9... Templates

Objectives...

- To study Class Template and Class Template with multiple parameters.
- To learn about Function Template and Function Template with multiple parameters.
- To understand Exception Handling.

9.1 INTRODUCTION

- Templates are a feature of the C++ programming language that allows functions and classes to operate with generic types. This allows a function or class to work on many different data types without being rewritten for each one.
- Templates are a way of making your classes more abstract by letting you define the behavior of the class without actually knowing what datatype will be handled by the operations of the class.
- Templates can be used in conjunction with abstract datatypes in order to allow them to handle any type of data.
- For example, you could make a templated stack class that can handle a stack of any datatype, rather than having to create a stack class for every different datatype for which you want the stack to function.
- The ability to have a single class that can handle several different datatypes means the code is easier to maintain, and it makes classes more reusable.
- **The basic syntax for declaring a templated class is as follows:**

```
template <class a_type> class a_class {...};
```

Where,

'class': The keyword 'class' means that the identifier.

a_type: It will stand for a datatype.

- The templates declared for functions are called as **function templates** and the templates declared for classes are called as **class templates.**
- C++ templates allows to reuse the software components such as functions, classes etc. and also make the source code compact.

- A function template behaves like a function except that the template can have arguments of many different types.
- There are two kinds of templates one is function template and another is class template.
- A class template provides a specification for generating classes based on parameters. Class templates are commonly used to implement containers.
- A class template is instantiated by passing a given set of types to it as template arguments.

9.2 CLASS TEMPLATES [S-18, 22, 23, W-18]

- Like function templates, class templates are also used to declare to operate on different data types.
- C++ class templates, creates a class which contain one or more generic data types.
- **The syntax for creating class template is:**

```
template <class T>
class classname
{
    ------------------
    ------------------
}
```

Here, class data items and functions arguments are of template type.

- **For example:**

```
template<class T>
class arr
{
    T a[10];  //This is used to change to any data type
    int i;
    public:
        arr ();
        void add (const T &ele);
        T remove (void);
        void show();
};
```

- Normally, without template we declare class of char array as:

```
class arr
{
    char a[10];
    int i;
    public:
        arr();
        void add (const char & ele);
        char remove (void);
        void show();
}
```

- Here, we can use any data type instead of **char** and writing again one or more classes for respective data type to declare an array. If we write **int** then it becomes array of integer. Drawback is we are writing more number of classes. So we can have class template as shown in above example.
- In the main( ), we can invoke the class template using object as:

```
classname <data type> objectname;
```

Program 9.1: Program for class template.

```
#include <iostream>
using namespace std;
template <class T1>
class mypair
{
    T1 Y, Z;
    public:
        mypair (T1 first, T1 second)
        {
            Y=first;
            Z=second;
        }
        T1 getmax ();
};
template <class T1>
T1 mypair<T1>::getmax ()
{
    T1 retval;
    retval = Y>Z? Y: Z;
    return retval;
}
int main ()
{
    mypair <int> myobject (100, 75);
    cout << myobject.getmax();
    return 0;
}
```

Output:

```
100
```

9.2.1 Member Function Templates

- We can define member function of a template class outside the class by the following syntax:

```
template<class template type>
returntype classname <tempdata type>::function_name
(arguments)
{
    ------------
};
```

Program 9.2: Program for member function template.

```
//Use of class template to multiply two numbers of different data types
#include<iostream.h>
template<class T>
class mult
{
    T result, a, b;
    public:
        void getdata();
        void mresult();
};
template<class T>
void mult <T>::getdata()
{
    cout<<"Enter two values";
    cin>>a>>b;
}
template<class T>
void mult <T>:: mresult()
{
    result = a * b;
    cout<<"the multiplication is: "<<result<<endl;
}
void main()
{
    mult<int>01;
    mult<float>02;
    mult<long>03;
    01.getdata();
    01.mresult();
```

```
        02.getdata();
        02.mresult();
        03.getdata();
        03.mresult();
    }
```

Output:

```
Enter two values
10        20
the multiplication is: 200
Enter two values
10.5      1.5
the multiplication is: 15.75
Enter two values
3276      25
the multiplication is: 81900
```

9.2.2 Class Templates with Multiple Parameters

- The declaration of template classes with multiple parameters is similar to the function template with multiple parameters. However, all the parameters need not be of template type. We can have any type of arguments. The syntax of class template with multiple parameters is shown below.

Syntax:

```
template<class T1, class T2, ……>
class classname
{
    ……
    ……    //body of class
};
```

Program 9.3: Use of multiple arguments in class template.

```
#include <iostream>
using namespace std;
template <class T>
T GetMax (T a, T b)
{
  T result;
  result = (a>b)? a: b;
  return (result);
}
```

```
int main ()
{
  int i=5, j=6, k;
  long l=10, m=5, n;
  k=GetMax<int>(i,j);
  n=GetMax<long>(l,m);
  cout << k << endl;
  cout << n << endl;
  return 0;
}
```

Output:

```
6
10
```

9.3 FUNCTION TEMPLATES [W-18]

- C++ provides a facility for function templates that allow you to write one function which is act as template for a number of functions which are performing similar tasks.
- A function template will not use actual type of arguments but it uses generic type as a placeholder implicit when we make a call to a function. For each version of data type, the compiler generates a copy of that function. For example: when integer version is called, the compiler generates "**add**" function with integer data type and so on.
- **The general syntax is:**

```
template <class template_data_type>
return_type func_name (arguments)                    //template function
{
   ----------
}
```

- The function template is preceded by keyword **template** and a list of template type arguments, (generic data types). The template function uses variables whose data types are known only when a call is made.

Program 9.4: Write a function template to find the biggest of two numbers. [W-18]

```
#include<iostream.h>
#include<conio.h>
template<class T>
T big (T a, T b)
{
   if (a>b)
   return (a);
   else return (b);
}
```

```
int main()
{
    cout<<"\n bigger no. is:"<<big(20, 15)<<endl;
    cout<<"\n bigger no. is:"<<big(10.5, 20.5)<<endl;
    cout<<"\n bigger is:"<<big('A', 'M')<<endl;
}
```

Output:

```
bigger no. is: 20
bigger no. is: 20.5
bigger is: M
```

- In the above Program 9.4, the function big( ) is function template having T is a generic type which is used as a placeholder. The **big( )** is called using three different data types int, float and char. The compiler creates three version of big( ), one for each data type.

Program 9.5: Swapping using function template.

```
#include<iostream>
using namespace std;
template<class X>
void Swap(X &a, X &b)          //by reference
{
    X temp = a;
    a = b;
    b = temp;
}
int main()
{
    int x, y;
    cout<<"Enter x & y before swapping"<<endl;
    cin>>x>>y;
    Swap(x, y);
    cout<<"after swapping:"<<x<<" "<<y;
    char c1, c2;
    cin>>c1>>c2;
    Swap (c1, c2);
    cout<<"after swapping"<<c1<<" "<<c2;
}
```

Output:

```
Enter x and y before swapping
10    20
after swapping:  20    10
```

9.3.1 Generic Function

- Using a generic function a single general procedure can be applied to a wide range of data.
- A genetic function defines a general set of operations that will be applied to various types of data.
- A generic function is created using the template keyword. The general **syntax** of a template function definition is:

```
template<class T>
ret_type func_name(para_list)
{
    .....
    .....     //body of function
}
```

- A specific instance of a generic function is called as a generated function.
- When we only need a type parameter for a specific function rather than an entire class, it is possible to use a generic function. A function template is a generic function description i.e., it defines a function in terms of a generic type for which a specific type, such as int.
- The following program creates a generic function that swaps the values of the two variables with which it is called.

Program 9.6: Program for generic function.

```
#include<iostream>
using namespace std;
template<class Y> void Swap (Y &m, Y &n)
{
    Y temp;
    temp = m;
    m = n;
    n = temp;
}
int main()
{
    int i = 20, j = 30;
    double x = 20.1, y = 33.3;
    char m = 'x', n = 'z';
    cout<<"Original i, j: "<< i <<' '<< j <<'\n';
    cout<<"Original x, y: "<< x <<' '<< y <<'\n';
    cout<<"Original m, n: "<< m <<' '<< n <<'\n';
    Swap(i, j);
```

```
    Swap(x, y);
    Swap(m, n);
    cout<<"Swapped i, j: "<< i <<' '<< j <<'\n';
    cout<<"Swapped x, y: "<< x <<' '<< y <<'\n';
    cout<<"Swapped m, n: "<< m <<' '<< n <<'\n';
    return 0;
}
```

Output:

```
Original i, j: 20 30
Original x, y: 20.1 33.3
Original m, n: x z
Swapped i, j: 30 20
Swapped x, y: 33.3 20.1
Swapped m, n: z x
```

9.3.2 Function Templates with Multiple Parameters

- We can use more than one generic class or generic data type. The syntax for function templates or generic functions with multiple parameter is shown below:

```
template <class T1, class T2, …..>
return_type function_name (arguments of types T1, T2, ……)
{
    ..............
    ..............        //body of function
}
```

The program 9.7 shows the two generic types functions.

Program 9.7: Program for two generic types' functions.

```
#include<iostream>
using namespace std;
template<class T1, class T2>
void display (T1 a, T2 b)
{
    cout<<a<<"  "<<b<<endl;
}
int main()
{
    display (10, 'A');
    display(10.5, 3456);
    return 0;
}
```

Output:

```
10 A
10.5   3456
```

- In this example, the placeholder types T1 and T2 are replaced by the compiler with the data types int, char, float and double, when the compiler generates the specific instances of display().

Note:

1. All template arguments for a function template must be of template type arguments, otherwise, an error will occur.
2. When you create a function template, the compiler generate as many different versions of that function as necessary to handle the various ways that your program calls the function.

9.3.3 Overloaded Function Template

- We can use overloaded function template, in the situation where all possible data types cannot be handled using function templates. A template function may be overloaded either by template functions or ordinary functions of its name. Overloading in such a case is as follows:
  1. Call an ordinary function with same name.
  2. Call a template function with the same name.
- The program 9.8 shows how a template function is overloaded with an explicit function.

Program 9.8: Overloaded function template.

```
#include<iostream.h>
#include<string.h>
template<class T>
void display(T a)
{
   cout<<"Template display ="<<a<<endl;
}
void display(int a)          //overloads the template
{cout<<"Explicit display = "<<a<<endl; }
int main()
{
   display(50);              //overload function display()
   display('A');             //uses template display()
   display(40.5);            //uses template display()
   return 0;
}
```

Output:

```
Explicit display = 50
Template display = A
Template display = 40.5
```

Program 9.9: To find biggest of two strings i.e. big ("A", "B") then compiler give error message. This problem we can solve using overloaded function template.

```
//overloaded function template
#include <iostream>
#include<cstring>
using namespace std;
char *big (char *x, char *y)
{
    if (strcmp (x, y)>0)
    return x;
    else return y;
}
template <class T>
T big (T x, T y)
{
    if(x > y)
    return x;
    else return y;
}
int main()
{
    cout<<"Bigger no. is:"<<big (10, 20)<<endl;
    cout<<"Bigger is:"<<big ("A", "B");
}
```

Output:

```
Bigger no. is: 20
Bigger is: B
```

9.4 INTRODUCTION OF EXCEPTION HANDLING [S-18, 22, 23, W-18]

- When we work with program or compile the program, we normally get logic errors or syntactic errors. We can detect these errors by debugging. We often come across the errors other than logic and syntactic errors. They are known as exceptions. Exceptions are run-time errors that a program encounters while executing. For example, Array out of bounds, divide by zero.
- C++ provides a built-in error handling mechanism that is called **Exception Handling.** Using exception handling you can more easily manage and respond to run-time errors.

- The purpose of exception handling mechanism is to detect the exception in the following way:
  1. Find the run-time problem i.e. exception.
  2. Inform that an error has occurred by throwing the exception.
  3. Get the error information from compiler i.e. catch the exception.
  4. Handle the exception in catch block.

9.4.1 Exception Handling [S-18, W-22]

- C++ exception handling mechanism uses three keywords: **try**, **catch** and **throw**.
  - **try** block contains program statements that we want to monitor for exceptions.
  - When an exception is detected it is thrown using a **throw** statement in the try block.
  - The exception is caught using **catch**. Typically, a catch(...) block is used to log errors and perform special cleanup before program execution is stopped.
  - Any exception must be caught by a catch statement that immediately follows the try statement that throws the exception.
- **The general form of try and catch is given below:**

```
try
{
    ..........          //try block
    throw exception     //detects and throws exception
    ..........
}
catch(type1 arg)
{
    //catch block
}
catch(type2 arg)
{
    //catch block
}
    .
    .
    .
catch(typeN arg)
{
    //catch block
}
```

9.4.2 Simple try-catch Mechanism

- Let us first discuss try with only one catch block. This simple try-catch mechanism is illustrate in Program 9.10.

Program 9.10: Program for try-catch mechanism.

```
#include<iostream.h>
//using namespace std;
int main()
{
    cout<<"start\n";
    try
    {
        cout<<"try block \n";
        throw 10;        //throw an error
        cout<<"This will not work";
    }
    catch(int i)
    {
        cout<<"exception caught, Number is:";
        cout<<i<<"\n";
    }
        cout <<"end";
        return 0
}
```

Output:

```
start
try block
exception caught, Number is:10
end
```

- In this above Program 9.10:
  - In try block, only two statements will execute, the first cout statement and second throw. Once, the exception is thrown, control passes to the catch block and try block is terminated.
  - When the control passes to catch block, the statements in the catch block are executed.
  - Here, exception is integer exception.

Program 9.11: Simple program for try-catch statement.

```
//simple try-catch
#include<iostream>
using namespace std;
```

```
int main()
{
    int a, b;
    cin>>a>>b;
    int s = a - b;
    try
    {
        if(s! = 0)
        {
            cout<<"Result is="<<a/s<<'\n';
        }
    else
        {
            throw(s);    //throws integer exception
        }
    }
    catch(int i)
    {
        cout<<"exception caught:s="<<s<<endl;
    }
    return 0;
}
```

Output:

```
15    10
Result = 3
10    10
Exception caught: s = 0
```

- This program detects a division by zero error in try block. The program will not abort due to catch block.
- The exception is thrown using the object s.
- Catch statement contains integer type argument, catches the exception.
- If we change the argument in catch statement as **double**, then the exception will not be caught and abnormal termination occur.

9.4.3 Throwing the Exception from Function

- An exception can be thrown from a statement that is outside the try block but within a function that is called from within the try block.

Program 9.12: Program for throwing an exception from a function outside the try block

```
#include<iostream.h>
//using namespace std;
```

```
void funct(int t)
{
   cout<<"Inside function:"<<t<<'\n';
   if(t) throw t;
}
int main()
{
   cout<<"start\n";
   try
   {
   cout<<"Inside try \n";
      funct(0);
      funct(1);
      funct(2);
   }
}
catch(int i)
{
   cout<<"exception caught:"<<i<<"\n";
}
   cout<<"end";
   return 0;
}
```

Output:

```
start
inside try
inside function:0
inside function:1
exception caught:1
end
```

9.4.4 Try-Catch inside a Function

- When try block is inside a function, then each time the function is entered, the exception handling relative to that function is reset.

Program 9.13: Program for try catch inside a function.

```
#include<iostream>
//using namespace std;
void func(int x)
{
```

```
    try
    {
       if(x) throw x;
    }
    catch(int i)
    {
    cout<<"exception caught:"<< i <<"\n";
    }
}
int main()
{
    cout<<"start"<<"\n";
    func(1);                    //for every function call
    func(2);                    //exception handling is reset
    func(0);
    func(3);
    return 0;
}
```

Output:

```
start
exception caught: 1
exception caught: 2
exception caught: 3
```

9.4.5 Multiple Catch Statements

- We can have more than one catch associated with a try. When the exception is thrown, the exception handlers are searched in order for an appropriate match. When the match is found that catch block is executed. When no match is found, the program is terminated.

Program 9.14: Program for multiple catch statements.

```
#include<iostream.h>
//using namespace std;
void func(int x)
{
   try
   {
      if(x==1) throw x;            //int
      else
      if(x==0) throw 'x';          //char
      else
      if(x==-1) throw 1.5;         //double
      cout<<"try ends:\n";
   }
```

```
    catch(char ch)
    { cout<<"caught character:\n"<<ch<<'\n'; }
    catch(int i)
    { cout<<"caught integer:\n"<<i<<"\n"; }
    catch(double d)
    { cout<<"caught double:\n"<<d<<"\n"; }
    cout<<"end of catch block";
}
int main()
{ cout<<"start \n";
    func(1);
    func(0);
    func(-1);
    func(2);
    return0;
}
```

Output:

```
start
caught integer:1
end of catch blockcaught character:x
end of catch blockcaught double:1.5
end of catch blocktry ends: end of catch block
end of catch block
```

- In this program, when x = 1, it throws integer exception. This matches with the second catch block, therefore the second catch will executed. When x = 0, it throws character exception, therefore first catch block is executed. When x = –1, it throws double exception hence 3rd catch block is executed. Finally, when x = 2, program terminates (no. exception thrown).

9.4.6 Catch all Exceptions

- Sometimes, we want an exception handler to catch all exceptions instead of just a certain type. We use the following **syntax:**

```
catch (….)
{
    //process all exceptions
}
```

Here, catch(...) matches any type of data.

Program 9.15: Program for catch all exceptions.

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

```
void func(int x)
{
    try
    {
        if(x==1)throw x;        //int
        if(x==0)throw 'x';      //char
        if(x==-1) throw 1.5;    //double
    }
    catch(…)            //catch all exceptions
    {
        cout<<"caught exception \n";
    }
}
int main()
{
    cout<<"start \n";
    func(0);
    func(1);
    func(-1);
    return 0;
}
```

Output:

```
start
caught exception
caught exception
caught exception
```

- Using catch(...) is a good way to catch all exceptions that you do not want to handle explicitly. In Program 9.15 if we add one more catch block before catch(...) block as:

```
catch (int i)
{
    cout<<"caught:"<< i<<'\n';
}
```

- Then integer exception is also handle. Other two exceptions are handle by using catch(...) and the output is:

```
start
caught exception
caught 1
caught exception
```

9.4.7 Rethrowing an Exception

- Rethrowing an exception means to allow multiple handlers access to the exception. An exception can only be rethrown from within a catch block. When we rethrow an exception, it will not be recaught by the same catch block or any other catch in that group. Rather, it will be caught by an appropriate catch in the outer try-catch block.

Program 9.16: Program for rethrowing an exception.

```
#include<iostream.h>
//using namespace std;
void div (double x, double y)
{
   try
   {
      if(y==0.0) throw y;          //double
      else
      cout<<"Division="<<x/y<<"\n";
   }
   catch(double)
   {
      cout<<"caught double \n";
      throw;                  //Rethrowing exception double
   }
   cout<<"function end \n";
}
//end of function
int main()
{
   cout<<"start\n";
   try
   {
      div(8.6, 2.0);
      div(10.5, 0.0);
   }
   catch(double)
   {
      cout<<"caught double in main\n";
   }
   cout<<"end";
   return 0;
}
```

Output:

```
start
division = 4.3
function end
caught double
caught double in main
end
```

Program 9.17: Write a template program to sort integer and float array elements of size five

```
#include<iostream>
using namespace std;
int n;
#define size 10
template<class T>
void sel(T A[size])
{
    int i,j,min;
    T temp;
    for(i=0;i<n-1;i++)
    {
        min=i;
        for(j=i+1;j<n;j++)
        {
            if(A[j]<A[min])
            min=j;
        }
        temp=A[i];
        A[i]=A[min];
        A[min]=temp;
    }
    cout<<"\nSorted array:";
    for(i=0;i<n;i++)
    {
        cout<<" "<<A[i];
    }
}
```

```
int main()
{
    int A[size];
    float B[size];
    int i;

    cout<<"\nEnter total no of int elements:";
    cin>>n;
    cout<<"\nEnter int elements:";
    for(i=0;i<n;i++)
    {
        cin>>A[i];
    }
    sel(A);

    cout<<"\nEnter total no of float elements:";
    cin>>n;
    cout<<"\nEnter float elements:";
    for(i=0;i<n;i++)
    {
        cin>>B[i];
    }
    sel(B);
}
```

Output:

```
Enter total no of int elements:5
Enter int elements:5 6 3 8 1
Sorted array: 1 3 5 6 8
Enter total no of float elements:5
Enter float elements: 1.1 5.8 9.03 10.65 5.66
Sorted array: 1.1 5.66 5.8 9.03 10.65
```

Summary

- Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type.
- A template is a blueprint or formula for creating a generic class or a function.
- Template in C++ allows us to define generic classes and functions and thus provides support for generic programming.
- Generic programming is an approach where we can define a single function or a class which can work with any data types.

- A template specifies how to construct an individual class or function by providing a blueprint description of classes or functions within the template.
- A template in C++ allows the construction of a family of template functions and classes to perform the same operation on different data types. The templates declared for functions are called function templates and those declared for classes are called class templates. They perform appropriate operations depending on the data type of the parameters passed to them.
- A generic function defines a general set of operations that will be applied to various types of data. The type of data that the function will operate upon is passed to it as a parameter. Through a generic function, a single general procedure can be applied to a wide range of data.
- We can use multiple parameters in both the class templates and function templates.
- Like other functions, template functions can be overloaded.

Check Your Understanding

1. Class template can have ..... parameters.
   (a) any type (b) class type
   (c) both (d) none
2. The actual source code for implementing a template function is created when .....
   (a) the function is actually executed
   (b) the declaration of the function appears
   (c) the definition of the function appears
   (d) the function is invoked.
3. The template argument is preceded by the keyword .....
   (a) class (b) T
   (c) datatype (d) all
4. What is the correct syntax of defining function template/template functions?
   (a) Template <class T> void(T a){cout<<a;}
   (b) Template <class T> void(T a){cout<<a;}
   (c) Template <T> void(T a){cout<<a;}
   (d) Template <T> void(T a){cout<<a;}
5. In how many ways templates concept can be used?
   (a) 1 (b) 2
   (c) 3 (d) 4
6. Function templates require more memory space than normal function.
   (a) True (b) False
7. Templates are processed by the compiler.
   (a) True (b) False

8. Template classes can work with different data types.
   (a) True (b) False

Answers

1. (c)	2. (d)	3. (a)	4. (a)	5. (b)	6. (b)	7. (a)	8. (a)

Practice Questions

Q.I Answer the following Questions in short:

1. A template can be considered as a kind of macro. Then, what is the difference between them?
2. Distinguish between Overloaded Functions and Function Templates.
3. Distinguish between the terms Class Template and Template Class.
4. State which of the following definitions are illegal:

(a)
```
template <class T>
    class city
    { ------- };
```

(b)
```
template<class P, R, class S>
    class city
    { ------- };
```

(c)
```
template <class T, typename S>
    class city
    { ------- };
```

(d)
```
class <class T, int size = 10>
    class list
    { ------- };
```

5. What is meant by Template? How to declare it?
6. Which keyword used for exception handling?

Q.II Answer the following questions:

1. What is a Function Template? Write a function template for finding the largest number in a given array. The array parameter must be of generic data types.
2. Explain how the Compiler Processes call to a Function Template.
3. What is Class Template? Explain the syntax of a class template with suitable example.
4. Write a template based program for adding objects of the vector class.
5. Explain simple try-catch mechanism.
6. Explain Exception Handling.
7. Write a function template to calculate area of circle.
8. Write short note on: template.
9. With the help of syntax and example describe Class Template.
10. Write short note on: Overloading a function template.

11. Distinguish Class and Function Templates.
12. Write class and function template program for finding minimum value contained in an array.

Q.III Define the term:

1. Template
2. Exception Handling
3. Class Template
4. Function Template

Previous Exam Questions

April 2018

1. Explain in which circumstances catch(....) statement would be used? **[2 M]**

Ans. Refer to Section 9.4.1.

2. What is exception? Explain how exception is handled in C++. **[4 M]**

Ans. Refer to Section 9.4.

3. Write a note on class templates. **[4 M]**

Ans. Refer to Section 9.2.

October 2018

1. Define: **[2 M]**

(i) class template

Ans. Refer to Section 9.2.

(ii) function template.

Ans. Refer to Section 9.3.

2. Write a C++ program to find maximum and minimum of two integers and two float numbers by using function template. **[4 M]**

Ans. Refer to Program 9.4.

3. Write a note on exception handling. **[4 M]**

Ans. Refer to Section 9.4.

April 2019

1. Write a template program to sort integer and float array elements of size five. **[4 M]**

Ans. Refer to Program 9.17.

Solved Question Papers

Summer 2022

Time : $2^1/_2$ **Max. Marks : 70**

Q. 1 Attempt any EIGHT of the following. (Out of Ten) **[8 × 2 = 16]**

(a) What is extraction and insertion operator?

Ans. Refer to Section 6.2.6.

(b) Explain any two manipulators.

Ans. Refer to Section 2.6.

(c) Define constructor.

Ans. Refer to Section 4.2.

(d) What is inline function.

Ans. Refer to Section 2.10

(e) What is reference variable? What is its major use.

Ans. Refer to Section 2.4.3.

(f) What is Abstraction and Encapsulation.

Ans. Refer to Sections 1.2.3 and 1.2.4.

(g) What is compile - Time polymorphism.

Ans. Refer to Section 6.2.

(h) What is default argument.

Ans. Refer to Section 2.11.

(i) What is the use of scope resolution operator.

Ans. Refer to Section 2.5.1

(j) What are the access specifiers used in C++.

Ans. Refer to Section 3.4.

Q. 2 Attempt any four of the following. (Out of Five) **[4 × 4 = 16]**

(a) Explain memory management operators with the help of suitable example.

Ans. Refer to Section 2.5.2.

(b) Explain memory allocation for objects with non-static data member and static data member.

Ans. Refer to Sections 3.7 and 3.8.

(c) When do we make a class virtual base class? Explain it with suitable example.

Ans. Refer to Section 5.4.

(d) Explain array of object in C++ with example.

Ans. Refer to Section 3.9.

(e) Explain any four formatted input/output functions.

Ans. Refer to Section 7.5.

Q. 3 Attempt any Four of the following. (Out of Five) **[4 × 4 = 16]**

(a) Write a C++ program to create a class which contains two data members. Write member functions to accept, display and swap two entered numbers using call by reference.

Ans. Refer to Section 3.5.

(b) Write a C++ program to create a class Book which contains data members as B–Id, B–Name, B–Author, B– publication. Write member functions to accept and display Book information also display count of books.

(Use static data member to maintain count of books)

Ans. Refer to Section 3.8.

(c) Write a C++ program to calculate square and cube of integer number by using inline function.

Ans. Refer to Program 2.16 of Chapter 2.

(d) Design C++ class which contains function display(). Write a program to count number of times display() is called. (use static data member).

Ans. Refer to Program 3.3 of Chapter 3.

(e) Write a C++ program to read contents of a text file and count number of characters. Words and lines in a file.

Ans. Refer to Program 8.11 of Chapter 8.

Q. 4 Attempt any four of the following : (out of Five) **[4 × 4 = 16]**

(a) Can we pass class object as function arguments? Explain with the help of an example.

Ans. Refer to Section 3.10.

(b) Explain various stream classes used to perform console input/output (I/O) operations.

Ans. Refer to Section 7.3.

(c) What is class Template? Explain syntax of class template with suitable example.

Ans. Refer to Program 9.2.

(d) Write a program to perform addition of two matrices using operator overloading.

Ans. Refer to Section 6.2.2.

(e) Trace the output of the following program and explain it. Assume there is no syntax error.

```
# include < iostream.h>
Class abc
{
   int i;
   public :
   abc (int v = 0)
   [5803]-402
   {
      Cout <<"In the constructor\n'';
      i = v;
```

```
    }
    Void print (Void)
    {
        Cout <<"The value of i is " <<<endl;
    }
};
Void main ( )
{
    abc a (10);
    abc b ;
    a. print ( );
    b. print ( );
}
```

Output:

compile.time error.

- For variable i member function is not defined.
- return type of main function should be 'int'.

Q. 5 Write a short note on any two of the following : (Out of three) **[2 × 3 = 6]**

(a) Exception Handling

Ans. Refer to Section 9.4.

(b) Operator overloading

Ans. Refer to Section 6.2.2.

(c) Pointer to object with example

Ans. Refer to Section 6.3.2.

■■■

Winter 2022

Time : $2^1/_2$ **Max. Marks : 70**

Q.1 Attempt any Eight of the following (out of TEN). **[2 × 8 = 16]**

(a) What is Encapsulation?

Ans. Refer to Section 1.2.4.

(b) Define the terms: (i) Early Binding, (ii) Late Binding.

Ans. **(i) Early Binding:** Early binding is also known as static binding. Early binding occurs when we make the explicit or direct function call in our program.

(ii) Late Binding: Refer to Section 1.2.7.

(c) What is Inline function.

Ans. Refer to Section 2.10.

(d) Explain get() and put() function.

Ans. Refer to Section 7.4.2.

(e) What is stream?

Ans. Refer to Section 7.2.

(f) Define Friend function.

Ans. Refer to Section 3.11.

(g) Explain the use of new operator, state the syntax.

Ans. Refer to Section 2.5.2.

(h) State the need of virtual keyword.

Ans. Refer to Section 6.3.4.

(i) State user defined data types in C++.

Ans. Refer to Section 2.3.2.

(j) Explain the use of Scope Resolution operator.

Ans. Refer to Section 2.5.1.

Q.2 Attempt any FOUR of the following (out of FIVE). **[4 × 4 = 16]**

(a) List different types of constructor. Explain any constructor with example.

Ans. Refer to Section 4.3.

(b) What is function overloading? Explain with suitable example.

Ans. Refer to Section 6.2.1.

(c) Describe different types of inheritance.

Ans. Refer to Section 5.3.

(d) Explain virtual base class with suitable diagram.

Ans. Refer to Section 5.4.

(e) Describe file manipulators with their syntaxes.

Ans. Refer to Section 8.4.1.

Q.3 Attempt any FOUR of the following (out of FIVE). **[4 × 4 = 16]**

(a) Write a C++ program to copy contents of one file to another file.

Ans. Refer to Program 8.12, Chapter 8.

(b) Write a program to calculate area and circumference of a circle using inline function.

Ans. Refer to Program 2.19, Chapter 2.

(c) Declare a class of vehicle. Derived classes are two wheeler, three wheeler and four wheeler. Display the properties of each type of vehicle using member functions of class.

Ans. Refer to Section 3.5.2.

(d) Write a C++ program to use setfill() and setiosflags() manipulator.

Ans. Refer to Section 2.6.

(e) Write a C++ program to compare two strings using overload operator "==".

Ans. Refer to Program 6.10, Chapter 6.

Q.4 Attempt any FOUR of the following (out of FIVE). **[4 × 4 = 16]**

(a) Trace the output of the following program and explain it. Assume there is no syntax error.

```
#include<iostream.h>
int i, j;
Class sample
{
Public:
   Sample(int a = 0, int b = 0)
   {
      i = a;
      j = b;
```

```
    show();
}
Void show()
{
Cout<<j<<" ";
}
};
Void main()
{
    Sample(5, 10);
    Int & x = i;
    int & y = j;
    i++;
Cout<<x--<<" "<<++y;
}
```

Ans. Error, class is not a type

(b) Explain try, catch and throw in exception handling.

Ans. Refer to Section 9.4.1.

(c) Design C++ class which contain function display(). Write a program to count number of times display() function is called (Use static data member).

Ans. Refer to Program 3.3, Chapter 3.

(d) What is Destructor? State the importance of destructor with example.

Ans. Refer to Section 4.8.

(e) What is tokens in C++? Explain in detail.

Ans. Refer to Section 2.1.1.

Q.5 Write short note on any TWO of the following (out of THREE): **[3 × 2 = 6]**

(a) Call-by-value and call-by-reference.

Ans. Refer to Sections 2.9.1 and 2.9.2.

(b) Data abstraction.

Ans. Refer to Section 1.2.3.

(c) Default Argument.

Ans. Refer to Section 2.11.

■■■

Summer 2023

Time : $2^{1}/_{2}$ **Max. Marks : 70**

Q. 1 Attempt any Eight of the following : (Out of 10) **[8×2=16]**

(a) Explain tellg() and tellp() with syntax.

Ans. Refer to Section 8.4.1.

(b) Explain any two manipulators.

Ans. Refer to Section 2.6.

(c) What is destructor?

Ans. Refer to Section 4.8.

(d) What are the visibility lables used in C++.

Ans. Refer to Section 3.4.

(e) What is extraction and insertion operator?

Ans. Refer to Section 6.2.6.

(f) What is abstraction and Encapsulation?

Ans. Refer to Sections 1.2.3 and 1.2.4.

(g) What is default argument in function?

Ans. Refer to Section 2.11.

(h) Write any two uses of scope resolution operator.

Ans. Refer to Section 2.5.1.

(i) What is static Polymorphism.

Ans. Refer to Section 6.1.

(j) Explain structure of C++ program.

Ans. Refer to Section 1.7.

Q. 2 Attempt any four of the following: (Out of 5) **[4×4=16]**

(a) Explain operator overloading in C++ with an example.

Ans. Refer to Section 6.2.2.

(b) Explain memory allocation for objects with non-static data member and static data member.

Ans. Refer to Sections 3.7 and 3.8.

(c) What is pure virtual function and explain with the help of example program.

Ans. Refer to Section 6.3.5.

(d) Explain Dynamic constructor with suitable example.

Ans. Refer to Section 4.7.

(e) What is inheritance and explain the hierarchical inheritance.

Ans. Refer to Sections 5.1 and 5.3.4.

Q. 3 Attempt any four of the following : (Out of 5) **[4×4=16]**

(a) Write a C++ program to create a class which contains two data members. Write member functions to accept, display and swap two entered numbers using call by reference.

Ans. Refer to Section 3.5.

(b) Write a C++ program to create a class customer which contains data members as C_id, C_name, C_Salary. Write member functions to accept and display customer information, also display information of customer having maximum salary.

Ans. Refer to Program 3.12 of Chapter 3.

(c) Write a C++ program to calculate factorial of integer number by using inline function.

Ans. Refer to Program 2.16 of Chapter 2.

(d) Design C++ class which contains function count(). Write a program to count number of time count() is called. (Use static data member.)

Ans. Refer to Program 3.3 of Chapter 3.

(e) Write a C++ program to copy the contents of a text file into another text file.

Ans. Refer to Program 8.12 of Chapter 8.

Q. 4 Attempt any four of the following. (Out of 5) **[4×4=16]**

(a) Explain object as function arguments? Explain with the help of an example program.

Ans. Refer to Section 3.10.

(b) Explain different characteristics of friend function.

Ans. Refer to Section 3.11.

(c) What is class Template? Explain syntax of class template with suitable example.

Ans. Refer to Section 9.2.1.

(d) Write a program to overload binary + operator to add two strings.

Ans. Refer to Program 6.5 of Chapter 6.

(e) Trace the output of the following program and explain it. Assume there is no syntax error.

```
# include < iostream.h >
class Point {
private :
    int x ;
    int y ;
public :
    Point (int i, int j); // constructor
};
    Point : : Point (int i = 0; int j = 0) {
x = i;
y = j;
cout << "constructor called" ;
}
int main()
{
    point t1, * t2;
    return 0;
}
```

Output:

Constructor called

If statement 'Point t1, *t2'; then we can see that only one object is constructed here. t2 is just a pointer variable not an object.

Q. 5 Write a short note on any two of the following : (Out of three) **[2×3=6]**

(a) This pointer

Ans. Refer to Section 6.3.1.

(b) function overriding

Ans.

- Function overriding is a concept in Object-Oriented Programming (OOP) where a subclass provides a specific implementation for a method that is

already defined in its superclass. The overridden method in the subclass has the same signature (name, return type, and parameters) as the method in the superclass. The purpose of function overriding is to enable a derived class to provide a specialized implementation of a method inherited from its base class.

- **Key points about function overriding:**
  1. **Inheritance is a prerequisite:** Function overriding is closely tied to inheritance, as it involves creating a hierarchy of classes where a subclass inherits the properties and behaviors of a superclass.
  2. **Same method signature:** The overriding method in the subclass must have the same method signature (name, return type, and parameters) as the method in the superclass. This ensures that the compiler correctly identifies it as an overridden method.
  3. **Override annotation (optional):** Depending on the programming language, there may be an optional "override" keyword or annotation to explicitly indicate that a method is intended to override a method in the superclass. This can help catch errors if the method signature in the subclass does not match any method in the superclass.
  4. **Dynamic polymorphism:** Function overriding is a fundamental mechanism for achieving dynamic polymorphism, where the appropriate method to call is determined at runtime based on the actual type of the object.
  5. **Enhancing or customizing behavior:** Subclasses override methods to provide their own implementation while retaining the general interface defined by the superclass. This allows for customization of behavior to suit the specific needs of the subclass.

(c) Exception handling.

Ans. Refer to Section 9.4.

■■■