C++

Programming

Subject: OOP using C++

www.hbpatel.in

Subject Code: CC111-N
Subject Title: Object Oriented Programming Using C++
Detail Contents:
https://ldrp.ac.in/images/syllabus/CC111-N-OBJECT%20ORIENTED%20%20PROGRAMMIENG%20USING%20C++.pdf

Kadi Sarva Vishwavidyalaya

Faculty of Engineering & Technology

First Year Bachelor of Engineering (CE / IT /EC)

(With Effect From: Academic Year 2017-18)

Subject Code: CC111-N	Subject Title: OBJECT ORIENTED PROGRAMMING USING 'C++'

Teaching scheme				Total Credit	Evaluation Scheme					
L	T	P	Total		Theory		Mid Sem Exam	CIA	Pract.	Total
Hrs	Hrs	Hrs	Hrs		Hrs	Marks	Marks	Marks	Marks	Marks
03	00	02	05	04	03	70	30	20	30	150

C++ Programming: Modules

www.hbpatel.in

- Module 1: Essentials of C Programming
- Module 2: Fundamental Concepts of OOP with C++
- Module 3: C++ Programming Syntactical Basics
- Module 4: C++ Functions
- Module 5: Objects and Class
- Module 6: Operator Overloading
- Module 7: Inheritance
- Module 8: Polymorphism & Virtual Functions
- Module 9: Templates and Exception Handling
- Module 10: Introduction to Streams and Files

KSV Examination Questions from this unit

1. What is structure? Write a program in C using structure to enter rollno and marks of three subjects for 3 students and find total obtained by each student. [5, July-2022]
2. Define a structure called student that represents student's information including their name, ID and marks for three subjects: Maths, English and Science. [5, Jun-2023]
3. What is structure? Explain the C syntax of structure declaration with example. [5, Feb-2022]
4. What is pointer? Explain how pointer variable declared and initialized? Explain it with example.
5. Define pointer. Explain pointer and array with example. [5, Jun-2023] [5, Jun-2023]
6. Define and explain datatypes: Structure and Pointer. [5, Jan-2024]
7. Explain call by reference and call by reference with example. [5, Jun-2023]

Essentials of C Programming

www.hbpatel.in

For these three topics (Structures, Pointers and File Management), I recommend you to watch the following videos prepared by me on https://www.youtube.com/@hbpatel1976.

1. Structures

a. Structures And Unions-Introduction (https://youtu.be/E7hXpesgsfg)
b. Structures And Unions-Advanced (https://youtu.be/MwnWFcdSbLQ)

2. Pointers

a. Pointer-Introduction(Part-1) (https://youtu.be/bBCbxrq3YSU)
b. Pointer-Advanced(Part-2) (https://youtu.be/_BPOS-89Jfs)

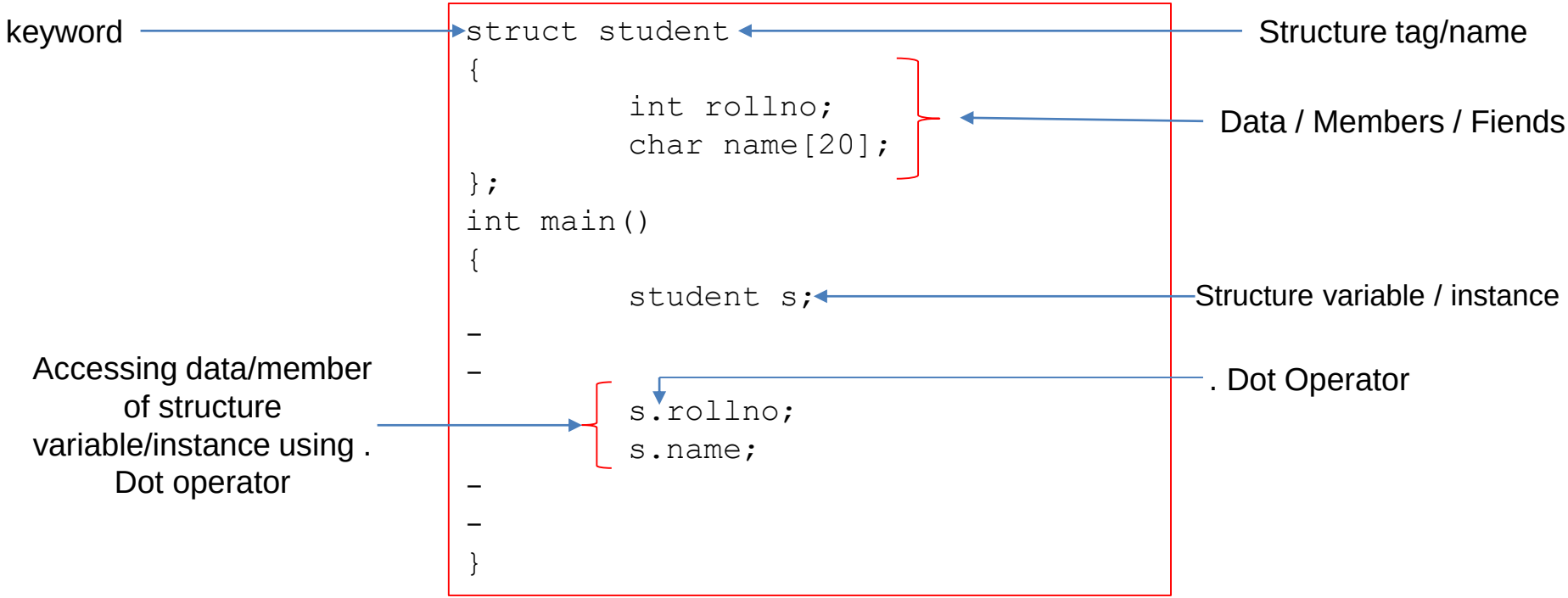
3. File Management / Handling

a. File Management Or Handling - Introduction (https://youtu.be/dSIyDKNdnX8)
b. File Management Or Handling - Advanced (https://youtu.be/xNyAw-v8OmI)

Structure in C

www.hbpatel.in

The structure in C is a user-defined data type that can be used to group items of possibly different types into a single type.



Essentials of C Programming: Structure

www.hbpatel.in

```
#include <iostream>
using namespace std;
#define totalStudents 3
struct student
{
            int rollno;
            char name[20];
};
int main()
{
            student s[totalStudents];
            int i;
            for(i=0; i<totalStudents; ++i)
            {
                        cout << "Student # " << i << endl;
                        cout << "Enter Rollno : "; cin >> s[i].rollno;
                        cout << "Enter Name : "; cin >> s[i].name;
            }
            for(i=0; i<totalStudents; ++i)
            {
                        cout << "Student # " << i << "Rollno : " << s[i].rollno << " Name : " << s[i].name << endl;
            }
}
```

OUTPUT

```
Enter Rollno : 1
Enter Name : Pradip
Student # 1
Enter Rollno : 2
Enter Name : Sanjay
Student # 2
Enter Rollno : 3
Enter Name : Parimal
Student # 0Rollno : 1 Name : Pradip
Student # 1Rollno : 2 Name : Sanjay
Student # 2Rollno : 3 Name : Parimal
```

Code: https://github.com/hbpatel1976/CPP/blob/main/102.cpp

Essentials of C Programming: Structure

www.hbpatel.in

```
#include <iostream>
using namespace std;
#define totalStudents 3
struct student
{
            int rollno;
            char name[20];
            int maths, english, science;
};
int main()
{
            student s[totalStudents];
            int i, totalMarks;
            for(i=0; i<totalStudents; ++i)
            {
                        cout << "Student # " << i << endl;
                        cout << "Enter Rollno : ";cin >> s[i].rollno;
                        cout << "Enter Name : "; cin >> s[i].name;
                        cout << "Enter Maths Marks: "; cin >> s[i].maths;
                        cout << "Enter English Marks: "; cin >> s[i].english;
                        cout << "Enter Science Marks: "; cin >> s[i].science;
            }
```

OUTPUT

```
Student # 0
Enter Rollno : 1001
Enter Name : Pradip
Enter Maths Marks: 80
Enter English Marks: 90
Enter Science Marks: 95
Student # 1
Enter Rollno : 1020
Enter Name : Sanjay
Enter Maths Marks: 55
Enter English Marks: 65
Enter Science Marks: 79
Student # 2
Enter Rollno : 1049
Enter Name : Parimal
Enter Maths Marks: 78
Enter English Marks: 39
Enter Science Marks: 66
Student # 0Rollno : 1001 Name : Pradip
Maths : 80 English : 90 Science : 95
 Total : 265 Average : 88
Student # 1Rollno : 1020 Name : Sanjay
Maths : 55 English : 65 Science : 79
 Total : 199 Average : 66
Student # 2Rollno : 1049 Name : Parimal
Maths : 78 English : 39 Science : 66
 Total : 183 Average : 61
```

Code: https://github.com/hbpatel1976/CPP/blob/main/102.cpp

Essentials of C Programming: Structure

www.hbpatel.in

```
for(i=0; i<totalStudents; ++i)
{
        cout << "Student # " << i << "Rollno : " << s[i].rollno << " Name : " << s[i].name << endl;
        cout << "Maths : " << s[i].maths << " English : " << s[i].english << " Science : " << s[i].science << endl
        totalMarks = s[i].maths + s[i].english + s[i].science;
        cout << " Total : " << totalMarks << " Average : " << totalMarks/totalStudents << endl;
}
}
```

OUTPUT

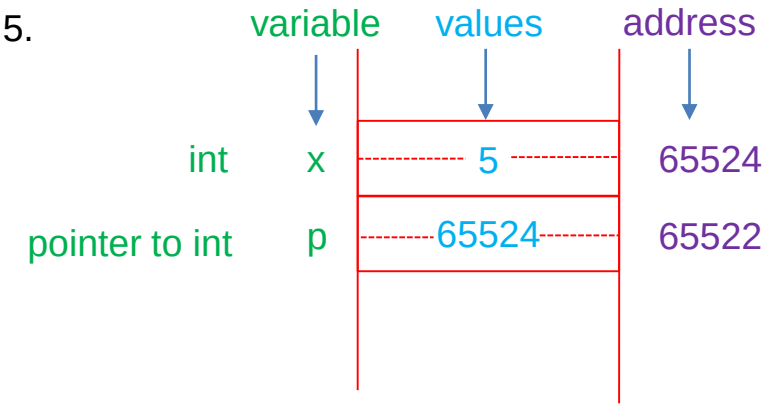
```
Student # 0
Enter Rollno : 1001
Enter Name : Pradip
Enter Maths Marks: 80
Enter English Marks: 90
Enter Science Marks: 95
Student # 1
Enter Rollno : 1020
Enter Name : Sanjay
Enter Maths Marks: 55
Enter English Marks: 65
Enter Science Marks: 79
Student # 2
Enter Rollno : 1049
Enter Name : Parimal
Enter Maths Marks: 78
Enter English Marks: 39
Enter Science Marks: 66
Student # 0Rollno : 1001 Name : Pradip
Maths : 80 English : 90 Science : 95
 Total : 265 Average : 88
Student # 1Rollno : 1020 Name : Sanjay
Maths : 55 English : 65 Science : 79
 Total : 199 Average : 66
Student # 2Rollno : 1049 Name : Parimal
Maths : 78 English : 39 Science : 66
 Total : 183 Average : 61
```

Code: https://github.com/hbpatel1976/CPP/blob/main/102.cpp

Essentials of C Programming: Pointer

A pointer is defined as a derived data type that can store the address of other C variables or a memory location. We can access and manipulate the data stored in that memory location using pointers.

`int x=5;` Here x is a simple integer variable whose value is 5.
`int *p=&x;` Here p is a pointer who holds the address of x.



Essentials of C Programming: Pointer


```
#include <stdio.h>
void main()
{
int x=5;
int *p=&x;
printf("x is stored at %u location and value is %d\n",&x,x);
printf("value of p is %u and contents pointed by it is %d\n",p,*p);
}
```

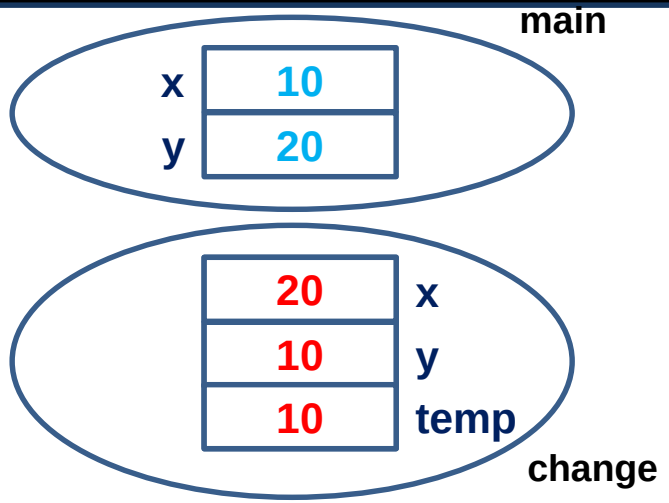
OUTPUT

```
x is stored at 65524 location and value is 5
value of p is 65524 and contents pointed by it is 5
```

Essentials of C Programming: Pass by value

www.hbpatel.in

```
#include <stdio.h>
void main()
{
void change(int,int);
int x=10,y=20;
printf("Before Function : x=%d y=%d\n",x,y);
change(x,y);
printf("After Function : x=%d y=%d\n",x,y);
}


void change(int x, int y)
{
int temp;
temp=x;
x=y;
y=temp;
}
```



OUTPUT

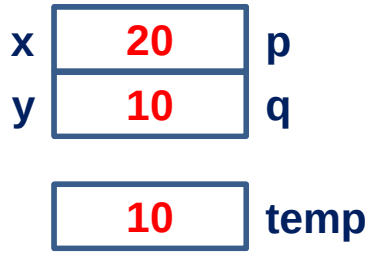
```
Before Function : x=10 y=20
After Function : x=10 y=20
```

Pass/Call By Value/Reference: Video: https://youtu.be/dAtoOZN_kTk

Essentials of C Programming: Pass by reference

www.hbpatel.in

```
#include <stdio.h>
void main()
{
void change(int&,int&);
int x=10,y=20;
printf("Before Function : x=%d y=%d\n",x,y);
change(x,y);
printf("After Function : x=%d y=%d\n",x,y);
}
```




```
void change(int &p, int &q)
{
int temp;
temp=p;
p=q;
q=temp;
}
```

OUTPUT

```
Before Function : x=10 y=20
After Function : x=20 y=10
```

Pass/Call By Value/Reference: Video: https://youtu.be/dAtoOZN_kTk

C++ Programming: Modules

www.hbpatel.in

- Module 1: Essentials of C Programming
- Module 2: Fundamental Concepts of OOP with C++
- Module 3: C++ Programming Syntactical Basics
- Module 4: C++ Functions
- Module 5: Objects and Class
- Module 6: Operator Overloading
- Module 7: Inheritance
- Module 8: Polymorphism & Virtual Functions
- Module 9: Templates and Exception Handling
- Module 10: Introduction to Streams and Files

Fundamental Concepts of OOP with C++

KSV Examination Questions from this unit

1. Explain basic concepts of OOPC and its advantages. [5, Jan-2023] [5, July-2022] [5, Jan-2024]
2. Explain the basic building concept of OOP. [5, Jan-2024]
3. Write structure of C++ program and explain in brief. [5, Jan-2023]
4. Differentiate between Procedure Oriented Programming (POP) and Object Oriented Programming (OOP). [5, Jan-2023] [5, July-2022] [5, Feb-2022] [5, Jan-2024]
5. Distinguish between: Data abstraction and Data encapsulation. [5, Jan-2023] [5, Feb-2022]
6. Distinguish between: Dynamic binding and Message passing. [5, Jan-2023]
7. Explain Dynamic Binding with a program. [5, Jan-2024]
8. Define terms: (i) Objects (ii) Class (iii) Constant Variable (iv) Token (v) Identifier. [5, July-2022] (iii) destructor [5, Jun-2023]
9. How does main() function in C differ from C++? Give general format of Class. [5, July-2022] [2.5, Jun-2023]

C++ Program Structure

www.hbpatel.in

	#include <iostream>	Header File
	using namespace std;	Standard Namespace
Class Body	class Car	Class Definition
	{	
	public:	Access Modifiers
	int price;	Data Member
	void showCost()	Member Function
	{	
	cout << price;	
	}	
	};	
Main function	int main()	main method
	{	
	Car city;	Object declaration
	city.price=1000000;	Accessing data member
	city.showCost();	Accessing member function
	}	

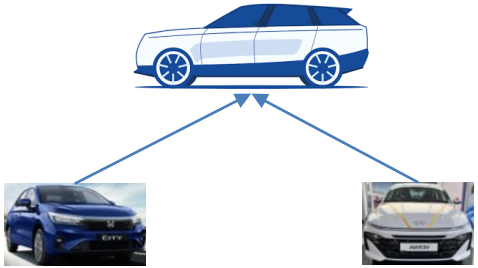
Fundamental Concepts of OOP with C++

www.hbpatel.in

1. **Class**
2. Objects
3. Encapsulation
4. Abstraction
5. Polymorphism
6. Inheritance
7. Dynamic Binding
8. Message Passing

1. **Class**: It is a blueprint type user-defined datatype from which an object is created. Class contains definition & declaration of data members and functions.

```
class Car
            {
            public:
                        int price;
                        void showCost()
                        {
                        cout << price;
                        }
            };
int main()
{
            Car city;
            city.price=1000000;
            city.showCost();
}
```



City
Color:
Price:
Average:
Model:

Verna
Color:
Price:
Average:
Model:

Class: `Car`
Object: `city`
Data member: `price`
Member function: `showCost`

Fundamental Concepts of OOP with C++

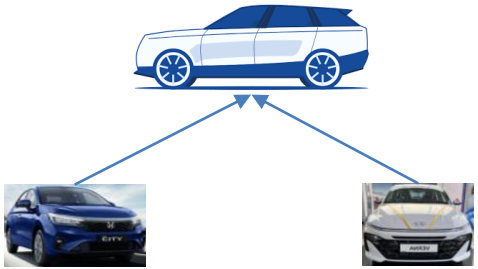
www.hbpatel.in

1. Class
2. **Objects**
3. Encapsulation
4. Abstraction
5. Polymorphism
6. Inheritance
7. Dynamic Binding
8. Message Passing

2. **Object**: It is an instance of a class which is uniquely identifiable and physical memory is allocated for it.

Class: `Car`
Object: `city`
Data member: `price`
Member function: `showCost`

```
class Car
            {
            public:
                        int price;
                        void showCost()
                        {
                        cout << price;
                        }
            };
int main()
{
            Car city;
            city.price=1000000;
            city.showCost();
}
```



City
Color:
Price:
Average:
Model:

Verna
Color:
Price:
Average:
Model:

Fundamental Concepts of OOP with C++

www.hbpatel.in

1. Class
2. Objects
3. **Encapsulation**
4. Abstraction
5. Polymorphism
6. Inheritance
7. Dynamic Binding
8. Message Passing

3. **Encapsulation**: It is a mean to encapsulate/wrap data and functions together in a single entity called Class.

```
class Car
            {
            public:
                        int price;
                        void showCost()
                        {
                        cout << price;
                        }
            };
int main()
{
            Car city;
            city.price=1000000;
            city.showCost();
}
```

Class: Car

Class: `Car`
Object: `city`
Data member: `price`
Member function: `showCost`

Fundamental Concepts of OOP with C++

www.hbpatel.in

1. Class
2. Objects
3. Encapsulation
4. **Abstraction**
5. Polymorphism
6. Inheritance
7. Dynamic Binding
8. Message Passing

4. **Abstraction**: It is about permitting displaying only required data/function and hiding the rest. General classes are designed as abstract, and more specific classes are derived from these general classes.

```
#include <iostream>
using namespace std;
#include <iostream>
using namespace std;
class Base
            {
            int x;
            public:
                        virtual void fun() = 0;
                        int getX() { return x; }
            };
```

```
class Derived : public Base
{
int y;
public:
            void fun() { cout << "fun() called"; }
};

int main(void)
{
            Derived d;
            d.fun();
            return 0;
}
```

1. Class
2. Objects
3. Encapsulation
4. Abstraction
5. **Polymorphism**
6. Inheritance
7. Dynamic Binding
8. Message Passing

5. **Polymorphism**: It is about having multiple forms of a single entity. A lady can be a sister, a mother, a daughter, an employee etc. E.g. operator overloading, function overloading.

```
#include <iostream>
using namespace std;
class poly
{
        public:
                int sum(int x, int y) {return x+y;}
                int sum(int x, int y, int z) {return x+y+z;}
};

int main(void)
{
        poly p1, p2;
        cout << p1.sum(10,20) << endl;
        cout << p2.sum(10,20,30) << endl;
}
```

Fundamental Concepts of OOP with C++

www.hbpatel.in

1. Class
2. Objects
3. Encapsulation
4. Abstraction
5. Polymorphism
6. **Inheritance**
7. Dynamic Binding
8. Message Passing

6. **Inheritance**: It is a process by which a new class is derived from an existing one. (Reusability)

```
#include <iostream>
using namespace std;
class Father
{
                private:
                                int saving;
                public:
                                int setSaving(int x) {saving=x;}
                                void showSaving() {cout << "Father Saving : " << saving << endl;}
};
class Son : private Father
{
                private:
                                int salary;
                public:
                                int setProperty(int x, int y) {setSaving(y);salary=x;}
                                int showProperty() {showSaving(); cout << "Son Salary : " << salary << endl;}
};
int main(void)
{
                Son s;
                s.setProperty(100000, 5000000);
                s.showProperty();
}
```

Fundamental Concepts of OOP with C++

www.hbpatel.in

1. Class
2. Objects
3. Encapsulation
4. Abstraction
5. Polymorphism
6. Inheritance
7. **Dynamic Binding**
8. Message Passing

7. **Dynamic Binding**: It is a process by which a new class is derived from an existing one. (Reusability)

```
class Father
{
                public:
                                        void callFunction() {show();}
                                        virtual void show(){cout << "This is a Father(base) class" << endl;}
};
class Son : public Father
{
                public:
                                        void show(){cout << "This is a Son(derived) class" << endl;}
};
int main(void)
{
                Father f;
                Son s;
                f.show();
                s.show();
}
```

OUTPUT

```
This is a Father(base) class
This is a Son(derived) class
```

Fundamental Concepts of OOP with C++

1. Class
2. Objects
3. Encapsulation
4. Abstraction
5. Polymorphism
6. Inheritance
7. Dynamic Binding
8. **Message Passing**

8. **Message Passing**: Objects communicate with one another by sending and receiving information. A message for an object is a request for the execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function, and the information to be sent.

Procedural Programming Vs. OOP

www.hbpatel.in

Procedural Oriented Programming	Object-Oriented Programming
In procedural programming, the program is divided into small parts called ***functions***.	In object-oriented programming, the program is divided into small parts called ***objects***.
Procedural programming follows a ***top-down approach***.	Object-oriented programming follows a ***bottom-up approach***.
There is no access specifier in procedural programming.	Object-oriented programming has access specifiers like private, public, protected, etc.
Adding new data and functions is not easy.	Adding new data and function is easy.
Procedural programming does not have any proper way of hiding data so it is ***less secure***.	Object-oriented programming provides data hiding so it is ***more secure***.
In procedural programming, overloading is not possible.	Overloading is possible in object-oriented programming.
In procedural programming, there is no concept of data hiding and inheritance.	In object-oriented programming, the concept of data hiding and inheritance is used.
In procedural programming, the function is more important than the data.	In object-oriented programming, data is more important than function.
Procedural programming is based on the ***unreal world***.	Object-oriented programming is based on the ***real world***.
Procedural programming uses the concept of procedure abstraction.	Object-oriented programming uses the concept of data abstraction.
Examples: C, FORTRAN, Pascal, Basic, etc.	**Examples:** C++, Java, Python, C#, etc.

Source: www.geeksforgeeks.org/

C++ Tokens

www.hbpatel.in

- Keywords
  - `char, short, int, long, float, double, signed, unsigned extern, static, auto, const struct, union for, while, do break, continue, return if, else switch, case void, default, enum, goto, register, sizeof, typedef, volatile`
- Identifiers
  - Naming Convention:
    1. Only the alphabetic characters, digits, and underscores are allowed.
    2. The first letter should be an alphabet or an underscore (_).
    3. The identifiers are case-sensitive.
    4. Keywords that are reserved can't be used as the name of the identifier.
- Constants
  - Constants are similar to a variable, except the fact that their values do not get changed. `const` and `#define` are the two means to declare constants.
- Variables
  - A variable is a meaningful name given to a data storage location in main memory.
- Operators
  - Next Slide

Operators in C++

www.hbpatel.in

Arithmetic

* Multiplication
/ Division
% Modulo
+ Addition
–Subtraction

Unary

– Unary minus
+ Unary plus
++ Increment
– – Decrement
! Logical not

Assignment

= Assignment
+= -= *= /= %=

Relational

== Equal to
!= Not Equal to
< less than
<= less than or equal to
> Greater than
>= Greater than or equal to

Logical

&& Logical AND
|| Logical OR
! Logical NOT

Shift

<< Left shift operator
>> Signed Right shift
>>> Unsigned Right shift

Other operator

? : (Ternary)
:: (Scope Resolution)

Bitwise

& Bitwise AND
| Bitwise OR
^ Bitwise XOR
~ Bitwise Complement

C++ Programming: Modules

www.hbpatel.in

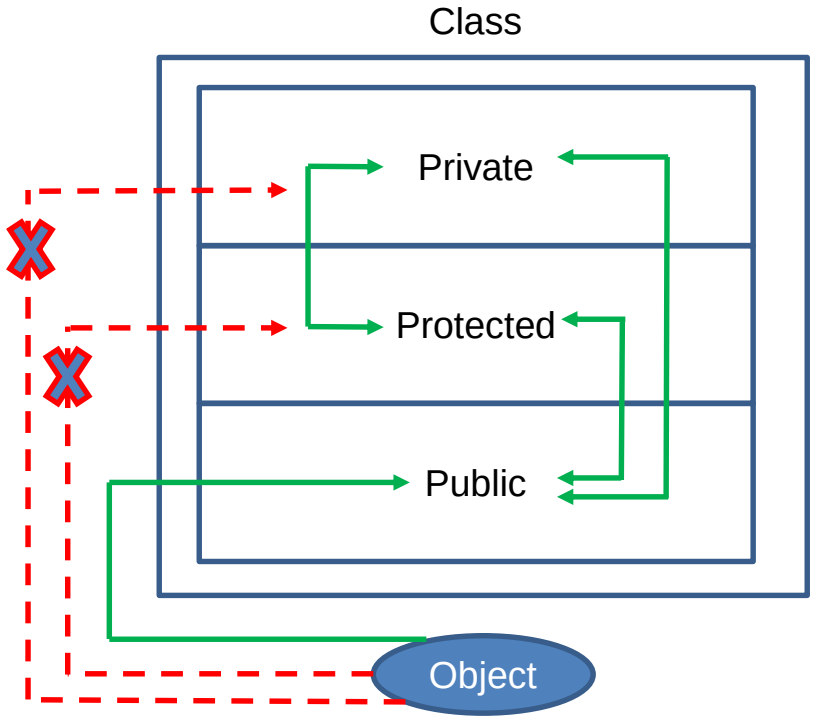
- Module 1: Essentials of C Programming
- Module 2: Fundamental Concepts of OOP with C++
- Module 3: C++ Programming Syntactical Basics
- Module 4: C++ Functions
- Module 5: Objects and Class
- Module 6: Operator Overloading
- Module 7: Inheritance
- Module 8: Polymorphism & Virtual Functions
- Module 9: Templates and Exception Handling
- Module 10: Introduction to Streams and Files

C++ Programming Syntactical Basics

KSV Examination Questions from this unit

1. Define Class and Object. Write syntax for accessing a data member and member function of a class. Explain how to define and access member function inside the class with example. [5, Jan-2023]
2. What is class and object? How is it created? Explain with example. [5, Feb-2022]
3. What is the use of access specifiers? Explain various types of access specifiers. [5, Jan-2023]
4. State the 3 access specifiers of inheritance and explain it in brief. [5, Jan-2024]
5. Explain access specifiers: public and private with [5, July-2022] [5, Jun-2023]
6. Discuss the role of access modes in inheritance and show their visibility when they are inherited as public, private and protected. [5, Feb-2022]
7. Explain the use of setw and endl manipulators. Explain << and >> operators. [5, July-2022]
8. List any three manipulators and explain in detail. [5, Jan-2024]
9. What is type conversion in C++? Explain implicit and explicit type conversion with example. [5, Feb-2022]
10. Explain type conversion from class type to basic type and one class type to another class type with suitable example. [5, Jun-2023] [5, Jan-2024]
11. Explain Scope Resolution (::) operator with suitable example. [5, Jan-2023] [5, July-2022] [5, Feb-2022] [5, Jun-2023]
12. Demonstrate the global scope and local scope using scope resolution operator in Program. [5, Jan-2024]
13. Explain “this” pointer with example. [5, Jun-2023]
14. What does 'this' keyword refer to? Explain with a program. [5, Jan-2024]

Access Specifiers



Access Control Specifier	Accessible To	
	Own Class Members	Objects of a Class
Private	Yes	No
Protected	Yes	No
Public	Yes	Yes

Access Specifiers


```
#include <iostream>
using namespace std;
class Test
{
private:
              int priData;
              void priFunction()
                            {cout << "Function in Private Section\n";}
protected:
              int proData;
              void proFunction()
                            {cout << "Function in Protected Section\n";}
public:
              int pubData;
              void pubFunction()
                            {cout << "Function in Public Section\n";}
};
void main()
{             Test t;
              t.priData = 10;   /* Error */
              t.priFunction(); /* Error */

              t.proData = 20;   /* Error */
              t.proFunction(); /* Error */

              t.pubData = 30;   /* OK */
              t.pubFunction(); /* OK */
}
```

Access Control Specifier	Accessible To	
	Own Class Members	Objects of a Class
Private	Yes	No
Protected	Yes	No
Public	Yes	Yes

Access Specifiers


```
#include <iostream>
using namespace std;
class Test
{
private:
        int priData;
        void priFunction()
                {cout << "Function in Private Section\n";
                proFunction(); /* OK */
                pubFunction(); /* OK */
                }
protected:
        int proData;
        void proFunction()
                {cout << "Function in Protected Section\n";
                priFunction(); /* OK */
                pubFunction(); /* OK */
                }
public:
        int pubData;
        void pubFunction()
                {cout << "Function in Public Section\n";
                priFunction(); /* OK */
                proFunction(); /* OK */
                }
};
```

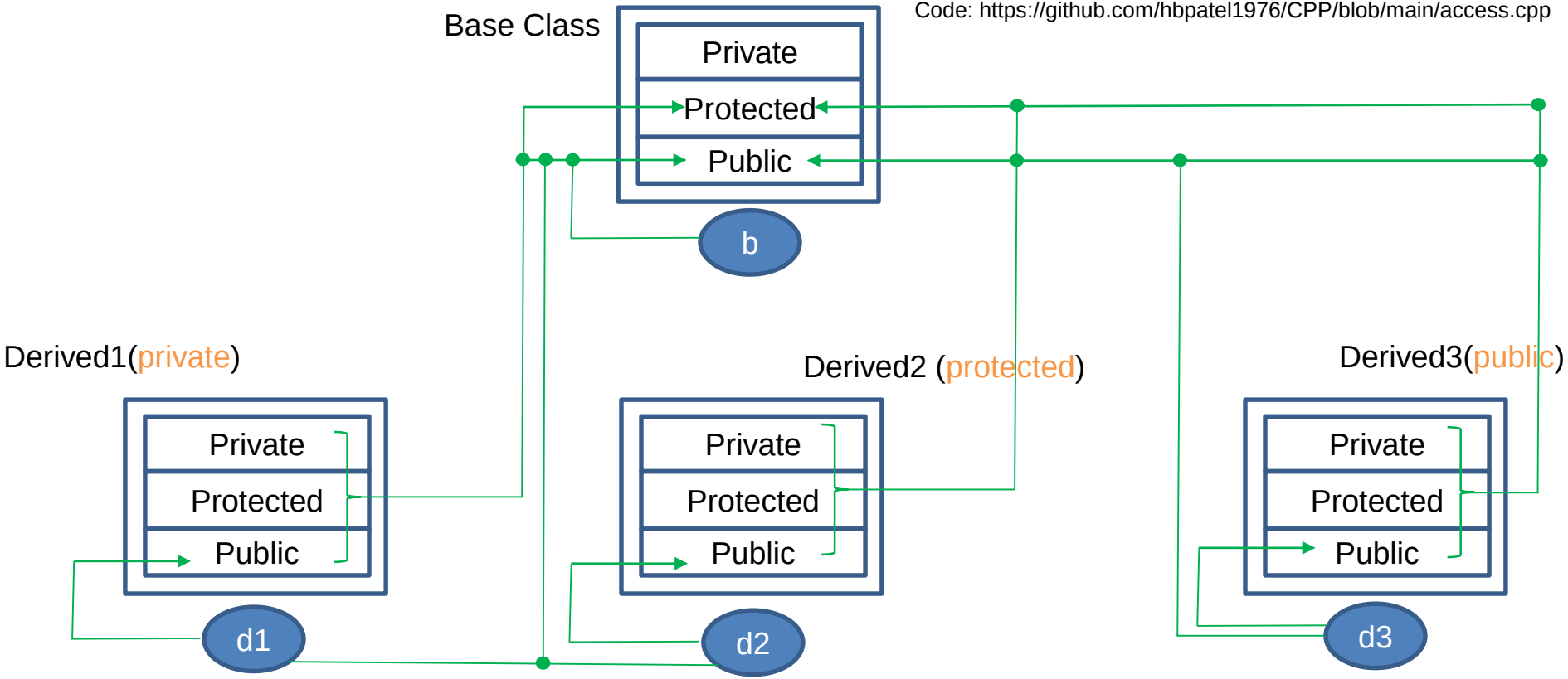
```
void main()
{
        Test t;
        t.priData = 10;  /* Error */
        t.priFunction(); /* Error */

        t.proData = 20;  /* Error */
        t.proFunction(); /* Error */

        t.pubData = 30;  /* OK */
        t.pubFunction(); /* OK */
}
```

Access Control Specifier	Accessible To	
	Own Class Members	Objects of a Class
Private	Yes	No
Protected	Yes	No
Public	Yes	Yes

Access Specifiers: Inheritance

www.hbpatel.in

Code: https://github.com/hbpatel1976/CPP/blob/main/access.cpp



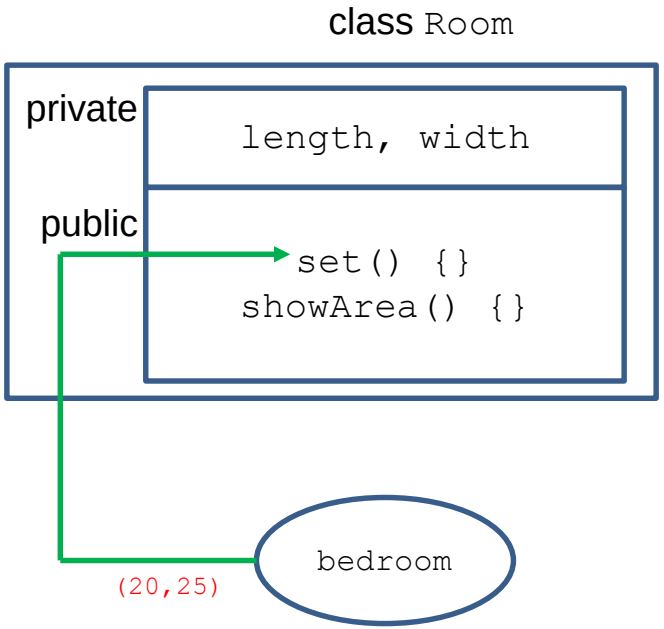
Programming: Object & Class
(A Class and An Object)

www.hbpatel.in

```
#include <iostream>
using namespace std;
class Room
{
private:
   float length, width;
public:
   void set(float l, float w){length=l; width=w;}
   void showArea()
        {cout << " Area : " << length * width << endl;}
};

int main()
{
Room bedroom;
bedroom.set(20, 25);
bedroom.showArea();
}
```

OUTPUT

```
Area : 500
```



Class Vs. Structure

www.hbpatel.in

```
#include <iostream>
using namespace std;
struct student
{
            int rollno, maths, english, hindi;
};
class employee
{
public:     int code, salary;
            employee(int c, int s) {code=c; salary=s;}
};
int main()
{
struct student himanshu={17,48,59,69}, monu={2,47,56,40};
printf("Himanshu Total = %d\n", himanshu.maths + himanshu.english + himanshu.hindi);
printf("Monu Total = %d\n", monu.maths + monu.english + monu.hindi);

employee hiren(2020,10000), hardik(2039, 15000);
cout << "Hiren Code = " << hiren.code << " Salary = " << hiren.salary << endl;
cout << "Hardik Code = " << hardik.code << " Salary = " << hardik.salary << endl;
}
```

OUTPUT

```
Himanshu Total = 176
Monu Total = 143
Hiren Code = 2020 Salary = 10000
Hardik Code = 2039 Salary = 15000
```

Class Vs. Structure

www.hbpatel.in

Class	Structure
1. Members of a class are private by default.	1. Members of a structure are public by default.
2. An instance of a class is called an ‘object’.	2. An instance of structure is called the ‘structure variable’.
3. It is declared using the `class` keyword.	3. It is declared using the `struct` keyword.
4. It is normally used for data abstraction and further inheritance.	4. It is normally used for the grouping of data
5. NULL values are possible in Class.	5. NULL values are not possible.

6. Syntax:
`class class_name{`
`data_member;`
`member_function;`
`};`
	6. Syntax:
	`struct structure_name{`
	`type structure_member1;`
	`type structure_member2;`
	`};`

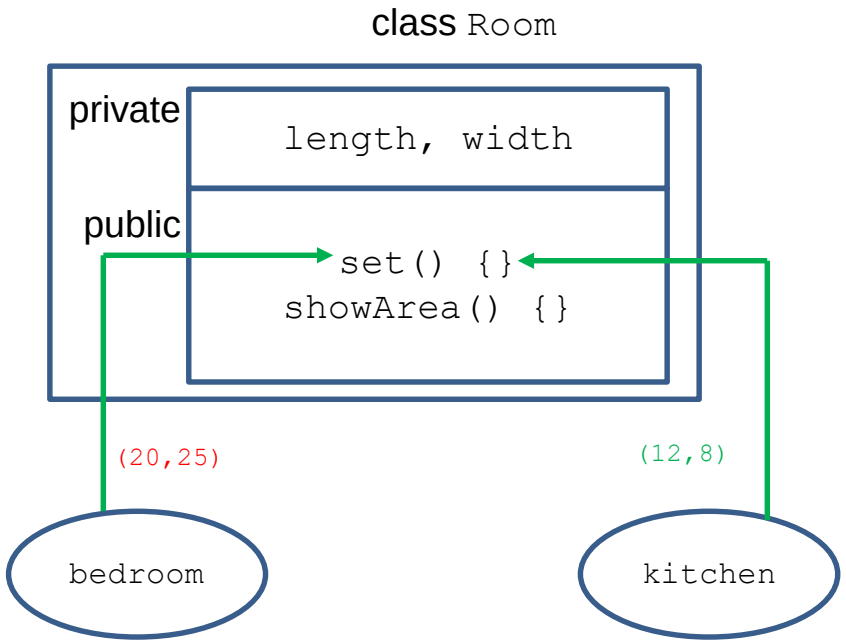
Programming: Object & Class
(A Class and Two Objects)

www.hbpatel.in

```
#include <iostream>
using namespace std;
class Room
{
private:
   float length, width;
public:
   void set(float l, float w){length=l; width=w;}
   void showArea()
       {cout << " Area : " << length * width << endl;}
};

int main()
{
Room bedroom, kitchen;
bedroom.set(20, 25);
kitchen.set(12,8);
std::cout << std::setw(5);
cout << "Bedroom ";
bedroom.showArea();
cout << "Kitchen ";
kitchen.showArea();
}
```

OUTPUT

```
Bedroom   Area : 500
Kitchen   Area : 96
```



Code: https://github.com/hbpatel1976/CPP/blob/main/obj01.cpp

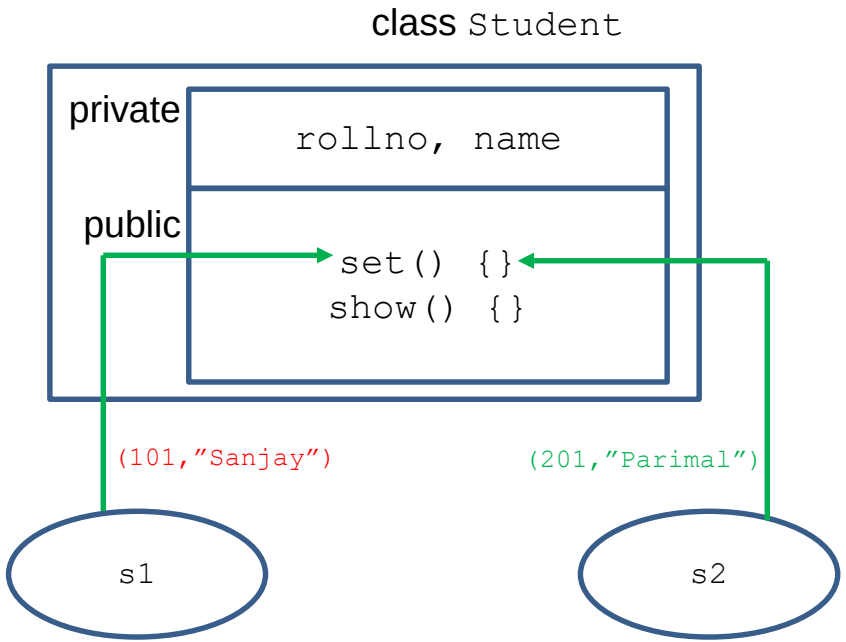
Programming: Object & Class
(A Class and Two Objects)

www.hbpatel.in

```
#include <iostream>
#include <string.h>
using namespace std;
class Student
{
private:
    int rollno;
    char name[20];
public:
    void set(int r, char n[20])
            {rollno=r; strcpy(name,n);}
     void show()
            {cout << rollno << "\t" << name << endl;}
};
int main()
{
            Student s1,s2;
            s1.set(101,"Sanjay");
            s2.set(201,"Parimal");
            s1.show();
            s2.show();
}
```

OUTPUT

```
101     Sanjay
201     Parimal
```



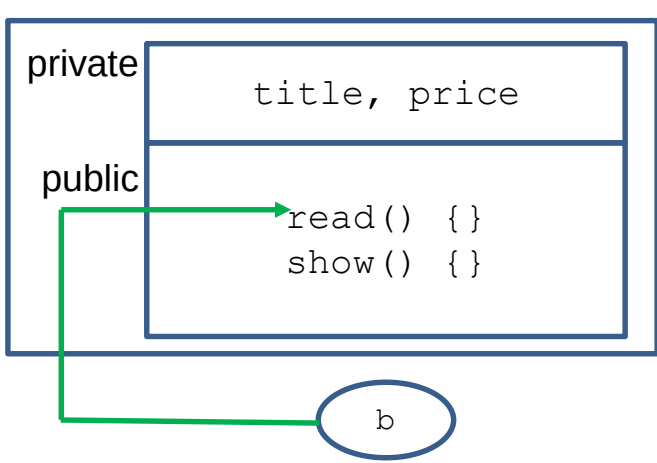
Code: https://github.com/hbpatel1976/CPP/blob/main/obj02.cpp

Programming: Object & Class

(A Class and An Object)

www.hbpatel.in

```
#include <iostream>
#include <string.h>
using namespace std;
class Book
{
private:
    char title[20];
    float price;
public:
    void read()
            {cout << "Enter Book Title: ";
            cin >> title;
            cout << "Enter Book Price: ";
            cin >> price;
            }
    void show()
            {cout << "Book : " << title << " Price : " << price << endl;}
};
int main()
{
    Book b;
    b.read();
    b.show();
}
```



OUTPUT

```
Enter Book Title: C++Programming
Enter Book Price: 200
Book : C++Programming Price : 200
```

Code: https://github.com/hbpatel1976/CPP/blob/main/obj03.cpp

Programming: Object & Class

(Array of Objects)

www.hbpatel.in

```
#include <iostream>
using namespace std;
#define total 3
class Room
{
private:
    float length, width;
public:
    void read()
            {
            cout << "Enter Length : "; cin >> length;
            cout << "Enter Width : "; cin >> width;
            }
    void showArea() {cout << "Area : " << length * width << endl;}
};

int main()
{
   int i;
   Room r[total];
   for(i=0; i<total; ++i){r[i].read();}
   for(i=0; i<total; ++i){r[i].showArea();}
}
```

OUTPUT

```
Enter Length : 10
Enter Width : 20
Enter Length : 12
Enter Width : 14
Enter Length : 9
Enter Width : 15
Area : 200
Area : 168
Area : 135
```

Code: https://github.com/hbpatel1976/CPP/blob/main/obj04.cpp

Programming: Object & Class

(Scope Resolution Operator ::)

www.hbpatel.in

```
#include <iostream>
using namespace std;
#define total 3
class Room
{
private:
    float length, width;
public:
    void read();
    void showArea();
};
```

```
void Room::read()
    {
    cout << "Enter Length : "; cin >> length;
    cout << "Enter Width : "; cin >> width;
    }
void Room::showArea()
    {
    cout << " Area : " << length * width << endl;
    }

int main()
{
    int i;
    Room r[total];
    for(i=0; i<total; ++i){r[i].read();}
    for(i=0; i<total; ++i){r[i].showArea();}
}
```

OUTPUT

```
Enter Length : 10
Enter Width : 20
Enter Length : 12
Enter Width : 14
Enter Length : 9
Enter Width : 15
Area : 200
Area : 168
Area : 135
```

Code: https://github.com/hbpatel1976/CPP/blob/main/obj05.cpp

Programming: Object & Class

(Local & Global Scope with Scope Resolution Operator)

www.hbpatel.in

```
#include <iostream>
using namespace std;
int data = 100; // Global Variable
int main()
{
        int data = 54; // Local Variable
        cout << "Local Variable : " << data;
        cout << "\nGlobal Variable : " << ::data;
}
```

OUTPUT

```
Local Variable : 54
Global Variable : 100
```

Code: https://github.com/hbpatel1976/CPP/blob/main/obj06.cpp

Programming: Object & Class

(setw, endl, showpoint, setprecision manipulators)

www.hbpatel.in

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
            int data1=10, data2=1000;
            long double pi = 3.141592653589793239;
            cout << data1 << endl;
            cout << data2 << endl;

            std::cout << std::setw(10);
            cout << data1 << endl;
            std::cout << std::setw(10);
            cout << data2 << endl;

            cout << "1.0 with showpoint: " << std::showpoint << 1.0 << '\n';
            cout << "1.0 with noshowpoint: " << std::noshowpoint << 1.0 << '\n';

            cout << "default precision (6): " << pi << '\n';
            cout << "std::setprecision(10): " << std::setprecision(10) << pi << '\n';
}
```

OUTPUT

```
10
1000
        10
      1000
1.0 with showpoint: 1.00000
1.0 with noshowpoint: 1
default precision (6): 3.14159
std::setprecision(10): 3.141592654
```

Code: https://github.com/hbpatel1976/CPP/blob/main/obj00.cpp

Type Conversion

www.hbpatel.in

```
#include <iostream>
using namespace std;
int main()
{
    int a1 = 10, a2;
    char b1 = 'a', b2;
    float c1 = 12.34, c2;
    double d1 = 4.2;

    a2 = a1 + b1; // Implicit Conversion ['a' -> 97]
    b2 = b1 + 10; // Implicit Conversion 'a' + 10 = 97 + 10 = 107 => 'k'
    c2 = c1 + a1; // Implicit Conversion [10 -> 10.0]

    cout << a2 << endl;
    cout << b2 << endl;
    cout << c2 << endl;

    c2 = a1 / a2;
    cout << c2 << endl;

    c2 = (float)a1 / a2; // Explicit Conversion of int into float
    cout << c2 << endl;
}
```

OUTPUT

```
107
k
22.34
0
0.0934579
```

C++ Programming: Modules

www.hbpatel.in

- Module 1: Essentials of C Programming
- Module 2: Fundamental Concepts of OOP with C++
- Module 3: C++ Programming Syntactical Basics
- Module 4: C++ Functions
- Module 5: Objects and Class
- Module 6: Operator Overloading
- Module 7: Inheritance
- Module 8: Polymorphism & Virtual Functions
- Module 9: Templates and Exception Handling
- Module 10: Introduction to Streams and Files

C++ Functions

www.hbpatel.in

KSV Examination Questions from this unit

1. What is function overloading in C++? [5, Feb-2022] Write a program that overloads volume functions that return volume of a cube, cuboids and cylinder. [5, Jan-2023]
2. Explain function overloading. [5, Jun-2023] Write a program to calculate area of rectangle and triangle using function overloading. [5, July-2022] [5, Feb-2022]
3. Explain the concept of function overloading by overloading 3 function in a program.[5, Jan-2024]
4. What is reference variable? Explain with suitable example. [5, Jan-2023]
5. What is reference variable in C++? [5, July-2022] [2.5, Jun-2023]
6. Create a class TIME with members: hour, minute and second. Read values from keyboard and add two TIME objects (hint: by passing objects to function) and display the result. [5, Feb-2022]
7. How to pass an object as an argument. Explain using a program. [5, Jan-2024]

Simple Function


```
#include <iostream>
using namespace std;

int main()
{
        void myFunction(void);     // Function Declaration
        myFunction();              // Function Call
}

void myFunction(void)              // Function Definition/Body
{
        cout << "This is myFunction\n";
}
```

OUTPUT

```
This is myFunction
```

Simple Function

www.hbpatel.in

```
#include <iostream>
using namespace std;

int main()
{
        void myFunction(void);    // Function Declaration
        myFunction();             // Function Call
        myFunction();             // Function Call
        myFunction();             // Function Call
}

void myFunction(void)             // Function Definition/Body
{
        cout << "This is myFunction\n";
}
```

OUTPUT

```
This is myFunction
This is myFunction
This is myFunction
```

Simple Function

www.hbpatel.in

```
#include <iostream>
using namespace std;

int main()
{
        void myFunction();          // Function Declaration
        myFunction();               // Function Call
        myFunction();               // Function Call
        myFunction();               // Function Call
}

void myFunction(void)               // Function Definition/Body
{
        cout << "This is myFunction\n";
}
```

OUTPUT

```
This is myFunction
This is myFunction
This is myFunction
```

Function Overloading


```
#include <iostream>
using namespace std;
int main()
{
        void Function(int), Function(int,int), Function(int,int,int);
        Function(10);
        Function(10,20);
        Function(10,20,30);
}
void Function(int x)
{
        cout << "Function with one argument\n";
}
void Function(int x, int y)
{
        cout << "Function with two arguments\n";
}
void Function(int x, int y, int z)
{
        cout << "Function with three arguments\n";
}
```

OUTPUT

```
Function with one argument
Function with two arguments
Function with three arguments
```

Function Overloading

www.hbpatel.in

- Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters (arguments).
- In function overloading "Function" name should be the same and the arguments should be different.
- Function overloading can be considered as an example of a polymorphism feature in C++.
- Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the function such as `a(int,int)` for two parameters, and `b(int,int,int)` for three parameters then it may be difficult for you to understand the behaviour of the function because its name differs.

```
#include <iostream>
using namespace std;
class Summation
{
public:
int add(int x) {return x;}
int add(int x, int y) {return x+y;}
int add(int x, int y, int z) {return x+y+z;}

};
int main()
{
Summation s;
cout << s.add(10) << endl;
cout << s.add(10,20) << endl;
cout << s.add(10,20,30) << endl;
}
```

OUTPUT

```
10
30
60
```

Function Overloading


```
#include <iostream>
using namespace std;

int main()
{
          int add(int, int);
          double add(double,double);
          cout << add(10,20) << endl;
          cout << add(5.8,7.9) << endl;
}
int add(int x, int y)
{
          cout << "Integer Addition Function : ";
          return x+y;
}
double add(double x, double y)
{
          cout << "Double Addition Function : ";
          return x+y;
}
```

OUTPUT

```
Integer Addition Function : 30
Double Addition Function : 13.7
```

Function Overloading


```
#include <iostream>
using namespace std;
int main()
{
        void print(int),print(char),print(double);
        print(10);
        print('a');
        print(10.345);
}
void print(int x)
{
        cout << "Integer : " << x << endl;
}
void print(char x)
{
        cout << "Character : " << x << endl;
}
void print(double x)
{
        cout << "Double : " << x << endl;
}
```

OUTPUT

```
Integer : 10
Character : a
Double : 10.345
```

this Pointer


```
#include <iostream>
using namespace std;
class Time
{
private: int hour, minute, second;
public:
            void setTime(int hour, int minute, int second)
                        {
                        this->hour = hour;
                        this->minute = minute;
                        this->second = second;
                        }
            void showTime(void)
                        {
                        cout << "Hour: " << hour << " Minute: " << minute << " Second: " << second;
                        }
};
int main()
{
            Time t;
            t.setTime(3,55,39);
            t.showTime();
}
```

OUTPUT

```
Hour: 3 Minute: 55 Second: 39
```

Passing Object as Argument (this pointer)

www.hbpatel.in

```
#include <iostream>
using namespace std;
class Time
{private: int hour, minute, second;
public:
            void readTime(void)
            {           cout << "Enter Hour : "; cin >> hour;
                        cout << "Enter Minute : "; cin >> minute;
                        cout << "Enter Second : "; cin >> second;
            }
            Time add(Time t)
            {           Time temp;
                        temp.hour = this.hour + t.hour;
                        temp.minute = this.minute + t.minute;
                        temp.second = this.second + t.second;
                        if(temp.second>=60){temp.minute++; temp.second-=60;}
                        if(temp.minute>=60){temp.hour++; temp.minute-=60;}
                        return temp;
            }
            void showTime(void)
            {cout << "Hour: " << hour << " Minute: " << minute << " Second: " << second;}
};
```

```
int main()
{
            Time t1, t2, t3;
            t1.readTime();
            t2.readTime();
            t3=t1.add(t2);
            t3.showTime();

}
```

OUTPUT

```
Enter Hour : 4
Enter Minute : 53
Enter Second : 49
Enter Hour : 8
Enter Minute : 15
Enter Second : 16
Hour: 13 Minute: 9 Second: 5
```

Reference variable

www.hbpatel.in

```
#include <iostream>
using namespace std;
int main()
{
    int x = 10;
    int& refToX = x;
    cout << &x << endl;
    cout << &refToX << endl;

    cout << "Original value of variable x = " << x << endl;
    cout << "Original value of reference to x = " << refToX << endl;

    refToX = 20;
    cout << "After Changing Reference to the variable: Value of variable x = " << x << endl;
    cout << "After Changing Reference to the variable: Value of reference to x = " << refToX << endl;

    x = 30;
    cout << "After Changing the value of the variable: Value of variable x = " << x << endl;
    cout << "After Changing the value of the variable: Value of reference to x = " << refToX << endl;
}
```

OUTPUT

```
0x6ffe04
0x6ffe04
Original value of variable x = 10
Original value of reference to x = 10
After Changing Reference to the variable: Value of variable x = 20
After Changing Reference to the variable: Value of reference to x = 20
After Changing the value of the variable: Value of variable x = 30
After Changing the value of the variable: Value of reference to x = 30
```

C++ Programming: Modules

www.hbpatel.in

- Module 1: Essentials of C Programming
- Module 2: Fundamental Concepts of OOP with C++
- Module 3: C++ Programming Syntactical Basics
- Module 4: C++ Functions
- Module 5: Objects and Class
- Module 6: Operator Overloading
- Module 7: Inheritance
- Module 8: Polymorphism & Virtual Functions
- Module 9: Templates and Exception Handling
- Module 10: Introduction to Streams and Files

Objects and Class

www.hbpatel.in

KSV Examination Questions from this unit

1. What is constructor? Explain parametrized constructor with example. [5, Jan-2023]
2. Explain constructor. Explain copy constructor and parametrized constructor with example. [5, July-2022] [5, Feb-2022] [5, Jun-2023] [5, Jan-2024]
3. Explain the use of destructor in C++. Discuss its features. [5, Jan-2023]
4. Explain the use of constructor and destructor. Explain default constructor with example. [5, July-2022]
5. Demonstrate the usage of destructor using a program. [5, Jan-2024]
6. What is inline function? Write a program to find area of a bigger circle using inline function. [5, Jan-2023]
7. What is inline function? Explain with an example. [5, July-2022] [5, Feb-2022] [5, Jun-2023]
8. What is friend function? What are the advantages and disadvantages of friend function? [5, Jan-2023] [5, Feb-2022] [5, Jun-2023]
9. What is friend function? Write a program to find out sum of two private data members a and b of two classes X and Y using common friend function. Assume that the prototype for both classes will be void sum(X, Y). [5, July-2022]

Constructor

www.hbpatel.in

Constructor is a special method that is called/invoked automatically whenever an object is created. (Unlike regular functions, you do not need to call it explicitly). It has the same name as the class. We can have multiple constructors in a class, wherein the constructors are differentiated based on the arguments passed to them. Constructor without argument is also known as default constructor.

```
#include <iostream>
using namespace std;
class Test
    {
    private:
        int data;
    public:
        Test()
            {cout << "Constructor Called\n";
            data=0;
            }
        void showData(void)
            {cout << data << endl;}
    };
```

```
int main()
    {
     Test obj;
     obj.showData();
    }
```

OUTPUT

```
Constructor Called
0
```

Constructor

www.hbpatel.in

Constructor is a special method that is called/invoked automatically whenever an object is created. (Unlike regular functions, you do not need to call it explicitly). It has the same name as the class. We can have multiple constructors in a class, wherein the constructors are differentiated based on the arguments passed to them.

```
#include <iostream>
using namespace std;
class Test
{
    private:  int data;
    public:
        Test()
            {cout << "Constructor Called - without argument \n";
            data=0;
            }
       Test(int x)
            {cout << "Constructor Called - with argument \n";
             data=x;
            }
       void showData(void)
            {cout << data << endl;}
};
```

```
int main()
{
    Test obj1, obj2(10);
    obj1.showData();
    obj2.showData();
}
```

OUTPUT

```
Constructor Called - without argument
Constructor Called - with argument
0
10
```

Constructor


```
#include <iostream>
using namespace std;
class Time
{
private: int hour, minute, second;
public:
            Time()
            {
                        cout << "Constructor called\n";
                        hour = minute = second = 0;
            }
            void setTime(int h, int m, int s)
            {
                        hour = h;
                        minute = m;
                        second = s;
            }
            void showTime(void)
            {
                        cout << "Hour: " << hour << " Minute: " << minute << " Second: " << second << endl;
            }
};
```

```
int main()
{
            Time t1, t2;
            t1.showTime();
            t2.showTime();
            t1.setTime(3,55,39);
            t1.showTime();
            t2.showTime();
}
```

OUTPUT

```
Constructor called
Constructor called
Hour: 0 Minute: 0 Second: 0
Hour: 0 Minute: 0 Second: 0
Hour: 3 Minute: 55 Second: 39
Hour: 0 Minute: 0 Second: 0
```

Destructor

Destructor is a special method that is called/invoked automatically whenever an object is destroyed. (Unlike regular functions, you do not need to call it explicitly). It has the same name as the class preceded by a tilde (~) sign. We can have only one destructor in a class. Destructor neither takes any argument nor returns anything. Destructor is used to release memory space occupied by the objects created by the constructor.

Destructor

www.hbpatel.in

```
#include <iostream>
using namespace std;
class SampleClass
{
public:
          SampleClass()
          {
                    cout << "This is a constructor\n";
          }
          ~SampleClass()
          {
                    cout << "This is a destructor\n";
          }
};

int main()
{
cout << "This is beginning of main function\n";
SampleClass s;
cout << "This is end of main function\n";
}
```

OUTPUT

```
This is beginning of main function
This is a constructor
This is end of main function
This is a destructor
```

Destructor

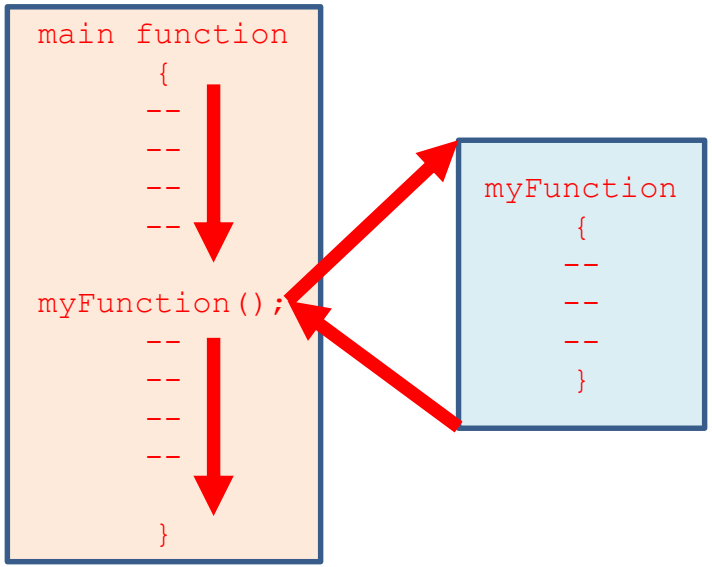
www.hbpatel.in

```
#include <iostream>
using namespace std;
static int totalObjects = 0;
class SampleClass
{
public:
            SampleClass()
            {
                        totalObjects++;
                        cout << "Object # " << totalObjects << " Created with address " << this << "\n";
            }
            ~SampleClass()
            {
                        totalObjects--;
                        cout << "Object # " << totalObjects << " Destroyed with address " << this << "\n";
            }
};
int main()
{
            cout << "This is beginning of main function\n";
            SampleClass s1, s2, s3;
            cout << "This is end of main function\n";
}
```

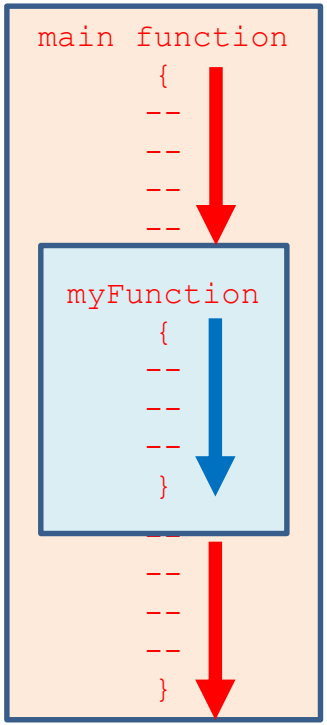
OUTPUT

```
This is beginning of main function
Object # 1 Created with address 0x6ffe0f
Object # 2 Created with address 0x6ffe0e
Object # 3 Created with address 0x6ffe0d
This is end of main function
Object # 2 Destroyed with address 0x6ffe0d
Object # 1 Destroyed with address 0x6ffe0e
Object # 0 Destroyed with address 0x6ffe0f
```

inline function



Normal function



inline function

Inline functions are used to reduce the function call overhead. When an inline function is called, the entire code of the inline function gets substituted at the point of function call. This is done by the C++ compiler at compile time to reduce the execution time.

inline function


```
#include <iostream>
using namespace std;
inline int factorial(int n)
{
        int answer=1;
        for(int i=2; i<n; ++i)answer*=i;
        return answer;
}
int main()
{
        cout << "Factorial of 5 = " << factorial(5);
}
```

OUTPUT

```
Factorial of 5 = 120
```

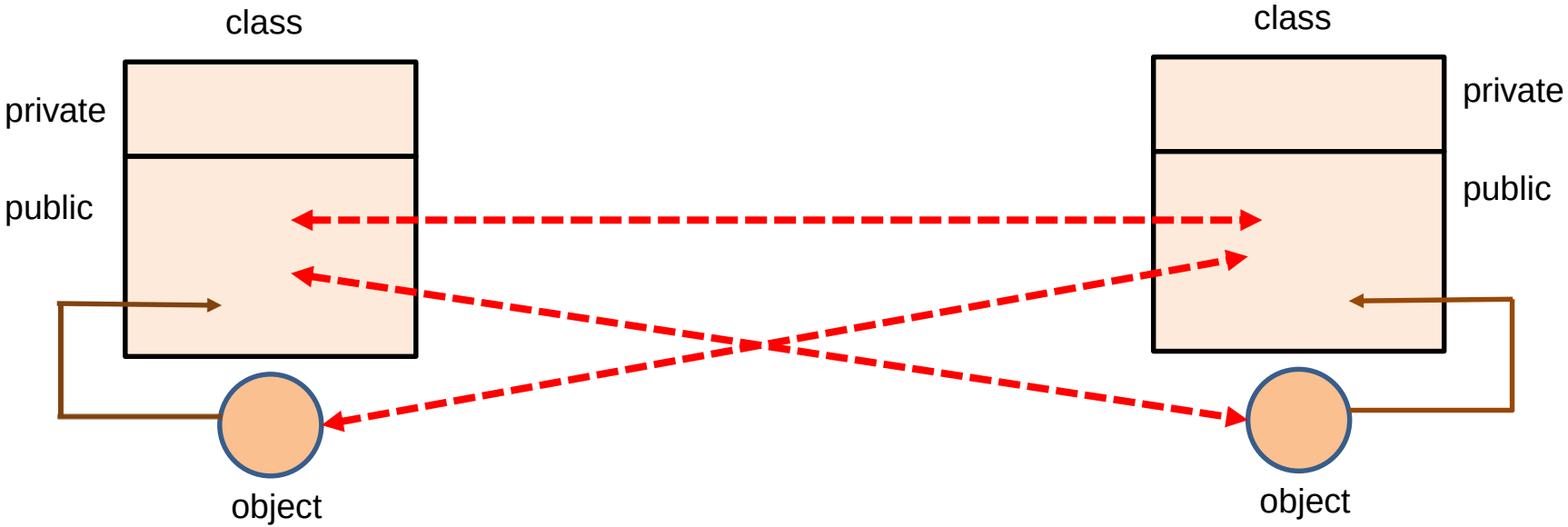
inline function


```
#include <iostream>
using namespace std;
class myMath
{
        public:
                int factorial(int);
};
inline int myMath:: factorial(int n)
{
        int answer=1;
        for(int i=2; i<=n; ++i)answer*=i;
        return answer;
}
int main()
{
        myMath obj;
        cout << "Factorial of 5 = " << obj.factorial(5);
}
```

OUTPUT

```
Factorial of 5 = 120
```

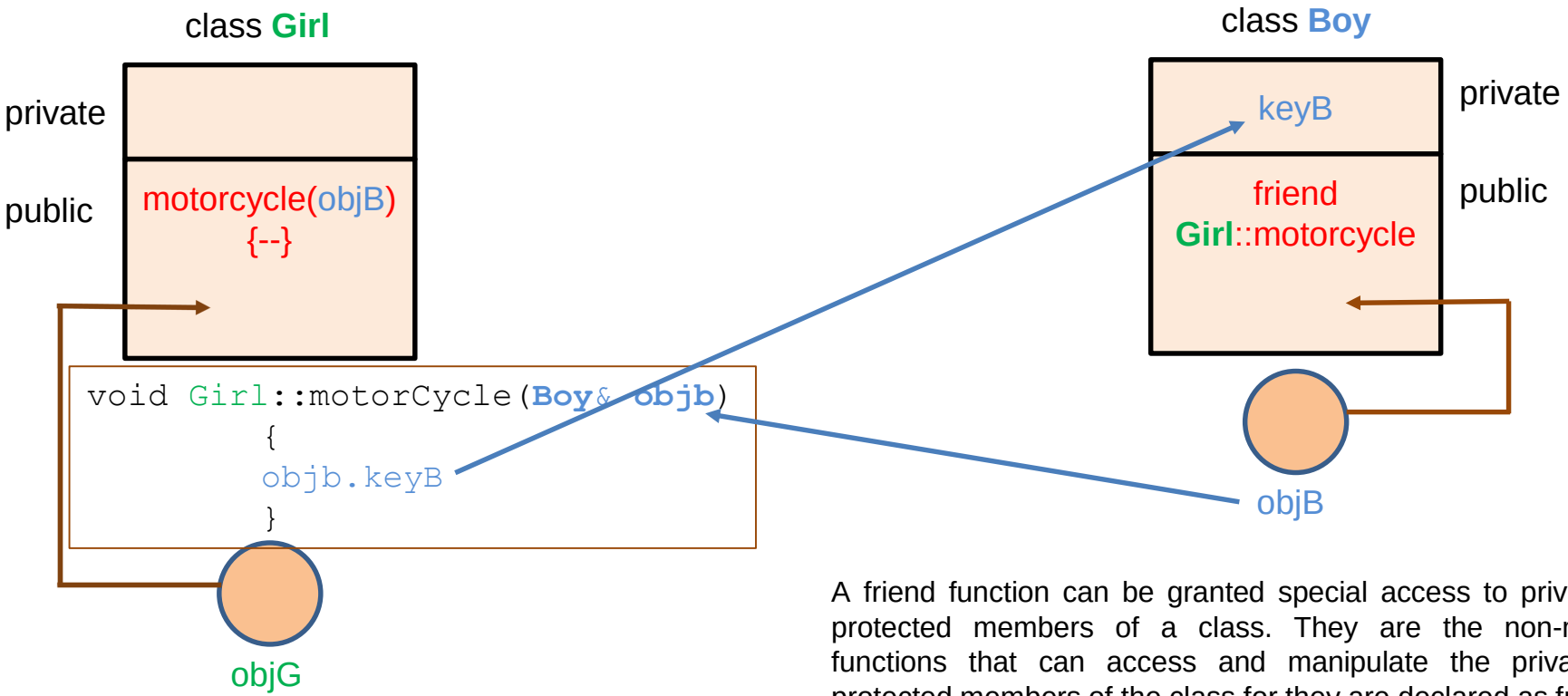
friend function



A friend function can be granted special access to private and protected members of a class. They are the non-member functions that can access and manipulate the private and protected members of the class for they are declared as friends.

friend function

www.hbpatel.in



A friend function can be granted special access to private and protected members of a class. They are the non-member functions that can access and manipulate the private and protected members of the class for they are declared as friends.

friend function


```
#include <iostream>
using namespace std;
class Boy;

class Girl
{
public: void motorCycle(Boy& obj);
};

class Boy
{
private: int keyB;
public:
        Boy(){keyB = 10;}
friend void Girl::motorCycle(Boy&);
}
```

```
void Girl::motorCycle(Boy& objb)
{
        cout << objb.keyB << endl;
}

int main()
{
        Boy objB;
        Girl objG;
        objG.motorCycle(objB);
}
```

OUTPUT

```
10
```

friend function

Write a program to find out sum of two private data members a and b of two classes X and Y using common friend function. Assume that the prototype for both classes will be void sum(X, Y).

```
#include <iostream>
using namespace std;
class Y;
class X
{
private:  int a;
public:
          void sum(X, Y);
          X(int argument) {a=argument;}
};
class Y
{
private:  int b;
public:
          Y(int argument) {b=argument;}
          friend void X :: sum(X, Y);
};
```

```
void X :: sum(X object1, Y object2)
{
          cout << object1.a + object2.b;
}

int main()
{
          X obj1(5);
          Y obj2(10);
          obj1.sum(obj1,obj2);
}
```

OUTPUT

```
15
```

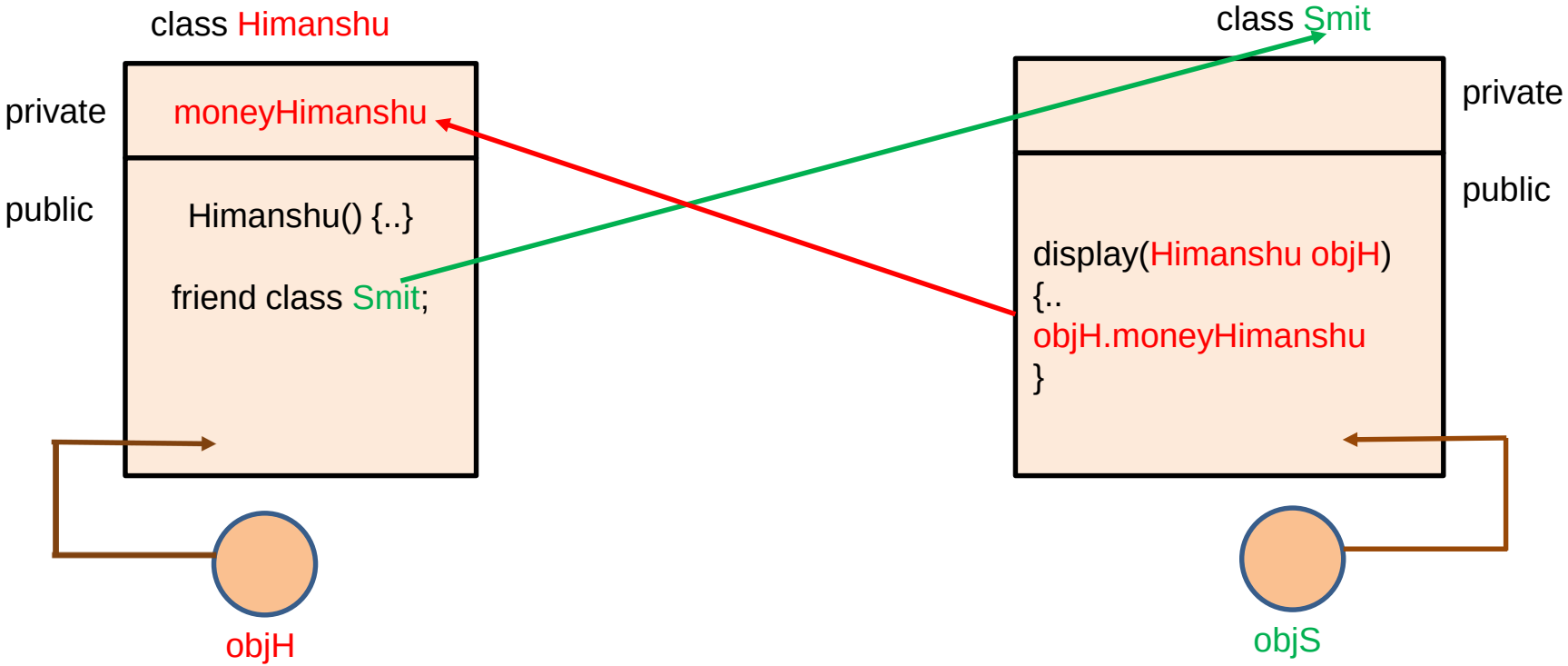
Advantage and Disadvantage of `friend` function

Advantages	Disadvantages
The declaration can be anywhere in the code	It is not passed to the derived class
There is no need to create an object to call it.	They don't have a storage-specified class.
Non-public members of the class can also be accessed using the friend function.	The friend function comes in handy when multiple classes are tied together.
It can add extra functionality.	Allows private and protected members to be shown as the information of the class.
Enables programming experience to be more efficient than ever before.	It can have both public and private protected members in the same class in which it has been defined.

Source: www.javatpoint.com

friend class

www.hbpatel.in



friend class


```
#include <iostream>
using namespace std;
class Himanshu
{
private:
        int moneyHimanshu;
public:
        Himanshu(){moneyHimanshu=1000;}
        friend class Smit;
};
class Smit
{
public:
        void display(Himanshu objh)
        {
                cout << objh.moneyHimanshu;
        }
};
```

```
int main()
{
        Himanshu objH;
        Smit objS;
        objS.display(objH);
}
```

OUTPUT

```
1000
```

C++ Programming: Modules

www.hbpatel.in

- Module 1: Essentials of C Programming
- Module 2: Fundamental Concepts of OOP with C++
- Module 3: C++ Programming Syntactical Basics
- Module 4: C++ Functions
- Module 5: Objects and Class
- Module 6: Operator Overloading
- Module 7: Inheritance
- Module 8: Polymorphism & Virtual Functions
- Module 9: Templates and Exception Handling
- Module 10: Introduction to Streams and Files

Operator Overloading

www.hbpatel.in

KSV Examination Questions from this unit

1. What is an operator overloading? Write a program to overload binary + operator as a member function. [5, Jan-2023] [5, July-2022] [5, Jun-2023]
2. Write down the example to overload unary and binary operator in C++. [5, Feb-2022]
3. Explain the pitfalls of Operator Overloading. [5, Jan-2024]
4. Define a class complex with real and imaginary as two data member, use necessary constructors and member function to initialize and display data of class. Class should overload the ' -' operator to subtract two complex objects and return the results. Invoke the statements like C3=C l -C2 in main() function. [5, Jan-2024]

Operator Overloading


```
#include <iostream>
using namespace std;
int main()
{
        int x=10, y=20, z;
        z = x + y;
        cout << z;
}
```

Regular operator (E.g. + here) work on built-in data types (E.g. int here)

```
#include <iostream>
using namespace std;
class SampleClass
{
private: int data;
public: SampleClass(int d) {data=d;}
}
int main()
{
SampleClass obj1(10), obj2(20), obj3(0);
obj3=obj1+obj2;
}
```

Regular operator (E.g. + here) does **not** work on user-defined data types (E.g. object here)

```
[Error] no match for 'operator+' (operand types are 'SampleClass' and 'SampleClass')
```

Operator Overloading


```
#include <iostream>
using namespace std;
class SampleClass
{
private:
          int data;
public:
          SampleClass() {data=0;}
          SampleClass(int d) {data=d;}
          SampleClass operator + (SampleClass obj)
          {
                    SampleClass temp;
                    temp.data = data + obj.data;
                    return temp;
          }
          void showData()
          {
                    cout << data;
          }
};
```

```
int main()
{
SampleClass obj1(10), obj2(20), obj3;
obj3=obj1+obj2;
obj3.showData();
}
```

OUTPUT

```
30
```

Operator Overloading


```
#include <iostream>
using namespace std;
class SampleClass
{
private: int data;
public:
            SampleClass() {data=0;}
            SampleClass(int d) {data=d;}
            SampleClass operator + (SampleClass &argument)
            {
            cout << "Address of calling object = " << this << endl;
            cout << "Address of argument object = " <<  &argument << endl;
            SampleClass temp;
            temp.data = data + argument.data;
            return temp;
            }
            void showData() {cout << data;}
};
```

```
int main()
{
SampleClass obj1(10), obj2(20), obj3;
cout << "Address of obj1 = " << &obj1 << endl;
cout << "Address of obj2 = " << &obj2 << endl;
cout << "Address of obj3 = " << &obj3 << endl;
obj3 = obj1 + obj2;
obj3.showData();
}
```

OUTPUT

```
Address of obj1 = 0x6ffe10
Address of obj2 = 0x6ffe00
Address of obj3 = 0x6ffdf0
Address of calling object = 0x6ffe10
Address of argument object = 0x6ffe00
30
```

Operator Overloading

www.hbpatel.in

Define a class complex with real and imaginary as two data member, use necessary constructors and member function to initialize and display data of class. Class should overload the ' -' operator to subtract two complex objects and return the results. Invoke the statements like C3=C l -C2 in main() function.

```
#include <iostream>
using namespace std;
class Complex
{
private: int realData, imaginaryData;
public:
Complex(){realData=0; imaginaryData=0;}
Complex(int r, int i){realData=r; imaginaryData=i;}
Complex operator - (Complex paramterObject)
            {
            Complex returnObject;
            returnObject.realData = realData - paramterObject.realData;
            returnObject.imaginaryData = imaginaryData - paramterObject.imaginaryData;
            return returnObject;
            }
void showComplex()
            {cout << "Real data = " << realData << " Imaginary Data = " << imaginaryData << endl;
            }
};
```

```
int main()
{
Complex C1(20,40), C2(7, 11), C3;
C3 = C1 - C2;
C3.showComplex();
}
```

OUTPUT

```
Real data = 13 Imaginary Data = 29
```

Pros and Cons of Operator Overloading

Pros:

- Programmers can utilize notation more closely related to the target domain thanks to operator overloading.
- They offer comparable support for user-defined types as built-in types do.
- Operator overloading facilitates program understanding.

Cons:

- There are certain exceptions to the rule of operator overloading, which applies to all existing C++ operations.

Source: herovired.com

C++ Programming: Modules

www.hbpatel.in

- Module 1: Essentials of C Programming
- Module 2: Fundamental Concepts of OOP with C++
- Module 3: C++ Programming Syntactical Basics
- Module 4: C++ Functions
- Module 5: Objects and Class
- Module 6: Operator Overloading
- Module 7: Inheritance
- Module 8: Polymorphism & Virtual Functions
- Module 9: Templates and Exception Handling
- Module 10: Introduction to Streams and Files

Inheritance

www.hbpatel.in

KSV Examination Questions from this unit

1. Explain the importance of inheritance. List its types and explain anyone with an example. [5, Jan-2023]
2. Define inheritance. Write the types of inheritance. Explain inheritance with example. [5, Feb-2022] Consider example with respect to print result of Student and Student details. [5, July-2022] [5, Jun-2023]
3. What is the meaning of Inheritance? State the advantages of using Inheritance concept in a program. [5, Jan-2024]
4. Define multi-path inheritance with example. [5, July-2022]
5. Explain abstract class with example. [5, Jan-2023] [5, Jun-2023]
6. What is the meaning of Abstract Class? Explain it in detail.[5, Jan-2024]
7. Explain late binding and abstract class with example. [5, July-2022] [5, Feb-2022]
8. Explain overriding member function with example. [5, Jun-2023]
9. Write a program having the demonstration of the Method Overriding concept. [5, Jan-2024]
10. Explain overridden/overriding function with example. [5, Jan-2023] [5, July-2022] [5, Feb-2022] [5, Jun-2023]

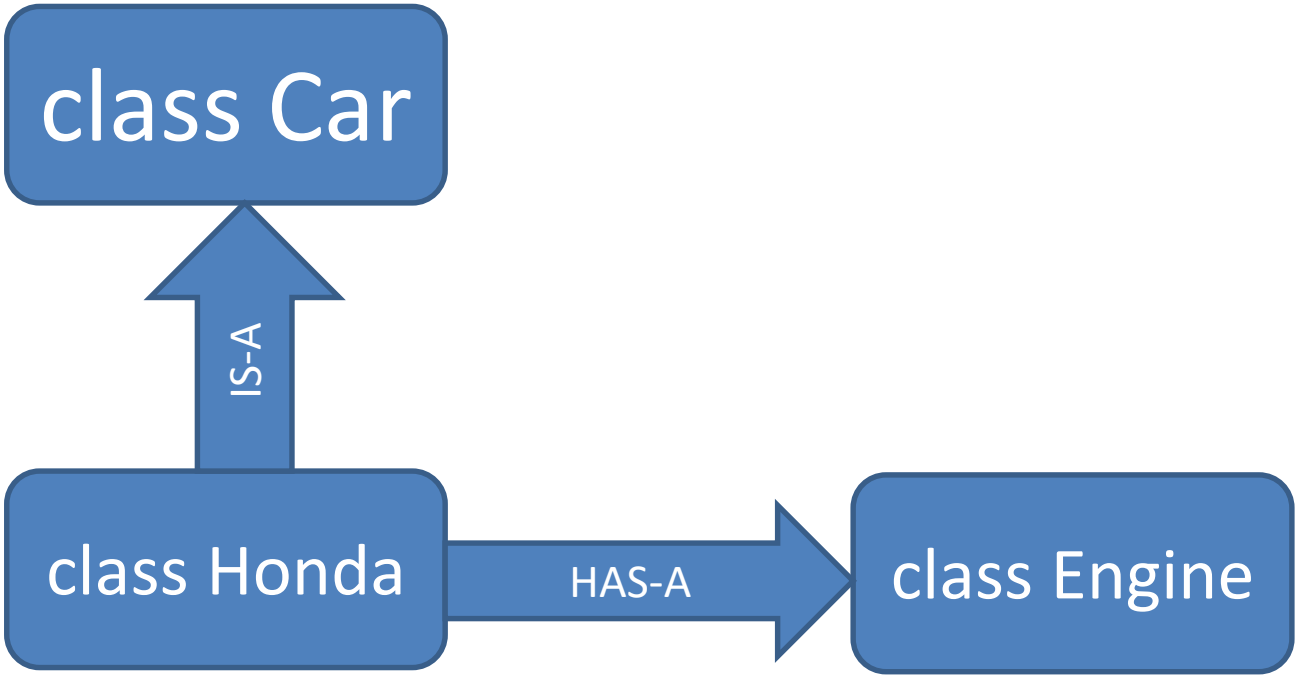
Inheritance

Inheritance is a mechanism in which one object acquires all the properties and behaviours of a parent object. It is an important part of OOP.

The idea behind inheritance is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can **reuse** functions and data of the parent class. Moreover, you can add new functions and data in your current class also.

Inheritance represents the IS-A relationship (shown in next slide) which is also known as a parent-child relationship.

Inheritance



Advantages of Inheritance

- Reusability: Inheritance helps the code to be reused in many situations. The base class is defined and once it is compiled, it needs not be reworked. Using the concept of inheritance, the programmer can create as many derived classes from the base class as needed while adding specific features to each derived class as required.

- Save time and efforts: The above concept of reusability achieved by inheritance saves the programmer time and effort. Since the main code written can be reused in various situation as needed.

- Data hiding: The base class can decide to keep some data private so that it cannot be altered by the derived class.

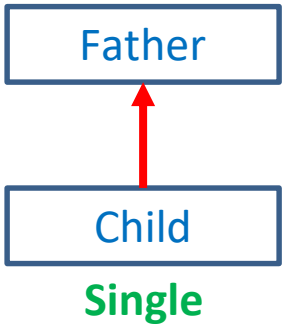
- Reliability: It increases program structure which result in greater reliability.

- Maintainability: It is easy to debug a program when divided in parts. Inheritance provides an opportunity to capture the program.

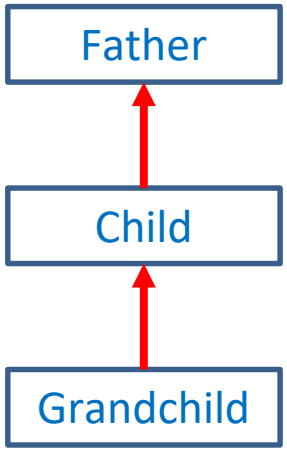
Source: www.scribd.com

Inheritance

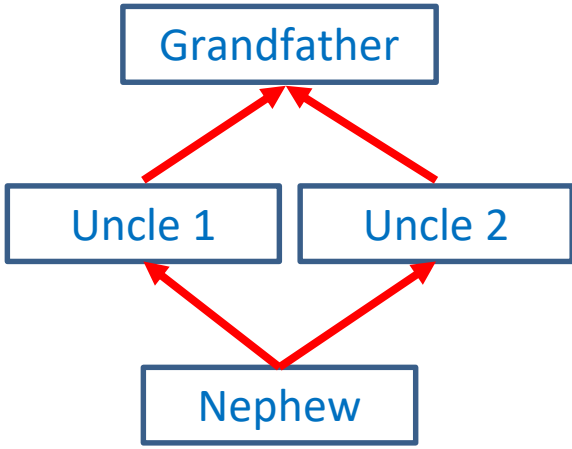
www.hbpatel.in



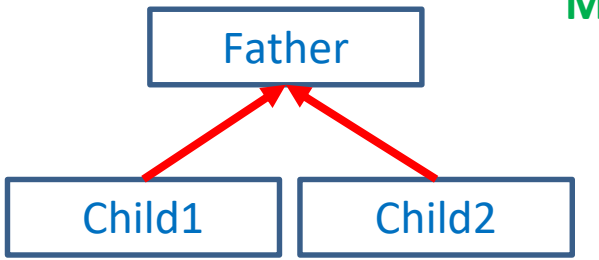
Single



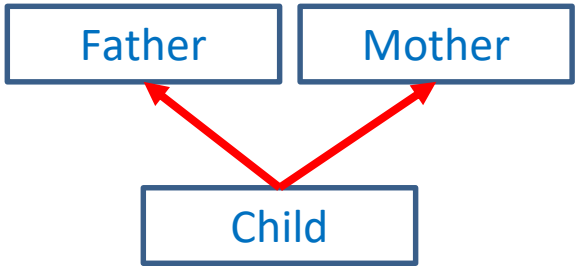
Multilevel



Hybrid



Hierarchical



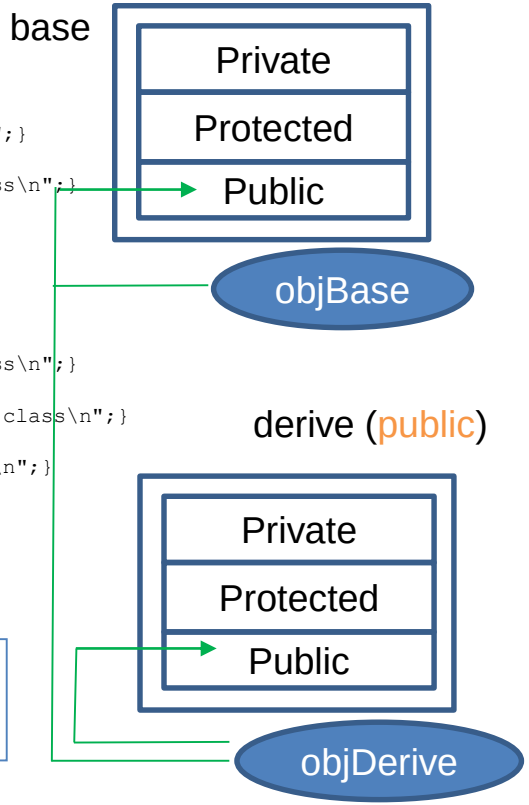
Multiple

Access Specifiers: Inheritance

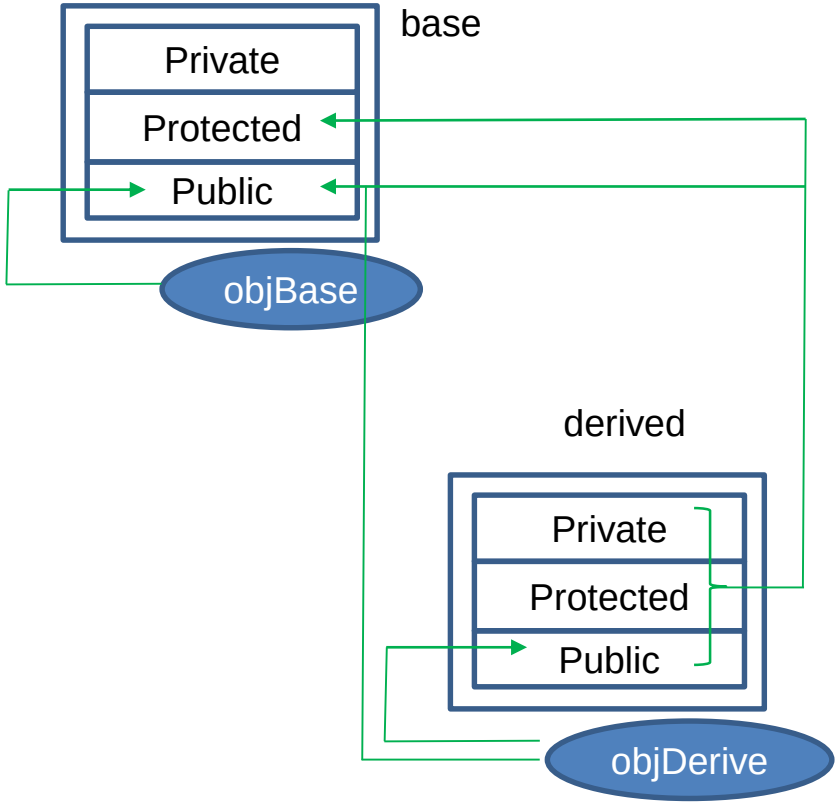

```
#include <iostream>
using namespace std;
class base
{
private:
                void showBasePrivate() {cout << "This is show function in private section of base class\n";}
protected:
                void showBaseProtected() {cout << "This is show function in protected section of base class\n";}
public:
                void showBasePublic() {cout << "This is show function in public section of base class\n";}
};
class derive : public base
{
private:
                void showDerivePrivate() {cout << "This is show function in private section of derive class\n";}
protected:
                void showDeriveProtected() {cout << "This is show function in protected section of derive class\n";}
public:
                void showDerivePublic() {cout << "This is show function in public section of derive class\n";}
};
int main(void)
{
base objBase;
derive objDerive;

objBase.showBasePublic();
objDerive.showBasePublic();
objDerive.showDerivePublic();
}
```

OUTPUT

```
This is show function in public section of base class
This is show function in public section of base class
This is show function in public section of derive class
```

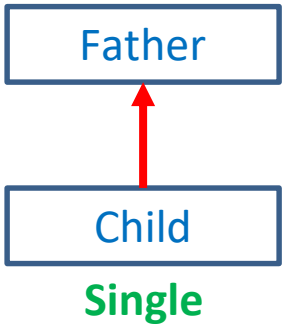


Access Specifiers: Inheritance

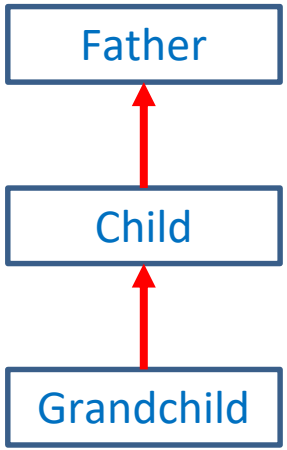


Inheritance

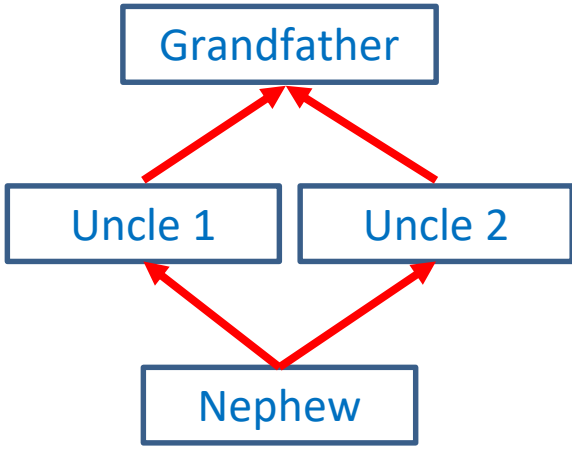
www.hbpatel.in



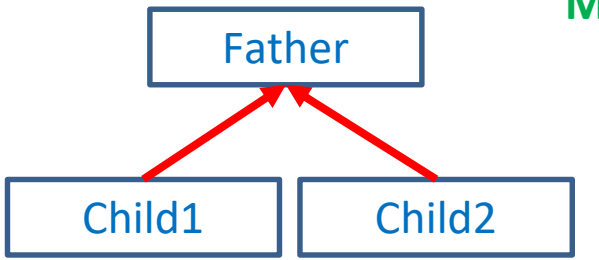
Single



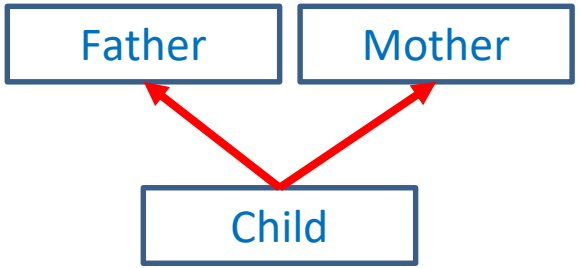
Multilevel



Hybrid



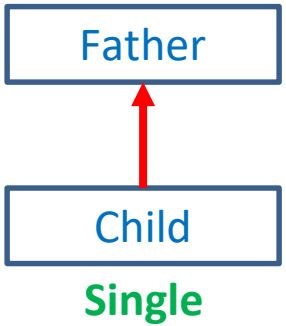
Hierarchical



Multiple

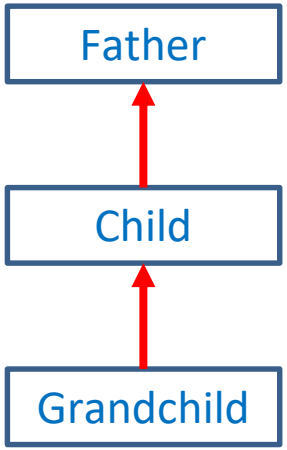
Single Inheritance

www.hbpatel.in



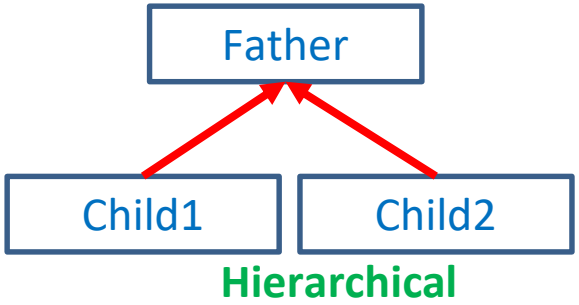

```
#include <iostream>
using namespace std;
class Father
{
private:    int Fmoney;
public:     void setMoneyF(int f){Fmoney=f;}
                        void showMoneyF(){cout << "Father Money : " << Fmoney << endl;}
};
class Son : public Father
{
private:    int Smoney;
public:     void setMoneyS(int s, int f)
                        {Smoney=s;
                        Father::setMoneyF(f);
                        }
            void showMoneyS()
                        {Father::showMoneyF();
                        cout << "Son Money : " << Smoney << endl;
                        }
};
int main(void)
{
Son object;
object.setMoneyS(5000, 80000);
object.showMoneyS();
}
```

Multilevel Inheritance

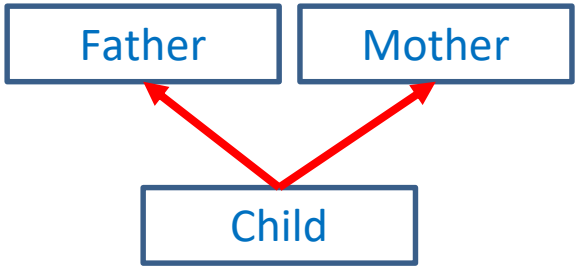


Multilevel

Hierarchical Inheritance

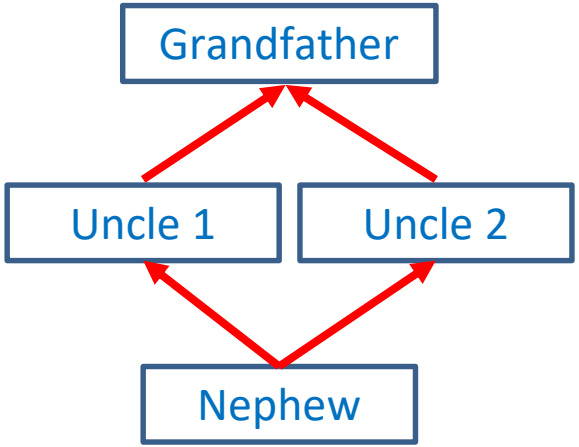


Multiple Inheritance



Multiple

Hybrid Inheritance



Hybrid

C++ Programming: Modules

www.hbpatel.in

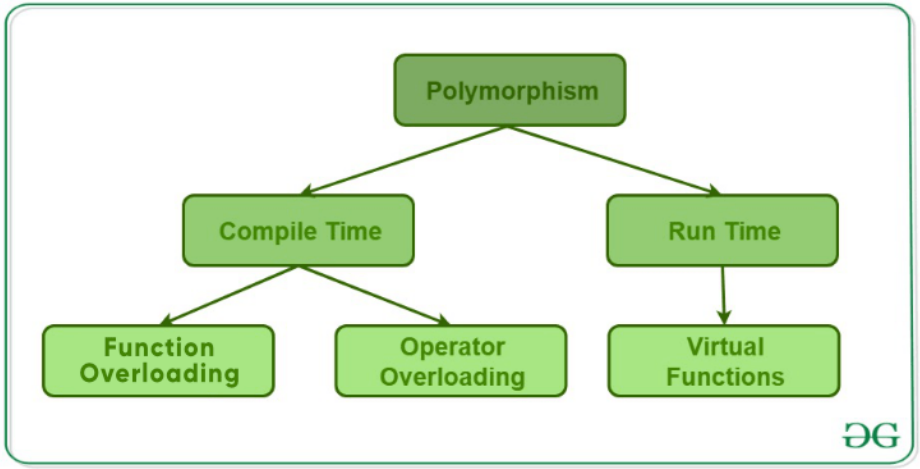
- Module 1: Essentials of C Programming
- Module 2: Fundamental Concepts of OOP with C++
- Module 3: C++ Programming Syntactical Basics
- Module 4: C++ Functions
- Module 5: Objects and Class
- Module 6: Operator Overloading
- Module 7: Inheritance
- Module 8: Polymorphism & Virtual Functions
- Module 9: Templates and Exception Handling
- Module 10: Introduction to Streams and Files

Polymorphism & Virtual Functions

KSV Examination Questions from this unit

1. Difference between virtual and pure virtual function. [5, Jan-2023] [5, July-2022] [5, Jun-2023]
2. Define virtual base class. [5, Jan-2023] [5, Feb-2022]
3. Explain polymorphism in C++. Explain compile time and run time polymorphism. [5, Jan-2023] [5, July-2022]
4. How does C++ use the concept of reusability? Write a program in C++ to illustrate use of polymorphism. [5, Feb-2022]
5. Explain runtime polymorphism. Explain and demonstrate, how virtual function to achieve runtime polymorphism. [5, Jun-2023]
6. Explain concept of virtual functions with an example. [5, Jan-2024]

Polymorphism



The word “polymorphism” means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Source: www.geeksforgeeks.org

Polymorphism
Function Overloading

www.hbpatel.in

```
#include <iostream>
using namespace std;
class polymorphismFunctionOverloading
{
public:
          void function(void) {cout << "This is a function without argument\n";}
          void function(int a) {cout << "This is a function with one argument\n";}
          void function(int a, int b) {cout << "This is a function two arguments\n";}
};
int main()
{
          polymorphismFunctionOverloading object;
          object.function();
          object.function(5);
          object.function(5,10);
}
```

OUTPUT

```
This is a function without argument
This is a function with one argument
This is a function two arguments
```

Polymorphism
Function Overriding

www.hbpatel.in

```
#include <iostream>
using namespace std;
class person
{
private:        int code;
public:         person(int c) {code=c;}
                void show() {cout << "Code = " << code << ", ";}
};

class student : public person
{
private:        int marks;
public:         student(int c, int m) : person(c)
                                {marks = m;}
                void show()
                                {person::show();
                                cout << "Marks = " << marks << endl;
                                }
};

class employee : public person
{
private:        int salary;
public:
                employee(int c, int s): person(c)
                {salary = s;}
                void show()
                                {person::show();
                                cout << "Salary = " << salary << endl;
                                }
};
```

```
int main()
{
student s(100, 79);
employee e(101, 10000);
s.show();
e.show();
}
```

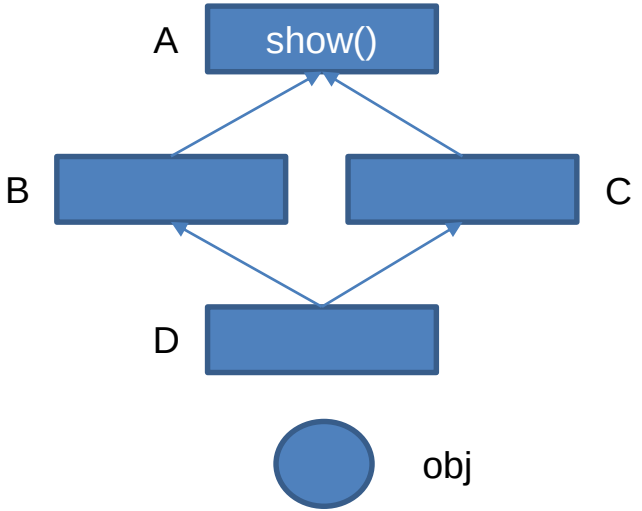
OUTPUT

```
Code = 100, Marks = 79
Code = 101, Salary = 10000
```

Virtual Class
(Why do we need?)

www.hbpatel.in

```
#include <iostream>
using namespace std;
class A
{
public: void show()
          {cout << "This is a show function in class A\n";}
};

class B: public A {};

class C: public A {};

class D: public B, public C {};

int main()
{
          D obj;
          obj.show();
}
```



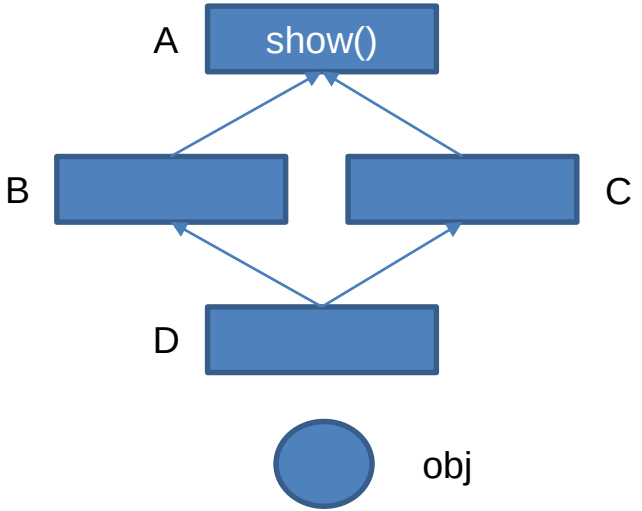
Error

Request for member 'show' is ambiguous

Virtual Class

www.hbpatel.in

```
#include <iostream>
using namespace std;
class A
{
public: void show()
          {cout << "This is a show function in class A\n";}
};

class B: virtual public A {};

class C: public virtual A {};

class D: public B, public C {};

int main()
{
          D obj;
          obj.show();
}
```



Output

```
This is a show function in class A
```

Virtual Function


```
#include <iostream>
using namespace std;
class A
{
          public: virtual void show() {cout << "This is a show function in class A\n";};
};
class B: public A
{
          public: void show() {cout << "This is a show function in class B\n";}
};
class C: public A
{
          public: void show() {cout << "This is a show function in class C\n";}
};
int main()
{         B objb;
          C objc;
          objb.show();
          objc.show();
}
```

Output

```
This is a show function in class B
This is a show function in class C
```

Pure Virtual Function

www.hbpatel.in

```
#include <iostream>
using namespace std;
class A
{
          public: virtual void show() = 0;
};
class B: public A
{
          public: void show() {cout << "This is a show function in class B\n";}
};
class C: public A
{
          public: void show() {cout << "This is a show function in class C\n";}
};
int main()
{         B objb;
          C objc;
          objb.show();
          objc.show();
}
```

Output

```
This is a show function in class B
This is a show function in class C
```

Virtual Vs. Pure Virtual Function

www.hbpatel.in

Virtual function	Pure virtual function
A virtual function is a member function of base class which can be redefined by derived class.	A pure virtual function is a member function of base class whose only declaration is provided in base class and should be defined in derived class otherwise derived class also becomes abstract.
Classes having virtual functions are not abstract.	Base class containing pure virtual function becomes abstract.

Syntax:
`virtual <func_type> <func_name>()`
`{// code}`
	Syntax:
	`virtual <func_type> <func_name>() = 0;`

Definition is given in base class.	No definition is given in base class.
Base class having virtual function can be instantiated i.e. its object can be made.	Base class having pure virtual function becomes abstract i.e. it cannot be instantiated.
If derived class do not redefine virtual function of base class, then it does not affect compilation.	If derived class do not redefine virtual function of base class, then no compilation error but derived class also becomes abstract just like the base class.
All derived class may or may not redefine virtual function of base class.	All derived class must redefine pure virtual function of base class otherwise derived class also becomes abstract just like base class.

Source: www.geeksforgeeks.org

Compile Time Polymorphism


```
#include <iostream>
using namespace std;
class Animal
{
public:
    virtual void display(){cout << "Virtual Display Function in Animal Class\n";}
    void show(){cout << "Regular Show Function in Animal Class\n";}
};

class Tiger : public Animal
{
public:
            void display() {cout << "Regular Display Function in Tiger Class\n";}
            void show(){cout << "Regular Show Function in Tiger Class\n";}
};

int main()
{
Animal a;
Tiger t;
a.display();
a.show();
t.display();
t.show();
}
```

Output

```
Virtual Display Function in Animal Class
Regular Show Function in Animal Class
Regular Display Function in Tiger Class
Regular Show Function in Tiger Class
```

Run Time Polymorphism


```
#include <iostream>
using namespace std;
class Animal
{
public:
    virtual void display(){cout << "Virtual Display Function in Animal Class\n";}
    void show(){cout << "Regular Show Function in Animal Class\n";}
};

class Tiger : public Animal
{
public:
            void display() {cout << "Regular Display Function in Tiger Class\n";}
            void show(){cout << "Regular Show Function in Tiger Class\n";}
};

int main()
{

Animal *a;
Tiger t;
a = &t;
a->display();
a->show();
}
```

Output

```
Regular Display Function in Tiger Class
Regular Show Function in Animal Class
```

Compile Time Vs. Run Time Polymorphism

www.hbpatel.in

COMPILE-TIME	RUN-TIME
Compile-time polymorphism is also known as static or early binding polymorphism.	Run-time polymorphism is also known as dynamic or late binding polymorphism.
The function calls are resolved by the compiler.	The function calls are not resolved by the compiler.
Compile-time polymorphism provides less flexibility to the programmers since everything is executed during compilation.	In contrast, run-time polymorphism is more flexible since everything is executed during run-time.
It can be implemented through function overloading and operator overloading.	It can be implemented through virtual functions and function overriding.
Method overloading is an application of compile-time polymorphism where the same name can be commissioned between more than one method of functions having different arguments or signatures and the same return types.	Method overriding is an application of run time polymorphism where two or more functions with the same name, arguments, and return type accompany different classes of the same structure.
This method has a much faster execution rate since all the methods that need to be executed are called during compile time.	This method has a comparatively slower execution rate since all the methods that need to be executed are called during the run time.
This method is less preferred for handling compound problems since all the methods and details come to light only during the compile time.	This method is known to be better for dealing with compound problems since all the methods and details turn up during the run time.

Source: www.simplilearn.com

C++ Programming: Modules

www.hbpatel.in

- Module 1: Essentials of C Programming
- Module 2: Fundamental Concepts of OOP with C++
- Module 3: C++ Programming Syntactical Basics
- Module 4: C++ Functions
- Module 5: Objects and Class
- Module 6: Operator Overloading
- Module 7: Inheritance
- Module 8: Polymorphism & Virtual Functions
- Module 9: Templates and Exception Handling
- Module 10: Introduction to Streams and Files

Templates and Exception Handling

www.hbpatel.in

KSV Examination Questions from this unit

1. What are the three keywords for exception handling? Explain these three keywords in details. [5, Jan-2023] Explain try, catch and throw exception handling in C++. [5, Feb-2022]
2. Explain exception handling with example. [5, July-2022]
3. What is exception? Demonstrate try…catch block with example. [5, Jun-2023]
4. Explain with an example, why templates are used in programming? [5, Jan-2023]
5. What is the purpose of using template in C++? Explain template with function and template class with example. [5, July-2022] [5, Feb-2022]
6. Explain function and class templates with appropriate example. [5, Jun-2023]
7. Write a C++ program to handle exception “divide by zero” situation. [5, Jun-2023]
8. Define Template. Write a program to define the function template for calculating the cube of given numbers with different data types. [5, Jan-2024]
9. Explain Exception handling in detail. [5, Jan-2024]

C++ Programming: Modules

www.hbpatel.in

- Module 1: Essentials of C Programming
- Module 2: Fundamental Concepts of OOP with C++
- Module 3: C++ Programming Syntactical Basics
- Module 4: C++ Functions
- Module 5: Objects and Class
- Module 6: Operator Overloading
- Module 7: Inheritance
- Module 8: Polymorphism & Virtual Functions
- Module 9: Templates and Exception Handling
- Module 10: Introduction to Streams and Files

Introduction to Streams and Files

www.hbpatel.in

KSV Examination Questions from this unit

1. What is stream class? Describe various stream classes for console I/O operation. [5, Jan-2023]
2. What is stream class? Explain ifstream, ofstream and fstream class.
3. Explain file handling in C++. [5, Jan-2023]
4. Explain file I/O with stream with one example. [5, July-2022]
5. Explain various file stream classes available for file operations. [5, Feb-2022]
1. Describe various stream classes for console I/O operators. [5, Jun-2023]
2. Explain stream errors in detail. [5, Jan-2024]
3. Explain stream classes and its hierarchy.[5, Jan-2024]