructor
Class → Basic	Casting Operator	Not Applicable
Class → Class	Casting Operator	Constructor

Example: Data Conversion

```
class stock2;
class stock1{
  int code, item; float price;
  public:
    stock1 (int a, int b, float c){
      code=a; item=b; price=c;
    }
    void disp( ){
      cout<<" code: "<<code <<"\n";
      cout<<" Items: "<<item <<"\n";
      cout<<" Price per item: Rs . "<<price <<"\n";
    }
    int getcode( ) {return code; }
    int getitem( ){return item; }
    int getprice( ) {return price;}
    operator float( ){return ( item*price );}
};
```

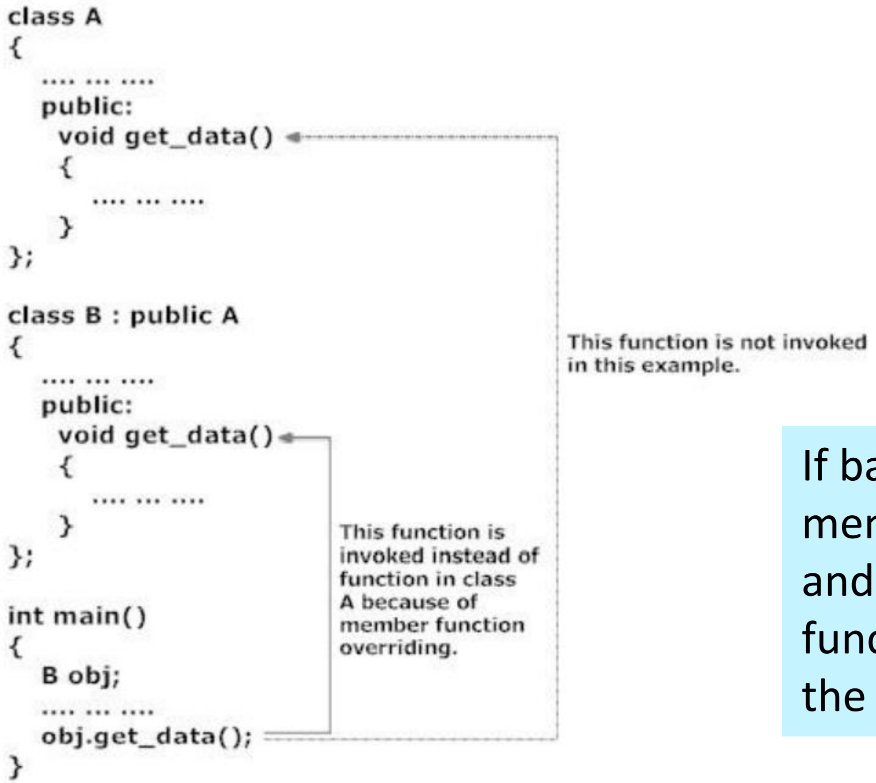
Example: Data Conversion

```
class stock2
{
    int code;
    float val;
    public:
      stock2(){code=0; val=0;}
      stock2(int x, float y){code=x; val=y;}
        void disp( ){
          cout<< " code: "<<code << "\n";
          cout<< " Total Value: Rs . " <<val <<"\n";
        }
        stock2 (stock1 p){
          code=p . getcode ( ) ;
           val=p.getitem( ) * p. getprice ( ) ;
        }
};
```

Example: Data Conversion

```
int main ( ){
        stock1 i1(101,10,125.0);
        stock2 i2;
        float tot_val;
        tot_val=i1 ;
        i2=i1 ;
        cout<<" Stock Details-stockl-type" <<"\n";
        i1.disp ( ) ;
        cout<<" Stock value: ";
        cout<< tot_val<<"\n";
        cout<<" Stock Details-stock2-type"<< "\n";
        i2.disp( ) ;
        return 0;
}
```

C++ Function Overriding



```
class A
{
   .... ... ....
   public:
     void get_data()
     {
        .... ... ....
     }
};

class B : public A
{
   .... ... ....
   public:
     void get_data()
     {
        .... ... ....
     }
};

int main()
{
   B obj;
   .... ... ....
   obj.get_data();
}
```

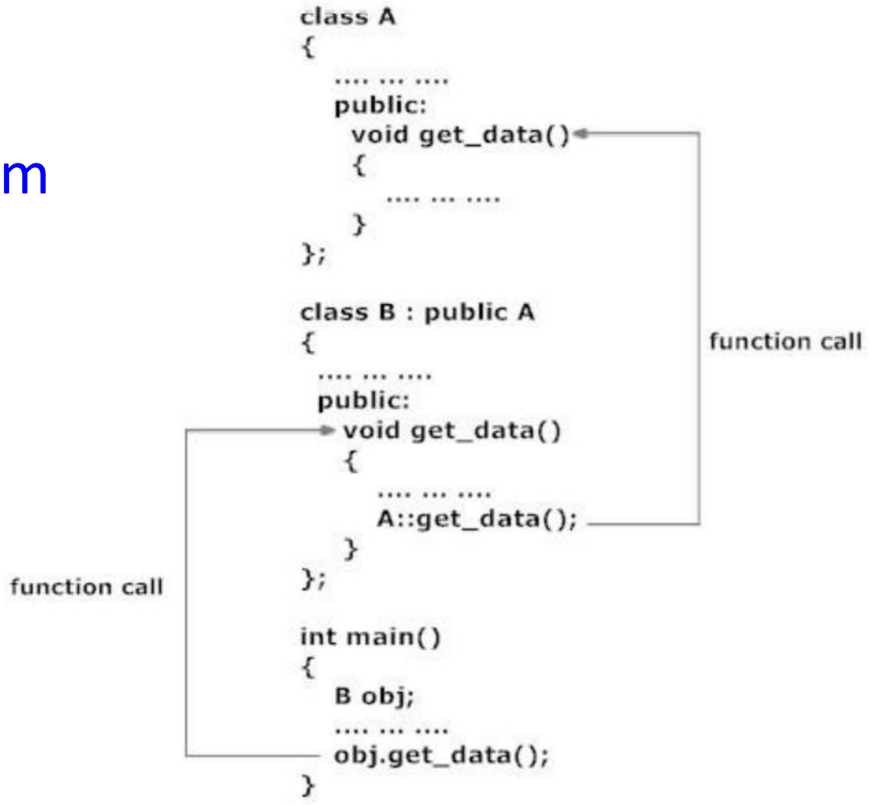
This function is not invoked in this example.

This function is invoked instead of function in class A because of member function overriding.

If base class and derived class have member functions with same name and arguments, the member function of derived class overrides the member function of base class.

C++ Function Overriding [contd.]

- Accessing the Overridden Function in Base Class From Derived Class
  - Use scope resolution operator ::



```
class A
{
   .... ... ....
   public:
     void get_data()
     {
        .... ... ....
     }
};

class B : public A
{
  .... ... ....
  public:
    void get_data()
    {
       .... ... ....
       A::get_data();
    }
};

int main()
{
   B obj;
   .... ... ....
   obj.get_data();
}
```

Programming Paradigms (revisit)

- Programming languages

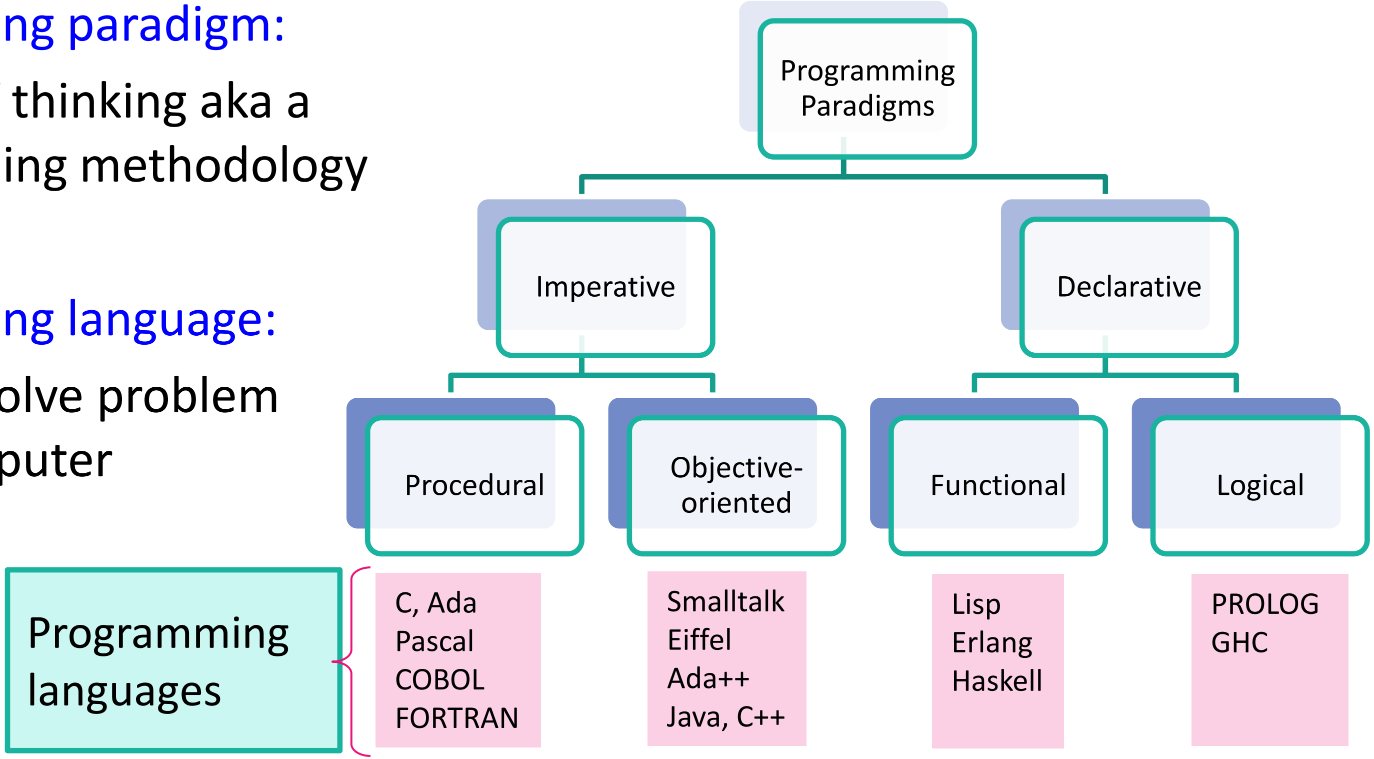
Programming Paradigms

- **Programming paradigm:**

  A mode of thinking aka a programming methodology

- **Programming language:**

  A tool to solve problem using computer



Declarative vs. Imperative Programming

Declarative vs. Imperative Programming	
A programming paradigm that expresses the logic of a computation without describing its control flow.	A programming paradigm that uses statements that changes the program’s state.
Main Focus	
Focuses on what the program should accomplish.	Focuses on how the program should achieve the result.
Flexibility	
Provides less flexibility.	Provides more flexibility.
Complexity	
Simplifies the program.	Increase the complexity of the program.
Categorization	
Functional, Logic, Query programming falls into declarative programming.	Procedural and Object Oriented programming falls into imperative programming.

Procedural vs. Object-oriented Programming (OOP)

Procedural programming	Object-oriented Programming (OOP)
In Procedural programming, a program is divided into small programs that are referred to as functions.	In OOP, a program is divided into small parts that are referred to as objects.
It follows a top-down approach	It follows a bottom-up approach
It treats data and methods separately	It encapsulates data and methods together
It is less secure than OOPs	It is more secure than procedural programming

Functional vs. Logical Programming

Functional Programming	Logical Programming
Programs are composed of functions	Programs are composed of facts and rules
Program evaluation is one-way	Program evaluation can be two-way
Helps increasing modularity	Helps representing and extracting knowledge

Logic Programming

- Programs are composed of facts and rules
  - Fact: A predicate expression that makes a declarative statement about the problem domain.
  - Rule: A predicate expression that uses logical implication (:-) to describe a relationship among facts

```
left_hand_side :- right_hand_side .
```

- Example Language:
  - **Prolog:** Programming in Logic

Demo for
Logical Programming
using Prolog

Questions?