USN | | | | | | | | | | |

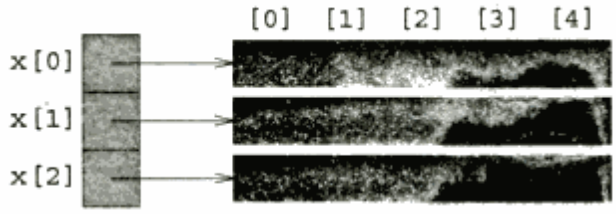
Internal Assessment Test 1 Souktion and scheme – May. 2022

Sub:	Data Structure using C++					Sub Code:	18EC643	Branch:	ECE
Date:	-05-2022	Duration:	90 Minutes	Max Marks:	50	Sem / Sec:	6 (A, B,C,D)		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	(a) What function ? Explain different types of function with sysntax. For better understanding of arguments and return in functions, user-defined functions can be categorised as: • Function with no argument and no return value • Function with no argument but return value • Function with argument but no return value • Function with argument and return value The parameters passed to function are called ***actual parameters***. The parameters receive here are two most popular ways to pass parameters. ***Pass by Value:*** In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of caller. parameters of function are called ***formal parameters***. **Passing** an argument by **address** involves **passing** the **address** of the argument variable rather than the argument variable itself. Because the argument is an **address**, the function parameter must be a pointer. Pointer Int *a; Int b a=&b (b) Write a program to check entered number is postive or negative using call by reference function. `#include <iostream>` `using namespace std;` `int main()` `{` `signed long num1 = 0;` `cout << "\n\n Check whether a number is positive, negative or zero :\n";` `cout << "----------\n";` `cout << " Input a number : ";` `cin >> num1;` `even(num);`	(5+5)M	CO1	L1 L3

	void num(int &num1) if(num1 > 0) { cout << " The entered number is positive.\n\n"; } else if(num1 < 0) { cout << " The entered number is negative.\n\n"; } else { std::cout << " The number is zero.\n\n"; } return 0; }			
2	(a)What is template? Explain its type with syntax. • A template is a simple and yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types. • C++ adds two new keywords to support templates: *'template'* and *'typename'*. The second keyword can always be replaced by keyword 'class'. • The **c++ template** can be defined as a blueprint or formula for creating a generic class or a function. Templates are the foundation of generic programming. **template**<**class** T> T add(T &a,T &b) { T result = a+b; **return** result; } • **Function Templates with Multiple Parameters We can use more than one generic type in the template function by using the comma to separate the list.** • **Syntax:** **template<class T1, class T2,…..>** **return_type function_name (arguments of type T1, T2….)** **{** **//body of function.** **}** #include<iostream> using namespace std; template<class X,class Y> void fun(X a,Y b) { cout << "Value of a is : " <<a<< endl; cout << "Value of b is : " <<b<< endl; } int main()	(5+5)M	CO1	L1 L3

	{ fun(15,12.3); return 0; } b) Write a program in C++ to swap two variable content using on templates function.			
3	(a)What is pointer? Explain with an example. **1.4.1 The Operator `new`** Run-time or dynamic allocation of memory may be done using the C++ operator **new**. This operator returns a pointer to the allocated memory. For example, to dynamically allocate memory for an integer, we must declare a variable (e.g., **y**) to be a pointer to an integer using this statement: `int *y;` When the program needs to actually use the integer, memory may be allocated to it using this syntax: `y = new int;` The operator **new** allocates enough memory to hold an integer, and a pointer to this memory is returned and saved in **y**. The variable **y** references the pointer to the integer, and ***y** references the integer. To store an integer value, for example 10, in the newly allocated memory, we can use the following syntax: `*y = 10;` We can combine the three steps–declare **y**, allocate memory, and assign a value to ***y**–into a smaller number of steps as shown in the following examples: `int *y = new int;` `*y = 10;` or `int *y = new int (10);` or `int *y;` `y = new int (10);` ▶ int *ptr = new int; ▶ By writing new int, we allocated the space in memory required by an integer. Then we assigned the address of that memory to an integer pointer ptr. ▶ We assign value to that memory as follows: ▶ *ptr = 4;	[4+6]M	CO1	L3 L3

	b)what is Dynamic Memory Allocation(DMA)? Explain 1-Dimension and 2-Dimension array declaration with neat figures and syntax. ▶ The main use of the concept of dynamic memory allocation is for allocating arrays when we have to declare an array by specifying its size but are not sure about the size. ▶ char name[30]; ▶ And if the user enters the name having only 12 characters, then the rest of the memory space which was allocated to the array at the time of its declaration would become waste, thus unnecessary consuming the memory. ▶ In this case, we will be using the new operator to dynamically allocate the memory at runtime. We use the new operator as follows. ▶ char *arr = new char[length] ▶  ▶ `template <class T>` `bool make2dArray(T ** &x, int numberOfRows, int numberOfCol` `{// Create a two dimensional array.` `try {` `// create pointers for the rows` `x = new T * [numberOfRows];` `// get memory for each row` `for (int i = 0; i < numberOfRows; i++)` `x[i] = new int [numberOfColumns];` `return true;` `}` `catch (bad_alloc) {return false;}` `}` ▶			
4	(a)Write a C++ program which prints principal diagonal elements of square matrix of order mXn in reverse order using DMA concept. `class GFG {` `static int MAX = 100;` `// Function to print the Principal Diagonal` `static void printPrincipalDiagonal(int mat[][], int n)` `{` `System.out.print("Principal Diagonal: ");` `for (int i = 0; i < n; i++) {` `for (int j = 0; j < n; j++) {` `// Condition for principal diagonal`	[6+4]M	CO1	L3 L3

```
                if (i == j) {
                    System.out.print(mat[i][j] + ", ");
                }
            }
        }
        System.out.println("");
    }

    // Function to print the Secondary Diagonal
    static void printSecondaryDiagonal(int mat[][], int n)
    {
        System.out.print("Secondary Diagonal: ");

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {

                // Condition for secondary diagonal
                if ((i + j) == (n - 1)) {
                    System.out.print(mat[i][j] + ", ");
                }
            }
        }
        System.out.println("");
    }

    // Driver code
    public static void main(String args[])
    {
        int n = 4;
        int a[][] = { { 1, 2, 3, 4 },
                      { 5, 6, 7, 8 },
                      { 1, 2, 3, 4 },
                      { 5, 6, 7, 8 } };

        printPrincipalDiagonal(a, n);
        printSecondaryDiagonal(a, n);
    }
}
```

(b)Create student class, display his info using class template.

5	(a)Explain function overloading with an appropriate example. Function overloading is a feature of object oriented programming where two or more functions can have the same name but different parameters. When a function name is overloaded with different jobs it is called Function Overloading. In Function Overloading "Function" name should be the same and the arguments should be different. Function overloading can be considered as an example of polymorphism feature in C++. Following is a simple C++ example to demonstrate function overloading. `void print(int i) {` `  cout << " Here is int " << i << endl;`	[5+5]M	CO1	L3 L2

```
}
void print(double  f) {
  cout << " Here is float " << f << endl;
}
void print(char const *c) {
  cout << " Here is char* " << c << endl;
}

int main() {
  print(10);
  print(10.10);
  print("ten");
  return 0;
}
```

(b)Write a C++ program to substract two matrices and print result in matrix form.

```
#include<iostream>
using namespace std;
int main()
{
    int matOne[3][3], matTwo[3][3], matSub[3][3], i, j;
    cout<<"Enter 9 Elements for First Matrix: ";
    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
            cin>>matOne[i][j];
    }
    cout<<"Enter 9 Elements for Second Matrix: ";
    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
            cin>>matTwo[i][j];
    }
    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
            matSub[i][j] = matOne[i][j] - matTwo[i][j];
    }
    cout<<"\nThe New Matrix (Subtraction Result) is:\n";
    for(i=0; i<3; i++)
    {
        for(j=0; j<3; j++)
            cout<<matSub[i][j]<<"  ";
        cout<<endl;
    }
    cout<<endl;
    return 0;
}
```

6	a) What is recursive function? Using function template write a C++ program to find a factorial of given number using recursive concept.	[6+4]M	CO1	L3 L1

In this tutorial, we will learn about recursive function in C++ and its working with the help of examples.

A function that calls itself is known as a recursive function. And, this technique is known as recursion

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

int factorial(int);

int main() {
    int n, result;

    cout << "Enter a non-negative number: ";
    cin >> n;

    result = factorial(n);
    cout << "Factorial of " << n << " = " << result;
    return 0;
}

int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}
```

b)Explain C++ constructors and Destructors..

Constructor is a member function of class, whose name is same as the class.

A constructor is a special type of member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) is created.

Constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructors.

Constructor does not have a return value, hence they do not have a return type.

Prototype of Constructors:-

<class-name> (list-of-parameters);

Constructors can be defined inside or outside the class declaration:-

1. Syntax for defining the constructor within the class:

<class-name> (list-of-parameters)

{

// constructor definition

}

2. Syntax for defining the constructor outside the class:

<class-name>: :<class-name> (list-of-parameters)

{

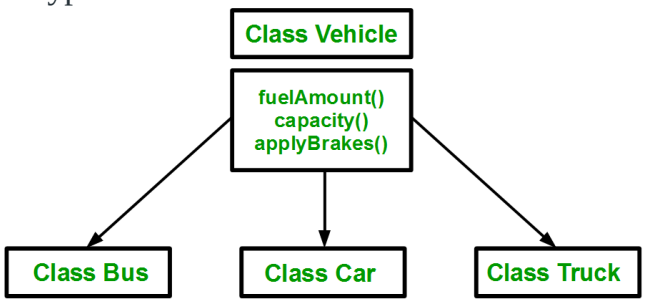
// constructor definition

}

C++

```
// defining the constructor within the class


#include <iostream>

using namespace std;


class student {

    int rno;

    char name[10];

    double fee;


public:

    student()

    {

        cout<<"Enter the RollNo:";
```

```
        cin>>rno;

        cout<<"Enter the Name:";

        cin>>name;

        cout<<"Enter the Fee:";

        cin>>fee;

    }



    void display()
```

7	(a)What is inheritance? Explain it types with neat diagrams and syntax. The capability of a class to derive properties and characteristics from another class is called **Inheritance**. Inheritance is one of the most important features of Object-Oriented Programming. • **Sub Class:** The class that inherits properties from another class is called Subclass or Derived Class. • **Super Class:** The class whose properties are inherited by a subclass is called Base Class or Superclass. **The article is divided into the following subtopics:** • Why and when to use inheritance? • Modes of Inheritance • Types of Inheritance  sing inheritance, we have to write the functions only one time instead of three times as we have inherited the rest of the three classes from the base class (Vehicle). **Implementing inheritance in C++**: For creating a sub-class that is inherited from the base class we have to follow the below syntax. **Syntax**: `class subclass_name : access_mode base_class_name` `{` `  // body of subclass`	[7+3]	CO1	L3 L1

	}; Here, **subclass_name** is the name of the subclass, **access_mode** is the mode in which you want to inherit the subclass for example public, private, etc. and **base_class_name** is the name of the base class from which you want to inherit the subclass. (b) Explain virtual and pure virtual functions. . Virtual Function in C++ A virtual function is a member function which is declared within a base class and is re-defined(Overriden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function. Pure Virtual Functions in C++ A pure virtual function (or abstract function) in C++ is a virtual function for which we don't have an implementation, we only declare it. A pure virtual function is declared by assigning 0 in the declaration. **Similarities between virtual function and pure virtual function** 1. These are the concepts of Run-time polymorphism. 2. Prototype i.e. Declaration of both the functions remains the same throughout the program. 3. These functions can't be global or static.			

HOD CI CCI