

Experiment No:1

Aim: Write a program to display the mark sheet of N students using concept of array and structures in C.

Theory: Structure & Classes concept can be used. Encapsulation concept is also used.

Algorithm:

Step 1: Input no. of students n.

Step 2: Input no. of subjects m.

Step 3: Input subjects names , input roll no. ,input marks.

Step 4: Calculate average, percentage of students.

Step 5: Generate mark sheet report.

Sample Input:

Enter the number of students: 1

Enter the number of subjects : 5

Enter the Sub names: phy , chem. , biology, maths, English

Enter the Roll no.: 101

Enter the name of Students: Gauri

Enter the marks: 40, 45, 45, 46, 48

Sample Output:

Roll no	Name	Phy	Chem.	Bio	Maths	English	Total	Per	remark
101	Gauri	40	45	45	46	48	224	89.6	pass

Viva voice:

1. What is difference between structure and array.

Experiment No:2

Aim: Write a program to implement the concept of Encapsulation in C++.

Theory:

Encapsulation:

In Object Oriented Programming, encapsulation is an attribute of object design. It means that all of the object's data is contained and hidden in the object and access to it restricted to members of that class.

Algorithm:

[A]

Step 1: Define the Constructor .
Inches=0 & feet=0

Step 2: Define Set function
Void set()
Input feet , Inches

Step 3: [Define add function]
Distance add(distance d2)
1) [Create an Object] distance d3
2) d3.inches=inches+d2.inches
3) d3.feet= feet+ d2.feet
4) If (d3.inches>= 12)
 1. n=d3.inches/12
 2. d3.inches=d3.inches/12
 3. d3.feet=d3.feet+n
5) return(d3)

Step 4: [Define display ()]
Void display()

Step 5: stop

[B]

Main Program

Step 1: [Create objects of type class distance]

Distance dd1,dd2,dd3

Step 2: [Call function set() of class distance]

dd1.set()

Step 3: dd2. set()

Step 4: [Call function add through object dd1 and pass dd2 as argument]

dd3=dd1.add(dd2)

Step 5: [Call function display() to display feet & inches]

dd1.display()

Step 6: dd2. display()

Step 7: dd3. display()

Step 8: Stop

Sample Input:

Set values for d1

Enter the values 3 4

Set values for d2

Enter the values 5 10

Sample Output:

d1:

3 feet 4 inches

d2:

5 feet 10 inches

d3:

9 feet 2 inches

Viva voice:

1. What do you mean by Encapsulation.

Experiment No:3

Aim: Write a program on function overloading in C++

Theory:

Function overloading involves writing two or more functions that have the same name, but different parameter lists. Then, during execution of a program, picking which of the functions to use is left up to the compiler, and is based on the parameters that have been passed to the function.

For instance, let's say we have a Color Object, a Shape Object, and a Texture Object. Instead of formally explaining the objects, let's just say they can have the following values:

Color	Shape	Texture
Blue	Square	Smooth
Red	Circle	Rough
Green	Ellipse	Bumpy

When using these objects it would be nice to be able to generate random combinations and be able ...

```
Color Increment(Color old_color) {
    if (old_color == Blue)
        return Red;
    if (old_color == Red)
        return Green;
    if (old_color == Green)
        return Blue;
    return Blue;
}
```

Algorithm:

A]

Class geometry

Private variable:

int r=4

Step 1: [Define function area ()]

Void area()

Display(3.1414 * r*r)

Step 2: [Define function area(int , int)]

int area(int a, int b)

return(a,b)

Step 3: [Define function area(int)
float area (int p)
return(0.5 * p*8)

Step 4: stop

B] Main Program

Step 1: [Create object]
Geometry g

Step 2: [function call]
g.area()

Step 3: [function call]
a1=g.area(4)

Step 4: Display a1

Step 5: [function call]
a2=g.area(3,8)

Step 6:
Display a2

Step 7:
Stop

Sample Output:

Area of a Circle is 50.24

Area of a triangle is 16

Area of a rectangle is 24

Viva voice:

Experiment No:4

Aim: Write a program to implement the concept call by value & call by reference in C.

Theory:

Call by value:

Call-by-value evaluation is the most common evaluation strategy, used in languages as far-ranging as [C](#) and [Scheme](#). In call-by-value, the argument expression is evaluated, and the resulting value is bound to the corresponding variable in the function (usually by capture-avoiding substitution or by copying the value into a new memory region). If the function or procedure is able to assign values to its parameters, only the local copy is assigned -- that is, anything passed into a function call is unchanged in the caller's scope when the function returns.

Call-by-value is not a single evaluation strategy, but rather the family of evaluation strategies in which a function's argument is evaluated before being passed to the function. While many programming languages that use call-by-value evaluate function arguments left-to-right, some evaluate functions and their arguments right-to-left

Call by Reference:

In *call-by-reference* evaluation, a function is passed an implicit [reference](#) to its argument rather than the argument value itself. If the function is able to modify such a parameter, then any changes it makes will be visible to the caller as well. If the argument expression is an [L-value](#), its address is used. Otherwise, a temporary object is constructed by the caller and a reference to this object is passed; the object is then discarded when the function returns.

Some languages contain a notion of references as first-class values. [ML](#), for example, has the "ref" constructor; `[[reference (C++)|references in C++]]` may also be created explicitly. In these languages, "call-by-reference" may be used to mean passing a reference value as an argument to a function.

In languages (such as [C](#)) that contain [unrestricted pointers](#) instead of or in addition to references, *call-by-address* is a variant of call-by-reference where the reference is an unrestricted pointer.

Algorithm:

A] Function :- func (int p, int q)

1. p=p*5
- 2 q= q* 5
- 3 Display p and q

B] Function : func1(int *p, int * q)

1. *p= *p * 5
- 2 *q= *q * 5
3. Display *p and *q

C] Main Program

- Step1: a=5 , b=10
Step2: Display a, b
Step3: [Function call
 func(a ,b)
Step4: Display a,b
Step5: [Function call
 func1(&a, &b)
Step6: Display a, b
Step7: Stop

Sample Input:

Original values:
a= 5 , b= 10

Sample Output:

Call by value :

Values outside function

a=5 b=10

Call by reference

Values outside function

a=25 b=50

Viva voice:

1. Differentiate between Call by value and Call by reference.
2. What is pointer?

Experiment No:5

Aim: Write a program to implement inheritance in C++

Theory:

Inheritance:

Inheritance is a technique of organizing information in a hierarchical form . It is like a child inheriting the features of its parents . In real world , object is described by using inheritance . It derives general properties of an object by tracing an inheritance tree from one specific instance , upwards towards the primitive concept at the root.

Inheritance allows new classes to be built from older and less specified classes instead of being rewritten from scratch . Classes are created by first inheriting all the variables and behavior defined by some primitive class and then adding specialized variables and behaviors. In object oriented programming , classes encapsulate data and functions into one package . New classes can be built from existing ones, just as a builder constructs a skyscraper out of bricks, stone ,and other relatively simple material . The technique of building new classes is called inheritance.

Algorithm:

A] Class employee:

Protected variables:

Char name[18]; in temp-no

Step1: [Define function set()]

Void set()

1. Input name
2. Input emp_no

Step2: [Define function display ()]

Void display()

1. Display name
2. Display emp_no

Step 3: Stop

B] Class manager inherits employee

Protected variables:

Step 1: [define function set()]

Void set()

1. Input title
2. Input Idues

Step2: [Define function display()]

1. employee. Display()
2. Display.title
3. Display gdues

Step 3: Stop

C] Class scientist inherits employee
Protected variables.

Step1: [define function set()]

- Void set()
1. employee .set()
 2. Input n

Step 2: [Define function display()]

- Void display()
1. employee .display()
 2. Display n

Step 3: Stop

D] Class worker inherits employee

Step 1: [Define function set()]

- Void set()
1. employee. set()

Step 2: [define function set()]

- Void display()
1. employee. display()

Step 3: Stop

E] Class foreman inherits worker protected variables:

Step 1: [Define function set()]

- Void set()
1. Worker. Set()
 2. Input quota

Step2: [Define function display()]

Void display()

1. Worker. display()
2. Display quota

Step 3: Stop

F] Main Program

Step1 : [Create objects]

1. manager m
2. scientist s
3. worker w
4. foreman f

Step 2: [function calls]

1. m. set ()
2. s. set()
3. w. set()
4. f. set()

Step 3: [Function calls for display]

1. m. display()
2. s. display()
3. w. display()
4. f. display()

Step 4: Stop

Sample Input:

Manager :

Enter name : xyz

Enter employee no: 5

Enter title : chief manager

Enter golf club dues : 10

Scientist :

Enter name: abc

Enter employee no: 9

Enter no. of Scholarly articles published by scientist :12

Worker:

Enter name: ccc

Enter employee no :2

Foreman :

Enter name: ddd

Enter employee no: 20

Program No. 6

Aim : Write a program to implement inheritance.

Theory: Inheritance is ability of a class to inherit the properties of Base class. The derived class can add its own features.

Algorithm:

Step 1: Class employee

```
I ] getdata():- cin >> name
                cin >> empno
II] printdata ():- cout<<name
                  cout << empno
```

Step 2: Class worker inherits class employee

Step 3: Class manager inherits class employee

```
(I)      getdata:- employee::getdata()
                cin>>title
                cin >> due
(II)     printdata()
1.  employee::printdata()
    1.cout<<title
    2. cout<< due
```

Step 4: Class scientist inherits class employee

```
1. getdata()
   1. employee::getdata()
     cin>> article
2. printdata()
   employee::printdata()
   cout<<article
```

Step 5: Class foreman inherits class employee

```
getdata()
1. employee::getdata()
   cin>> quota
2. printdata()
   1. employee::printdata()
   cout<<quota
```

Step 6: Main Program

```

1. x=1
2. while (x!=5)
    1.cin(x)
    2. switch(x)
        1.case 1
            1. w.getdata()
            2. w.printdata()
        2. Case 2
            1.m.getdata()
            2.m.printdata()
        3. Case 3
            1. s.getdata()
            2. s.printdata()
        4. Case 4
            1. f.getdata( )
            2.f.printdata ( )
    3. end while

```

Step 7: Stop.

Input & Output:

1. Worker 2. Manager 3. Scientist 4. Foreman 5. Exit

1

Enter name: kaustabh

Enter employee no: 144

Name: Kaustabh

Emp. No. 148

1. Worker 2. Manager 3. Scientist 4. Foreman 5. Exit

2

Enter name:Nikhil

Enter employee no:

Enter title: CEO

Enter golf dues 10,000

Name :Nikhil

Emp. No. 4

Title: CEO

Golf dues:1000

Program No. 7

Aim: Write a program to convert infix to postfix.

Theory: In infix expression operators are between operands & in postfix expression operators are arranged or shifted after all the operands.

Algorithm:

Step 1: [Main Program]

1. gets (str)
2. s = str
3. t = target
4. while (*s)
 1. if (*s == '\t')
 1. s ++
 2. end if
 3. if (isdigit (*s) || isalpha (*s))
 1. while (isdigit (*s) || isalpha (*s))
 1. *t = *s
 2. t ++
 3. s ++
 2. end while.
 4. end if
 5. if (*s == 'c')
 1. push (stack, & top, *s)
 2. s ++
 6. end if
 7. if (*s == 'c')
 1. n, = top (stack, & top)
 2. while (n, = 'c')
 1. *t = n,
 2. t ++
 3. top (stack & top)
 3. end while.
 4. s ++
 8. end if
 9. if (*s == '+' || *s == '*' || *s == '-')
 1. if (top == -1)
 1. push ((stack, & top, *s)
 2. else
 1. n, = top (stack, & top)
 2. while (priority (n1) >= priority (*s))
 1. *t = n,
 2. t ++
 3. n, = top (stack & top)
 3. end while.
 4. push ((stack, & top, n,)
 5. push ((stack, & top, *s)
 1. end if

```

    2. s++
10. end if
11. end while.
12. while (top! = - 1)
    1. n, = top (stack & top)
    2. *t = n,
    3. t++
13. end while
14. *t = '10'
15. t = target
16. while (*t)
    1. printf ( "% c,"*t)
    2. t++
5. end while
Step2: [ function priority]
    1. if (ele == '*' || ele == '1' || ele == '-')
        1. pri = 1
        2. else
            1. if (ele == '*' || ele == '1')
                1. pri = 0
            3. end if
            3. end if
        4. return pri

```

Step3: [push]

```

    1. if (*st == max) print stack is full
    2. else
        1. * st = * st +1;
        2. stk [*st] = item;
        3. end if

```

Step4: [pop]

```

    1. if (*st == - 1)
        1. return (-1)
    2. else
        1. item = stk [* st]
        2. * st = * st -1
        3. return (item)
    3. end if.

```

4. Output :

Enter an infix expression

(a + b)* (a - b)

Post fix expression is

ab+ ab - *

Program no. 8

Aim: Write a program to implement linked list.

Algorithm :

Step1: [function add()]

1. f1 = new node
2. cin (f1→data)
3. f1→ link = NULL
4. if(head = NULL)
 1. head = f1
 2. curr = head
5. else
 1. f2 = head
 2. while (f1→ link! = NULL)
 1. f1=f2→ link
 3. f2→ link = f1

Step2: [function print()]

1. f1 = head
2. while (f1!= NULL)
 1. cout (f1→ data)
 2. . f1=f1→ link;
3. end while.
- 4.

Step3:[function insert()]

1. f1 = new node
2. cin (f1→data,z)
3. f1 = head
4. while (f2→ data! = z)
 1. f2=f2→ link
5. end while
6. f1→ link= f2→ link
7. f2→ link = f1

Step4: [function delete]

1. cin (y)
2. f1 = head
3. while (f1→ data! = y)
 1. f2=f1
 2. f1=f1→ link
4. end while
5. f2→ link = f1→ link
6. delete (f1)

Step4:[main program]

1. choice =0

2. while(choice! =5)
 1. cin (choice)
 2. switch (choice)
 1. case 1
 1. add ()
 2. case 2
 1. print ()
 3. case 3
 1. insert()
 4. case 4
 1. del ()
3. end while

Output :

1. Create 2. Print 3. Insert 4. Delete.
1. Enter an element
4
1. Enter an element
7
2.
The list is
4 7
End of list
- 4
The list is
4
End of list
- 4
The list is empty.

Program no. 9

Aim: Write a program to implement the concept of function overloading in C++.

Theory: Functions having same name but perform different functions.

Algorithm :**Step1:** [start]**Step2:** Select a choice.**Step3:** 1. case (choice) of
 1. accept (base, height, area(base, height))
 2. base = base* height* 0.5
 3. cout (base)
2. cin (length, breadth)
 1. area (length, breadth)
 2. length = length* breadth
 3. cout (length)
3. cin (radius)
 1. area (radius)
 2. radius = 3.14* radius* radius.
 3. cout (radius)**Output :**

1. Triangle 2: Rectangle 3: Circle 4: Exit
3
Enter radius
1
Area of circle = 3.14
4

Program no. 10**Aim :** Write a program to implement all referencing environments.**Theory:** Three types of referencing environments are available: local, global

& nonlocal.

Algorithm :

Step1: [function dint l, m)

1. $d = l / m$
2. cout (d)
3. cout (d x/y)
4. cout (creation of new set of associations).

Step2: [subtract]

1. $s = q - w$
2. cout (s)
3. cout (s = x-y)
4. cout (Information of local referencing omicronment)
5. din (q, w)

Step3: [function product]

1. $p = r * s$
2. cout (p)
3. cout ($p = r * s$)
4. cout (Set of associations are created an entry to subprogram).
5. subtract (r, s)

Step4: [main program]

1. cin (x, y)
2. $sum = x + y$
3. cout (sum)
4. cout (Information of global referencing omicronment)
5. product (x.y)

Step5: [stop]

Output :

Enter 2 numbers

5 3

sum = 8

sum = x +y

These associations are created for x,y and sum these associations from global referencing environment.

Product = 15

$P = r * s$

Set of associations are created on entry to subprogram.

That represents formal parameters and local variables.

Subtraction = 2

$S = x - y$

Activation records form local referencing environment.

Since references are created for nested subprogram.

Division = 1

$d = x / y$

Nesting of all subprogram access which creates new set of associations.
new set of associations.

All activation records are destroyed.