24CB204/24CM204/24CS204/24DS204/24IT204

I/IV B.Tech. (Regular) DEGREE EXAMINATION

May, 2025 **Common to CB, CM, CSE, DS & IT Branches**

Second Semester **Programming for Problem Solving**

Time: Three Hours **Maximum:** 60 Marks

Scheme of Evaluation & Answers

1. a) Mention at least two uses of pointers. (1M)

Ans: **Any two of the following uses of pointers:**

➢ Pointers are more efficient in handling arrays and data tables.

➢ Pointers can be used to return multiple values from a function via function arguments.

➢ The use of pointer arrays to character string results in saving of data storage space in memory.

➢ Pointers allow C to support dynamic memory management.

➢ Pointers reduce the length and complexity of programs.

➢ They increase the execution speed and reduce the program execution time.

1. b) What is the purpose of strncat() function? (1M)

Ans: This is used for combining or concatenating n characters of one string into another.
It accepts three parameters. The first two are strings, while the last one is an integer specifying the maximum length of the second string to be appended at the end of the first.

1. c) Write syntax of structure definition. (1M)

Ans:
```
struct structure_name
{
   data_type member1;
   data_type member2;
   .
   .
   data_type memeberN;
};
```

1. d) Name one function used for error handling in file operations. (1M)

Ans: feof() **or** ferror().

1. e) What is the purpose of calloc()? (1M)

Ans: This function is used to allocate multiple blocks of memory and initialize them all to zero and the pointer will point to the first byte of the first block. Here all blocks are same size.

1. f) Define an object in Object-Oriented Programming (OOP). (1M)

Ans: An object is a fundamental unit in OOP that represents a real-world entity or an abstract concept. It combines data (called attributes or properties) and behaviour (called methods or functions) into a single unit.

1. g) What do you mean by data hiding in OOP? (1M)

Ans: Data hiding is an important concept in OOP that refers to restricting direct access to some of an object's data (attributes or internal details) from outside the object.

1. h) Is destructor overloading possible? If yes/no explain the reason. (1M)

Ans: *No*, destructor overloading is *not* possible. (**0.5M**)
Destructor cannot take parameters. Because function overloading requires functions to differ by parameter lists, it's impossible to overload a destructor since there's only one possible signature. (**0.5M**)

1. i) Which operator is used to denote a reference variable in C++? (1M)

Ans: & operator is used to denote a reference variable.

1. j) Write the syntax of operator function. (1M)

Ans: An operator function is a special function that overloads an operator.

return_type class_name::operator op (parameter_list)
{
    // function body
}
	return_type operator op (parameter_list)
	{
	    // function body
	}

1. k) What does object composition mean in OOP? (1M)

Ans: Composition means one class has an object of another class.
(It's a "has-a" relationship — like a Car has-a Engine)

1. l) What is an abstract class? (1M)

Ans:
- An abstract class is one that is not used to create objects. **(OR)**
- An abstract class is designed only to act as a base class (to be inherited by other classes). **(OR)**
- It is meant to provide a common interface for all derived classes.

UNIT – I

2 a) What is pointer? Explain how the pointer variable is declared and initialized. Give example. (6M)

Ans: Pointer (1M) - The pointer is a variable which stores the address of another variable.

Declaration of Pointers (2M): The pointer is declared just like an ordinary variable. They must be declared before they are used. The general syntax of a pointer declaration is as follows:

```
data_type *Pointer_Name;
```

Here the "*" tells that variable Pointer_Name is pointer type variable, i.e., it holds the address of another variable specified by the data_type. Pointer_Name needs a memory location. Pointer_Name points to a variable of type data_type.

Initialization of Pointer Variables (2M): The process of assigning the address of a variable to a pointer variable is known as initialization. The initialized pointers will produce good results otherwise they will produce erroneous results. Once a pointer variable has been declared we can use an assignment operator to initialize the pointer variable.

Example (1M):

```
int x;
int *p;
p = &x;
```

2 b) Write a C program to demonstrate a structure within a structure. (6M)

Ans: Model Program to demonstrate a structure within a structure (6M)

```
#include<stdio.h>
struct address
{
        char city[20];
        int pin;
        char phone[14];
};
struct employee
{
        char name[20];
        struct address add;
};
void main ()
{
        struct employee emp;
        printf("Enter employee information? ");
        scanf("%s %s %d %s",emp.name,emp.add.city, &emp.add.pin, emp.add.phone);
        printf("Printing the employee information....\n");
        printf("Name: %s\nCity: %s\nPincode: %d\nPhone: %s",emp.name,emp.add.city,emp.add.pin,
        emp.add.phone);
}
```

(OR)

3 a) Explain array of structures with suitable example. (6M)

Ans: Array of Structures (1M) - An array of structures can be defined as the collection of multiple structures variables where each variable contains information about different entities. The array of structures are used to store information about multiple entities of different data types. The array of structures is also known as the collection of structures.

`struct marks { int sub1; int sub2; int sub3; int total; }; struct marks student[3];`	**(2M)**

Any Example or Program (3M):

```
#include<stdio.h>
struct marks
{
        int sub1,sub2,sub3;
        int total;
};
```

```
int main()
{
        struct marks student[3],total={0,0,0,0};
        int i;
        for(i=0;i<3;i++)
        {
                printf("Enter student%d 3 subject marks: ",i+1);
                scanf("%d%d%d",&student[i].sub1,&student[i].sub2,&student[i].sub3);
        }
        for(i=0;i<3;i++)
        {
                student[i].total = student[i].sub1+student[i].sub2+student[i].sub3;
                total.sub1 += student[i].sub1;
                total.sub2 += student[i].sub2;
                total.sub3 += student[i].sub3;
                total.total += student[i].total;
        }
        printf("\nSTUDENTWISE TOTALS\n\n");
        for(i=0;i<3;i++)
                printf("Student%d Total: %d\n",i+1,student[i].total);
        printf("\nSUBJECTWISE TOTALS\n\n");
        printf("Subject1 Total: %d\n",total.sub1);
        printf("Subject2 Total: %d\n",total.sub2);
        printf("Subject3 Total: %d\n",total.sub3);
        printf("\nGRAND TOTAL: %d\n",total.total);
        return 0;
}
```

3b) Write a C program to reverse the given string using a pointer. (6M)

Ans: Model program to reverse the given string using a pointer (6M)

```
#include<stdio.h>
#include<string.h>
void reverse(char *);
int main()
{
   char str[20];
   printf("Enter a string: ");
   gets(str);
   printf("String Before Reverse: %s\n",str);
   reverse(str);
   printf("String After reverse: %s\n",str);
   return 0;
}
void reverse(char *str)
{
   char *bptr,*eptr,ch;
   int i,len;
```

```
    len=strlen(str);
    bptr=str;
    eptr=str+len-1;
    for(i=0;i<len/2;i++)
    {
       //Swap character
       ch=*bptr;
       *bptr=*eptr;
       *eptr=ch;
       //Update pointers
       bptr++;
       eptr--;
    }
}
```

UNIT-II

4 a) Explain various file I/O functions with suitable examples. (6M)

Ans: Character Input / Output Functions (1M)

getc() and fgetc() Functions:
They read next character from the file stream which can be a user defined stream or stdin and convert it to integer. If read detects end of file, it returns EOF. The syntax of these functions is:

```
int getc(FILE *fp);  (OR)  int fgetc(FILE *fp);
```

putc() and fputc() Functions:
They are used to write a character to the file stream specified which can be a user defined stream or stdout or stderr. The first parameter is a character, and second parameter is the file pointer. The syntax of these functions is:

```
int putc(int ch, FILE *fp);  (OR)  int fputc(int ch, FILE *fp);
```

String Input / Output Functions (1M)
fputs() and fgets() Functions:
fgets() and fputs(), read and write character strings from and to a disk file.

They have the following prototypes:

```
int fputs(const char *str, FILE *fp);
char *fgets(char *str, int length, FILE *fp);
```

The fputs() function writes the string pointed to by str to the specified stream. It returns EOF if an error occurs. The fgets() function reads a string from the specified stream until either a newline character is read or length–1 characters have been read.

Integer Input / Output Functions (1M)
The getw() and putw() functions
The getw() and putw() are integer-oriented functions. They are similar to getc and putc functions and are used to read and write integer values. These functions would be useful when we deal with only integer data. The general forms of getw and putw are:

```
putw(integer, fp);
n=getw(fp);
```

Examples or Programs (3M)

4 b) List and explain the functions used for dynamic memory management in C. Illustrate with examples. (6M)

Ans: Dynamic Memory Management Functions (2M)

The process of allocating memory at run time is known as **dynamic memory allocation.** The C programming does not have this feature. But it offers through memory management functions. The memory management functions are used to allocate and de-allocate storage during program execution. The functions are: 1. malloc 2. calloc 3. free 4. realloc

Explanation of these functions and Examples (4M)

malloc() (1M): This function is used to allocate one block of storage. It reserves the size specified by the pointer and returns void. The syntax is: **ptr = (cast-type *) malloc (byte-size);**

Example: **x = (int *) malloc (100 * sizeof(int));**

calloc() (1M): This function is used to allocate multiple blocks of memory. It is used to store run time data types such as arrays and structures. It allocates storage for multiple blocks and initializes them all to zero and the pointer will point to the first byte of the first block. Here all blocks are same size. The syntax is: **ptr = (cast-type *) calloc (n, element-size);**

Example: **x = (int *) calloc(n, sizeof(int));**

free() (1M): Whenever we are not in need of the variable for which the storage is allocated dynamically then such variable storage is removed. It is done by using **free** function.

The syntax is: **free (ptr);**

realloc() (1M): Reallocation is done when:

1. Storage allocated to a variable is not sufficient to modify the variable's storage.
2. If the storage is allocated more than the required storage, then also we need to alter the storage.

The process that is carried in the above situations is called **reallocation**.

The syntax is: **ptr = realloc (ptr, newsize);**

(OR)

5 a) What is the scope resolution operator in C++? Describe its uses with examples. (6M)

Ans: Scope resolution operator (1M) - Scope resolution operator is used to define or access identifiers (like variables, functions, or classes) outside their current scope — especially when there are naming conflicts or to access global definitions, class members, or namespace contents.

Main Uses of Scope Resolution Operator (At least two uses – 3M)

1. Access Global Variables or Functions: When a local variable has the same name as a global one, you can use :: to refer to the global version.
2. Define Class Member Functions Outside the Class: Instead of writing the function body inside the class, you can declare it in the class and define it outside using the scope resolution operator.
3. Access Namespaces: The scope resolution operator allows you to access members of a specific namespace.
4. Access Static Members of a Class: You can access static members of a class using the scope resolution operator.
5. Access Enums or Nested Classes: When you have enums or classes nested inside other classes or namespaces, :: helps access them.

Examples (2M)

5 b) Write a C program to copy contents of one file to another file. (6M)

Ans: Model program to copy contents of one file to another file (6M)

```
#include<stdio.h>
int main()
{
        FILE *fs,*ft;
        char ch;
        fs=fopen(“Source.txt”, “r”);
        ft=fopen(“Target.txt”, “w”);
        while((ch=getc(fs))!=EOF)
        {
                putc(ch,ft);
        }
        fclose(fs);
        fclose(ft);
        printf(“The file copy operation performed successfully.”);
        return 0;
}
```

UNIT-III

6 a) Explain the concept of constructors and how constructor overloading is achieved in C++. (6M)

Ans: Concept of Constructors (2M)

A constructor is a special member function whose task is to initialize the objects of its class. It is special because its name is the same as the class name. The constructor is invoked whenever an object of its associated class is created. It is called constructor because it constructs the values of data members of the class. A constructor is declared and defined as follows: (**Any example**)

```
class integer
{
        int m,n:
public:
        integer(void);//constructor declared
        …..
};
integer :: integer(void)
{
        m=0;  n=0;
}
```

Overloading Constructors (1M): Like the function, it is also possible to overload the constructors. A class can contain more than one constructor, if the argument list is different. This is known as "constructor overloading". All constructors are defined with the same name as the class name. Depending upon the number of arguments the compiler executes an appropriate constructor.

Any Example or Program (3M)

Model Program to demonstrate the usage of overloading constructors:

```
#include<iostream>
using namespace std;
class cse
{
        int b,g;
        public:
   cse()   //Default constructor
   {
      b=25;
      g=35;
   }
   cse(int x,int y)  //Parameterized constructor
   {
      b=x;
      g=y;
   }
   cse(int x)    //Parameterized constructor
   {
      b=g=x;
   }
   void display()
   {
      cout<<"The Boys are:"<<b<<endl;
      cout<<"The Girls are:"<<g<<endl;
   }
};
int main()
{
        cse ca,cb(15,45),cc(30); //Calling different constructors
        cout<<"-----First Object-----"<<endl;
        ca.display();
        cout<<"-----Second Oject-----"<<endl;
        cb.display();
        cout<<"-----Third Object-----"<<endl;
        cc.display();
        return 0;
}
```

6 b) Write a C++ program to demonstrate static member function with an example. (6M)

Ans: Any example program (6M).

Model program to illustrate static member function:

```
#include<iostream>
using namespace std;
class item
{
   int code;
   static int count;
public:
   void setcode(int);
   void showcode();
   static void showcount(void);
};
int item::count;
void item::setcode(int c)
{
   code=++count;
}
void item::showcode(void)
{
   cout<<"Code = "<<code<<endl;
}
void item::showcount(void)
{
   cout<<"Count = "<<count<<endl;
}
int main()
{
   item I1,I2,I3;
   item::showcount();
   I1.showcount();
   I2.showcount();
   I3.showcount();
   I1.setcode(10);
   I2.setcode(20);
   I3.setcode(30);
   I1.showcode();
   I2.showcode();
   I3.showcode();
   I1.showcount();
   I2.showcount();
   I3.showcount();
   item::showcount();
   return 0;
}
```

(OR)

7 a) What is class? Explain class specification with an example. (6M)

Ans: Class (1M): Class is a group of objects that share common properties and relationships. A class is a new data type that contains member variables and member functions that operate on the variables. A class is defined with the keyword class. It allows the data to be hidden, if necessary, from external use.

Class Specification (3M): Generally, a class specification has two parts:

a) Class declaration   b) Class function definitions

a) The class declaration describes the type and scope of its members. The class function definition describes how the class functions are implemented. The general form of the class declaration is:

```
class class_name
{
        private:
            variable declarations;
            function declaration ;
        public:
            variable declarations;
            function declaration;
    };
```

The members that have been declared as private can be accessed only from within the class. On the other hand, public members can be accessed from outside the class also. The data hiding is the key feature of OOPS. The use of keywords private is optional. By default, the members of a class are private.

b) The general form of a class (member) function definition is:

```
return-type class-name :: function-name (argument declaration)
{
    function-body
}
```

Example of simple class specification (2M): (Any example)

```
class item
{
    int number; //variables declaration
    float cost;         //private by default
  public:
    void getdata (int a,float b); //functions declarations
    void putdata (void);          //using prototype
};
```

7 b) What is a friend function? Explain in detail with example program. (6M)

Ans: Friend Function (2M) - A friend function is a function that is not a member of a class but is allowed to access the private and protected members of the class. It is declared inside the class with the keyword friend.

Any Program (4M) - Following is the model program to illustrate the use of a friend function:

```
#include<iostream>
using namespace std;
class sample
{
        int a;
        int b;
        public:
        void setdata(void);
        friend float mean(sample);
};
void sample::setdata(void)
{
        cout<<"Enter two values: ";
        cin>>a>>b;
}
```

```
float mean(sample S)
{
        return (S.a+S.b)/2.0;
}
int main()
{
        sample S; // Object S
        S.setdata();
        float avg;
        avg=mean(S);
        cout<<"Mean value of object S: "<<avg<<endl;
        return 0;
}
```

UNIT – IV

8 a) Write a C++ program to overload unary - operator with a friend function. (6M)

Ans: Model program to overload unary - operator using friend function (6M)

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

class space
{
   int x;
   int y;
   int z;
public:
   void getdata(int a,int b,int c)
   {
      x=a; y=b; z=c;
   }
   void display(void)
   {
      cout<<"x = "<<x<<endl;
      cout<<"y = "<<y<<endl;
      cout<<"z = "<<z<<endl;
   }
   friend void operator -(space &);
};

void operator -(space &p)
{
   p.x = -p.x;
   p.y = -p.y;
   p.z = -p.z;
}

int main()
{
   space S1,S2;
   S1.getdata(13,-2,35);
   S2.getdata(-52,81,1);
```

```
    cout<<"\nBefore Change\n";
    cout<<"Co-ordinates of S1 Point\n";
    S1.display();
    cout<<"Co-ordinates of S2 Point\n";
    S2.display();
    -S1;
    -S2;
    cout<<"\nAfter Change\n";
    cout<<"Co-ordinates of S1 Point\n";
    S1.display();
    cout<<"Co-ordinates of S2 Point\n";
    S2.display();
    return 0;
}
```

8 b) Explain the concept of multilevel inheritance with a suitable C++ program. (6M)

Ans: Multilevel Inheritance Concept and Diagram (2M)

When the inheritance is such that, class A serves as a base class for a derived class B which in turn serves as a base class for the derived class C. This type of inheritance is called 'MULTILEVEL INHERITANCE'. Class B is known as the 'INTERMEDIATE BASE CLASS' since it provides a link for the inheritance between A and C. The chain ABC is called 'INHERITANCE PATH'.

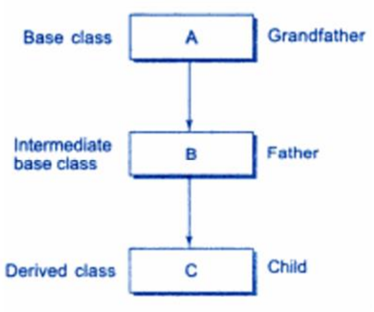


Fig ⇔ *Multilevel inheritance*

A derived class with multilevel inheritance is declared as follows:

```
class A {…..};              // Base class
class B: public A {…..};    // B derived from A
class C: public B {…..};    // C derived from B
```

This process can be extended to any number of levels.

Example Program (4M) – Model program to illustrate multilevel inheritance.

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

class Student
{
protected:
    int rno;
public:
    void getno(int);
    void putno(void);
};
```

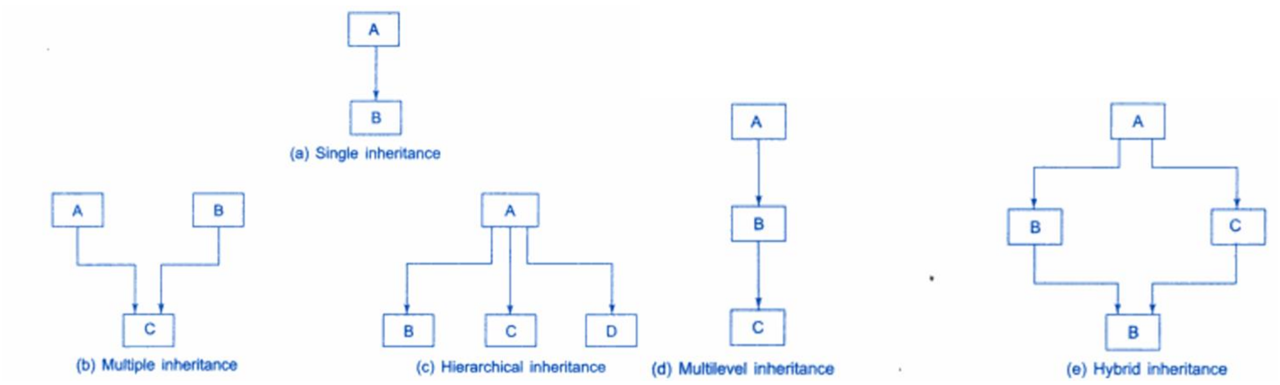
```
void Student::getno(int r)
{
   rno=r;
}
void Student::putno(void)
{
   cout<<"Roll Number: "<<rno<<endl;
}
class Test: public Student
{
protected:
   float sub1,sub2;
public:
   void getmarks(float,float);
   void putmarks(void);
};
void Test::getmarks(float s1,float s2)
{
   sub1=s1;
   sub2=s2;
}
void Test::putmarks(void)
{
   cout<<"Subject1 Marks: "<<sub1<<endl;
   cout<<"Subject2 Marks: "<<sub2<<endl;
}
class Result:public Test
{
   float total;
public:
   void display(void);
};
void Result::display()
{
   total=sub1+sub2;
   putno();
   putmarks();
   cout<<"Total Marks: "<<total<<endl;
}
int main()
{
   Result student1;
   student1.getno(71);
   student1.getmarks(64.0,78.0);
   student1.display();
   return 0;
}
```

(OR)

9 a) Describe Inheritance. State the types with a diagram for each and syntax for using different inheritances. (6M)

Ans: Inheritance (1M) - The mechanism of deriving a new class from an existing class is called 'INHERITANCE'. This is often referred to as 'IS-A' relationship because every object of the class being defined "is" also an object of inherited class. The existing class is called 'BASE' class, and the new class is called 'DERIVED' class.

Inheritance Types with Diagram (2M) – At least 4 types of Inheritances.



Syntax of at least 3 Inheritance Types (3M) – Even for writing programs marks can be given.

Single Inheritance Syntax:
class XYZ //base class
{
members of XYZ
};
class ABC: public XYZ //derived class
{
members of ABC
};
	Multi-Level Inheritance Syntax:
	class A // Base class
	{
	members of A
	};
	class B: public A // B derived from A
	{
	members of B
	};
	class C: public B // C derived from B
	{
	members of C
	};

Multiple Inheritance Syntax:
class A // Base class1
{
members of A
};
class B // Base class2
{
members of B
};
class C: public A, public B // derived class
{
members of C
};
	Hybrid Inheritance Syntax:
	class A // Base class
	{
	members of A
	};
	class B: public A // B derived from A
	{
	members of B
	};

Hierarchical Inheritance Syntax:
 	
class A // Base class
{
members of A
};
class B: public A // B derived from A
{
members of B
};
class C: public A // C derived from A
{
members of C
};
	class C
	{
	members of C
	};
	class D: public B, public C //derived class
	{
	members of B
	};

9 b) Write a program to overload the + operator to concatenate two strings. (6M)

Ans: Model program to overload the + operator to concatenate two strings (6M)

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

class String
{
   char *p;
   int len;
public:
   String() {len=0; p=0;}    //create null string
   String(const char *s);    //create string from character array
   String(const String & s); //copy constructor
   ~String() {delete []p;}     //destructor
   //overload + operator to concatenate strings
   friend String operator +(String &,String &);
   void show()
   {
      cout<<"Length: "<<len;
      cout<<"  String: "<<p<<endl;
   }
};

String::String(const char *s)
{
   len=strlen(s);
   p = new char[len+1];
   strcpy(p,s);
}

String::String(const String &s)
{
   len = s.len; // strlen(s.p);
   p = new char[len+1];
```

```
    strcpy(p,s.p);
}
String operator +(String &s, String &t)
{
    int l;
    l = strlen(s.p) + strlen(t.p);
    char* str = new char[l+1];
    strcpy(str,s.p);
    strcat(str,t.p);
    String temp(str);
    delete []str;
    return temp;
}
int main()
{
    String S1("New"),S2(" Delhi"),S3;
    S3 = S1 + S2;
    cout<<"String1\n";
    S1.show();
    cout<<"String2\n";
    S2.show();
    cout<<"Concatenated String\n";
    S3.show();
    int r;
    return 0;
}
```

Scheme Prepared By:

Mr. T.Y. Srinivasa Rao,
Assistant Professor,
Department of CSE,
Bapatla Engineering College.

Signature of the HOD, CSE Dept.

Paper Evaluators:

S. No.	Name of the College	Name of the Examiner	Signature