

Principles of Programming Languages

Paper code : ETHS-402



**Maharaja Agrasen
Institute Of Technology**

PSP Area, Sector-22, Rohini, New Delhi -110086

Submitted To :

Mr. Mool Chand Sharma
(Assistant Professor)
(Department of Computer Science)

Submitted By :

Pavitra Walia
(Enrollment Number : 42414802718)
(Branch : Computer science)
(Semester : 8th)
(Batch : 8C789)

Index

Exp. no	Experiment Name	Date of performance	Date of checking	Marks	Signature
1	Implement all major functions of string.h in single C program using switch case to select specific function from user choice (like strlen, strcat, strcpy, strcmp, strev)				
2	Write a program (WAP) in C to reverse a linked list iterative and recursive.				
3	WAP in C to implement iterative Towers of Hanoi.				
4	WAP in C++ to count the no.s of object of a class with the help of static data member, funtion and constructor.				
5	WAP in C++ & Java to declare a class Time with data members mm for minutes, ss for seconds and hh for hours. Define a parameterize constructor to assign time to its objects. Add two time objects using member function and assign to third objects. Implement all possible cases of time.				
6	WAP in C++ to define a class Complex to represents set of all complex numbers. Overload '+' operator to add two complex numbers using member function of the class and overload '*' operator to multiply two complex numbers using friend function of the class complex				
7	Implement simple multi-threaded server to perform all mathematics operation parallel in Java.				

Experiment 1

Aim : Implement all major functions of string.h in single C program using switch case to select specific function from user choice (like strlen, strcat, strcpy, strcmp, strrev)

Code :

```
#include<stdio.h>
#include<string.h>
void main(){
char str[50];
char temp[20];
char choice, ch;
//printf("For ")
printf("Pavitra Walia 42414802718 \n");
puts("To get the length of string, choose 'L'."); //strlen();//
puts("To convert the whole string in lower case, choose 'l'."); //strlwr();
puts("To convert the whole string in upper case, choose 'U'."); //strupr();
puts("To append a string behind other, choose 'A'."); //strcat();//
puts("To copy a string into another, choose 'c'."); //strcpy();//
puts("To compare two strings, choose 'C'."); //strcmp();//
puts("To find out first ocurence of given character in a string, choose 'O'");
//strchr();
puts("To find out first ocurence of given string in another string, choose 'S'");
//strstr();
puts("To reverse the string, choose 'R'"); //strrev();//
printf("\nEnter Your Choice: ");
scanf("%c", &choice);
switch(choice){
case 'L':
printf("\nEnter The String To Get Its Length: ");
scanf("%s", str);
printf("The Length Of The Entered String Is: %d", strlen(str));
break;
case 'l':
printf("\nEnter The String To Convert It Into Lower Case : ");
scanf("%s", str);
printf("The Entered String In Lowercase: %s", strlwr(str));
break;
case 'U':
printf("\nEnter The String To Convert It Into Upper Case : ");
scanf("%s", str);
printf("\nThe Entered String In Lowercase: %s", strupr(str));
break;
case 'A':
printf("\nEnter The First String: ");

scanf("%s", str);
printf("Enter The Second String To Append It Behind First One:");

scanf("%s", temp);
```

```

strcat(str, temp);
printf("\nNow, The First String Is: %s", str);
break;
case 'c':
printf("\nEnter The First String: ");
scanf("%s", str);
printf("Enter The Second String: ");
scanf("%s", temp);
strcpy(str, temp);
printf("\nNow, The First String Is: %s", str);
printf("\nAnd, The Second String Is: %s", temp);
break;
case 'C':
printf("\nEnter The First String: ");
scanf("%s", str);
printf("Enter The Second String: ");
scanf("%s", temp);
if(strcmp(str, temp)==0)printf("\nBoth Strings Are Similar.");
else printf("\nBoth Strings Are Different.");
break;
case 'O':
printf("\nEnter The String: ");
scanf("%s", str);
printf("Enter The Character To Be Searched: ");
scanf("%c", &ch);
printf("\nThe First Occurence of Character Is At: %s", strchr(str,ch));

break;
case 'S':
printf("\nEnter The String: ");
scanf("%s", str);
printf("Enter The String To Be Searched: ");
scanf("%s", temp);
printf("\nThe First Occurence of Character Is At: %s", strstr(str,temp));

break;
case 'R':
printf("\nEnter The String To Get Its Reverse: ");
scanf("%s", str);
printf("\nThe Reverse Of The Entered String Is: %s", strrev(str));
break;
default:

printf("\nYou Entered A Wrong Choise.");
break;

}
}

```

Output :

```
C:\Users\pavit\Downloads\exp1.exe
Pavitra Walia 42414802718
To get the length of string, choose 'L'.
To convert the whole string in lower case, choose 'l'.
To convert the whole string in upper case, choose 'U'.
To append a string behind other, choose 'A'.
To copy a string into another, choose 'c'.
To compare two strings, choose 'C'.
To find out first ocurence of given character in a string, choose 'O'
To find out first ocurence of given string in another string, choose 'S'
To reverse the string, choose 'R'

Enter Your Choice: L

Enter The String To Get Its Length: principle
The Length Of The Entered String Is: 9
Process returned 38 (0x26)   execution time : 16.704 s
Press any key to continue.
```

```
C:\Users\pavit\Downloads\exp1.exe
Pavitra Walia 42414802718
To get the length of string, choose 'L'.
To convert the whole string in lower case, choose 'l'.
To convert the whole string in upper case, choose 'U'.
To append a string behind other, choose 'A'.
To copy a string into another, choose 'c'.
To compare two strings, choose 'C'.
To find out first ocurence of given character in a string, choose 'O'
To find out first ocurence of given string in another string, choose 'S'
To reverse the string, choose 'R'

Enter Your Choice: C

Enter The First String: test
Enter The Second String: test

Both Strings Are Similar.
Process returned 26 (0x1A)   execution time : 14.875 s
Press any key to continue.
```

Viva Voce

Q1. What is the use of string.h header while and where is this file stored.

In C programming language, string.h is a header that defines one variable type, one macro, and various functions for manipulating arrays of characters. All C inbuilt functions are declared in string.h header file.

Q2. How can we create a header file?

To make a header file, we have to create one file with a name, and the extension should be (*.h). In that function, there will be no main() function. In that file, we can put some variables, some functions, etc.

Q3. Write the function to find the length of a string.

The strlen() function in C is used to calculate the length of a string.

Q4. Write the function to concatenate two strings.

The best way to concatenate two strings in C programming is by using the strcat() function.

Q5. Explain the use of the header file.

Header files are simply files in which you can declare your own functions that you can use in your main program or these can be used while writing large C programs. The header file implementation brings lots of readability to our program and it becomes easy to understand. If it is the way to write our code systematically and the header file brings abstraction, standardization, and loose coupling between our main function file(.c) and other (.c) files which we are using.

Experiment 2

Aim : Write a program (WAP) in C to reverse a linked list iterative and recursive.

Code :

```
// Iterative C program to reverse a linked list
#include<stdio.h>
#include<stdlib.h>
/* Link list node */

struct Node
{
    int data;
    struct Node* next;
};

/* Function to reverse the linked list */
void recursiveReverse(struct Node* head, struct Node** headRef)
{
    struct Node* first;
    struct Node* rest;
    // empty list base case
    if (head == NULL)
        return;
    first = head; // suppose first = {1, 2, 3}
    rest = first->next; // rest = {2, 3}
    // base case: List has only one node
    if (rest == NULL)
    {
        // fix the head pointer here
        *headRef = first;
        return;
    }
    // Recursively reverse the smaller {2, 3} case
    // after: rest = {3, 2}
    recursiveReverse(rest, headRef);
    // put the first elem on the end of the list
    rest->next = first;
    first->next = NULL; // (tricky step -- make a drawing)
}

void reverse(struct Node** head_ref)

{
    struct Node* prev = NULL;
    struct Node* current = *head_ref;
    struct Node* next = NULL;
    printf("\n\nReversing Linked List Iteratively...\n");
    while (current != NULL)
    {
        // Store next
        next = current->next;
```

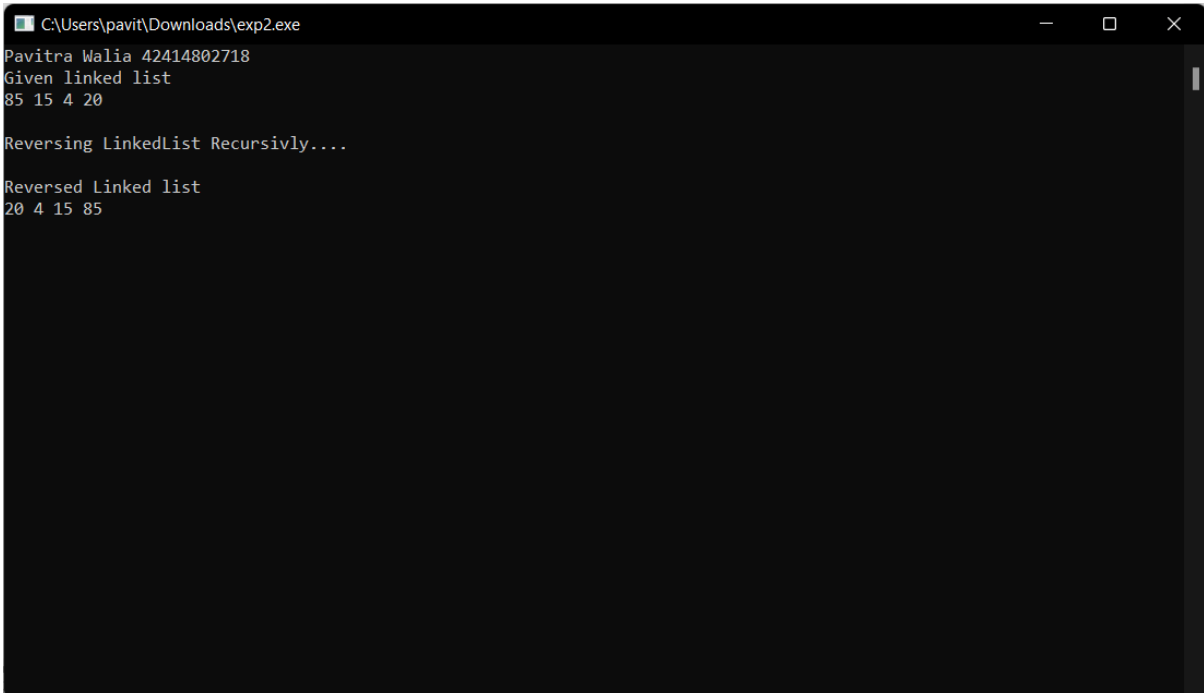
```

// Reverse current node's pointer
current->next = prev;
// Move pointers one position ahead.
prev = current;
current = next;
}
*head_ref = prev;
}
/* Function to push a node */
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}
/* Function to print linked list */
void printList(struct Node *head)
{
    struct Node *temp = head;
    while(temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}
int main()
{
    printf("Pavitra Walia 42414802718 \n");
    /* Start with the empty list */
    struct Node* head = NULL;
    push(&head, 20);

    push(&head, 4);
    push(&head, 15);
    push(&head, 85);
    printf("Given linked list\n");
    printList(head);
    //reverse(&head);
    printf("\n\nReversing LinkedList Recursivly...\n");
    recursiveReverse(head, &head);
    printf("\nReversed Linked list \n");
    printList(head);
    getchar();
}

```


Output :



```
C:\Users\pavit\Downloads\exp2.exe
Pavitra Walia 42414802718
Given linked list
85 15 4 20

Reversing LinkedList Recursivly...

Reversed Linked list
20 4 15 85
```

The image shows a Windows command prompt window with a black background and white text. The title bar at the top reads "C:\Users\pavit\Downloads\exp2.exe". The window contains the following text: "Pavitra Walia 42414802718", "Given linked list", "85 15 4 20", "Reversing LinkedList Recursivly...", "Reversed Linked list", and "20 4 15 85". A vertical scrollbar is visible on the right side of the window.

Viva Voce

Q1. What is the difference between iterative and recursive function call?

Recursion and iteration are both different ways to execute a set of instructions repeatedly. The main difference between these two is that in recursion, we use function calls to execute the statements repeatedly inside the function body, while in iteration, we use loops like “for” and “while” to do the same. Iteration is faster and more space-efficient than recursion, whereas it's easier to code a recursive approach for a given problem.

Q2. What are formal parameters in functions?

Formal parameters are the variables defined by the function that receives values when the function is called.

Q3. How is the structure node declared?

The general syntax for a struct declaration in C is:

```
struct tag_name {  
    type member1;  
    type member2;  
    /* declare as many members as desired, but the entire structure size must be known to the  
    compiler. */  
};
```

Q4. Define a link list.

A linked list is a sequence of data structures, which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array.

Q5. Why do we need to store the address of the starting node of a link list for reversing a list?

To reverse a LinkedList iteratively, we need to store the references of the next and previous elements, so that they don't get lost when we swap the memory address pointers to the next element in the LinkedList.

Experiment 3

Aim : WAP in C to implement iterative Towers of Hanoi.

Code :

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <limits.h>
// A structure to represent a stack
struct Stack
{
    unsigned capacity;
    int top;
    int *array;
};
// function to create a stack of given capacity.
struct Stack* createStack(unsigned capacity)
{
    struct Stack* stack =
        (struct Stack*) malloc(sizeof(struct Stack));
    stack->capacity = capacity;
    stack->top = -1;
    stack->array =
        (int*) malloc(stack->capacity * sizeof(int));
    return stack;
}
// Stack is full when top is equal to the last index
int isFull(struct Stack* stack)
{
    return (stack->top == stack->capacity - 1);
}
// Stack is empty when top is equal to -1
int isEmpty(struct Stack* stack)
{
    return (stack->top == -1);
}
// Function to add an item to stack. It increases
// top by 1

void push(struct Stack *stack, int item)
{
    if (isFull(stack))
        return;
    stack->array[++stack->top] = item;
}
// Function to remove an item from stack. It
// decreases top by 1
int pop(struct Stack* stack)
{
    if
```

```

if (isEmpty(stack))
return INT_MIN;
return stack -> array[stack -> top--];
}
//Function to show the movement of disks
void moveDisk(char fromPeg, char toPeg, int disk)
{
printf("Move the disk %d from \'%c\' to \'%c\'\n",
disk, fromPeg, toPeg);
}
// Function to implement legal movement between
// two poles
void moveDisksBetweenTwoPoles(struct Stack *src,
struct Stack *dest, char s, char d)
{
int pole1TopDisk = pop(src);
int pole2TopDisk = pop(dest);
// When pole 1 is empty
if (pole1TopDisk == INT_MIN)
{
push(src, pole2TopDisk);
moveDisk(d, s, pole2TopDisk);
}
// When pole2 pole is empty
else if (pole2TopDisk == INT_MIN)
{
push(dest, pole1TopDisk);
moveDisk(s, d, pole1TopDisk);
}
// When top disk of pole1 > top disk of pole2

else if (pole1TopDisk > pole2TopDisk)
{
push(src, pole1TopDisk);
push(src, pole2TopDisk);
moveDisk(d, s, pole2TopDisk);
}
// When top disk of pole1 < top disk of pole2
else
{
push(dest, pole2TopDisk);
push(dest, pole1TopDisk);
moveDisk(s, d, pole1TopDisk);
}
}
//Function to implement TOH puzzle
void tohIterative(int num_of_disks, struct Stack
*src, struct Stack *aux,
struct Stack *dest)
{
int i, total_num_of_moves;
char s = 'S', d = 'D', a = 'A';

```

```

//If number of disks is even, then interchange
//destination pole and auxiliary pole
if (num_of_disks % 2 == 0)
{
    char temp = d;
    d = a;
    a = temp;
}
total_num_of_moves = pow(2, num_of_disks) - 1;
//Larger disks will be pushed first
for (i = num_of_disks; i >= 1; i--)
    push(src, i);
for (i = 1; i <= total_num_of_moves; i++)
{
    if (i % 3 == 1)
        moveDisksBetweenTwoPoles(src, dest, s, d);
    else if (i % 3 == 2)
        moveDisksBetweenTwoPoles(src, aux, s, a);

    else if (i % 3 == 0)
        moveDisksBetweenTwoPoles(aux, dest, a, d);
}
}
// Driver Program
int main()
{
    printf("Pavitra Walia 42414802718 \n");
    // Input: number of disks
    unsigned num_of_disks = 3;
    struct Stack *src, *dest, *aux;
    // Create three stacks of size 'num_of_disks'
    // to hold the disks
    src = createStack(num_of_disks);
    aux = createStack(num_of_disks);
    dest = createStack(num_of_disks);
    tohIterative(num_of_disks, src, aux, dest);
    return 0;
}

```

Output :

```
C:\Users\pavit\Downloads\exp3.exe
Pavitra Walia 42414802718
Move the disk 1 from 'S' to 'D'
Move the disk 2 from 'S' to 'A'
Move the disk 1 from 'D' to 'A'
Move the disk 3 from 'S' to 'D'
Move the disk 1 from 'A' to 'S'
Move the disk 2 from 'A' to 'D'
Move the disk 1 from 'S' to 'D'

Process returned 0 (0x0)   execution time : 0.687 s
Press any key to continue.
_
```

Viva Voce

Q1. What is the tower of Hanoi problem?

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

Q2. What are the various ways is stack created?

There are two ways to implement a stack: Using array. Using linked list.

Q3. List the models of computation of language.

Models of computation can be classified into three categories: sequential models, functional models, and concurrent models.

Q4. What are objectives of principle of programming language?

Objectives are: To introduce several different paradigms of programming. To gain experience with these paradigms by using example programming languages. To understand concepts of syntax, translation, abstraction, and implementation.

Q5. What are the Paradigms of Programming?

Major Programming Paradigms

- Imperative.
- Logical.
- Functional.
- Object-Orient

Experiment 4

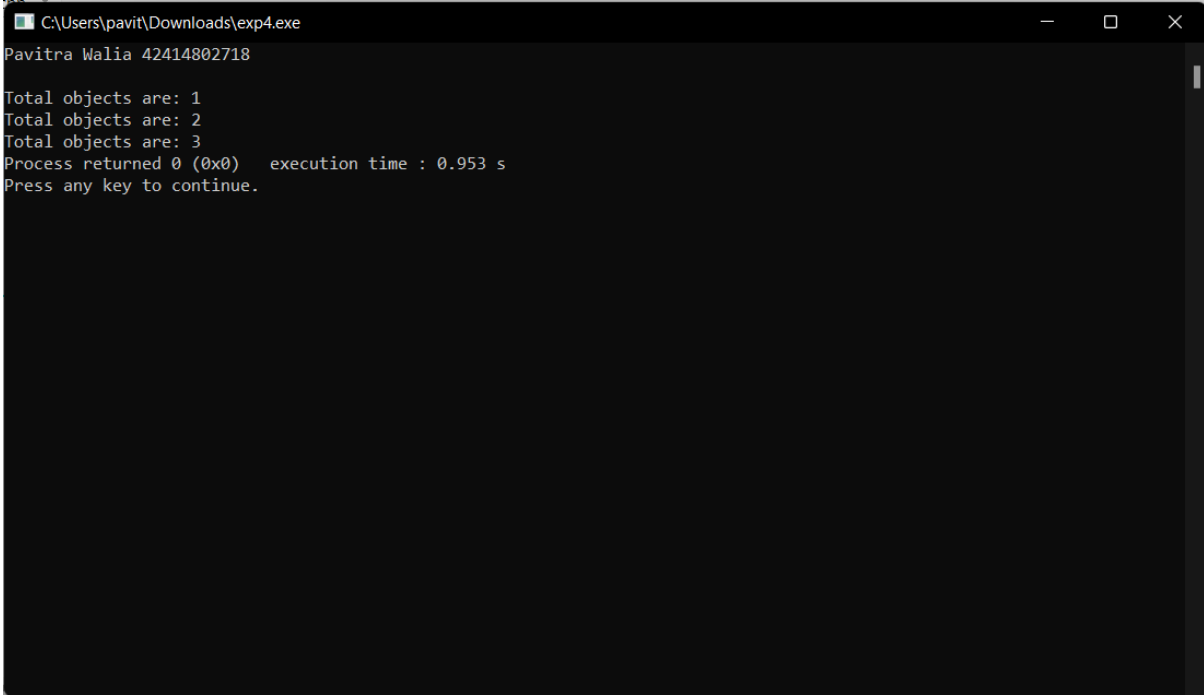
Aim : WAP in C++ to count the no.s of object of a class with the help of static data member, function and constructor

Code :

```
#include <iostream>
using namespace std;
class Counter
{
private:
//static data member as count
static int count;
public:
//default constructor
Counter()
{ count++; }
//static member function
static void Print()
{
cout<<"\nTotal objects are: "<<count;
}

};
//count initialization with 0
int Counter :: count = 0;
int main()
{
printf("Pavitra Walia 42414802718 \n");
Counter OB1;
OB1.Print();
Counter OB2;
OB2.Print();
Counter OB3;
OB3.Print();
return 0;
}
```


Output :



```
C:\Users\pavit\Downloads\exp4.exe
Pavitra Walia 42414802718
Total objects are: 1
Total objects are: 2
Total objects are: 3
Process returned 0 (0x0)   execution time : 0.953 s
Press any key to continue.
```

The image shows a Windows command prompt window with a dark background. The title bar at the top reads "C:\Users\pavit\Downloads\exp4.exe" and includes standard minimize, maximize, and close buttons. The command prompt displays the following text: "Pavitra Walia 42414802718", followed by three lines of "Total objects are:" with values 1, 2, and 3. Below these is the message "Process returned 0 (0x0) execution time : 0.953 s" and "Press any key to continue." at the bottom. A vertical scrollbar is visible on the right side of the window.

Viva Voce

Q1. List various type of languages.

The different types of programming languages are:

- Procedural Programming Language
- Functional Programming Language
- Object-oriented Programming Language
- Scripting Programming Language
- Logic Programming Language

Q2. What are the issues for languages?

Issues are:

- Development on hardware technologies
- Simplicity and performance trade off
- Usage Area
- Portability
- Variety of Project's Design Patterns
- Safety and security

Q3. What is translation?

Translation is a programming language processor that modifies a computer program from one language to another. It takes a program written in the source program and modifies it into a machine program. It can find and detect the error.

Q4. What are different types of translation and their roles?

Generally, there are three types of translators -

- Compilers - A compiler takes the source code as a whole and translates it into object code all in one go.
- Interpreters - An interpreter translates source code into object code one instruction at a time.
- Assemblers - The purpose of an assembler is to translate assembly language into object code.

Experiment 5

Aim : WAP in C++ & Java to declare a class Time with data members mm for minutes, ss for seconds and hh for hours. Define a parameterize constructor to assign time to its objects. Add two time objects using member function and assign to third objects. Implement all possible cases of time

Code :

C++

```
#include<iostream>
using namespace std;
class Time
{
    int hh,mm,ss;
public:
    Time(){}
    Time(int hh, int mm, int ss)
    {
        this->hh=hh;
        this->mm=mm;
        this->ss=ss;
    }
    void disp()
    {
        cout<<hh<<": "<<mm<<": "<<ss;
    }
    void sum(Time t1,Time t2)
    {
        ss=t1.ss+t2.ss;
        mm=ss/60;
        ss=ss%60;
        mm=mm+t1.mm+t2.mm;
        hh=mm/60;
        mm=mm%60;
        hh=hh+t1.hh+t2.hh;
    }
};

int main(){
    printf("Pavitra Walia 42414802718 \n");
    Time t1(2,22,34);
    cout<<"The Time T1 Is: ";
    t1.disp();
    Time t2(4, 33, 50);
    cout<<"\n\nThe Time T2 Is: ";
    t2.disp();
    Time t3;
    t3.sum(t1,t2);
```

```
cout<<"\n\nThe Resultant Time Is: ";
t3.disp();
}
```

JAVA

```
import java.io.*;
class time
{
    int hour,min,sec;

    public time(int h,int m,int s)
    {
        hour = h;
        min = m;
        sec = s;
    }
    public time()
    {
        hour = 0;
        min = 0;
        sec = 0;
    }
    public time sum(time t2)
    {
        time t3 = new time();
        t3.sec = sec + t2.sec;
        t3.min = min + t2.min;
        t3.hour = hour + t2.hour;

        if(t3.sec >= 60)
        {
            t3.min = t3.sec / 60;
            t3.sec = t3.sec % 60;
        }
        if(t3.min >= 60)
        {
            t3.hour = t3.min / 60;
            t3.min = t3.min % 60;
        }
        return(t3);
    }
    public void display()
    {
        System.out.println(hour+ ":"+min+":"+sec);
    }
}

public class Main {
    public static void main(String args[])throws IOException
    {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

```

        System.out.print ("Pavitra Walia 42414802718 \n\n" );

        time t1 = new time(1,56,55);
        time t2 = new time(2,00,10);
        time t3 = new time();

        t3 = t1.sum(t2);
        System.out.print ("Time 1:- " );
        t1.display();
        System.out.print ("Time 2:- " );
        t2.display();
        System.out.print ("Resultant time after addition is:- " );
        t3.display();
    }
}

```

Output :

C++

```

C:\Users\pavit\Downloads\exp5.exe
Pavitra Walia 42414802718
The Time T1 Is: 2:22:34

The Time T2 Is: 4:33:50

The Resultant Time Is: 6:56:24
Process returned 0 (0x0)   execution time : 4.415 s
Press any key to continue.

```

JAVA

```

Pavitra Walia 42414802718

Time 1:- 1:56:55
Time 2:- 2:0:10
Resultant time after addition is:- 3:1:5

```

Viva Voce

Q1. Write any four important uses of programming languages.

A programming language is an artificial language that can be used to control the behaviour of a machine, particularly a computer. Programming languages, like human languages, are defined through the use of syntactic and semantic rules, to determine structure and meaning respectively. Programming languages are used to facilitate communication about the task of organizing and manipulating information, and to express algorithms precisely.

Q2. Write the differences between lexical syntax and concrete syntax of the language.

Lexical syntax for defining the rules for basic symbols involving identifiers, literals, punctuators and operators. Concrete syntax specifies the real representation of the programs with the help of lexical symbols like its alphabet.

Q3. List the design principle of imperative languages.

Imperative programming is a software development paradigm where functions are implicitly coded in every step required to solve a problem. In imperative programming, every operation is coded and the code itself specifies how the problem is to be solved, which means that pre-coded models are not called on.

Q4. Write the differences between array and enumerated data types in imperative languages?

The main difference is that an array is a value and an enum is a type. And One main difference we can say that an array is a collection of other values (that is it contains other values, you can iterate through them or access individual ones by index), whereas an enum value is simply one atomic value.

Q5. Distinguish between dangling pointers and memory leakage.

Generally, dangling pointers arise when the referencing object is deleted or deallocated, without changing the value of the pointers. In opposite to the dangling pointer, a memory leak occurs when you forget to deallocate the allocated memory.

Experiment 6

Aim : WAP in C++ to define a class Complex to represents set of all complex numbers. Overload '+' operator to add two complex numbers using member function of the class and overload '*' operator to multiply two complex numbers using friend function of the class complex.

Code :

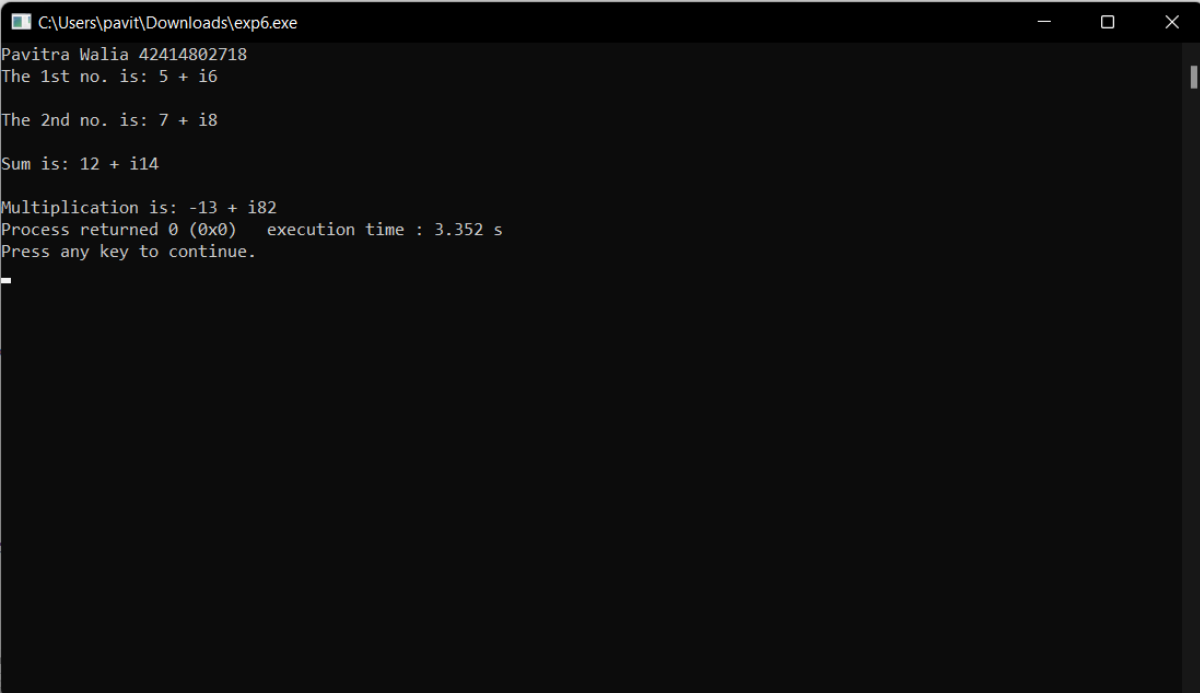
```
#include<iostream>
using namespace std;
class Complex
{
    int real,img;
public:
    Complex(){};
    Complex(int i,int j)
    {
        real=i;
        img=j;
    }
    void show()
    {
        cout<<real<<" + i"<<img;
    }
    Complex operator +(Complex obj){
        Complex temp;
        temp.real=real+obj.real;
        temp.img=img+obj.img;
        return(temp);
    }
    Complex operator *(Complex);
};

Complex Complex::operator *(Complex c)
{
    double real1,real2;
    real1=real;
    real2=c.real;
    real=(real*c.real)-(img*c.img);
    img=(real1*c.img)+(img*real2);
    Complex temp;
    temp.real=real;
    temp.img=img;
    return temp;
}

int main()
{
    printf("Pavitra Walia 42414802718 \n");
    Complex c1(5,6), c2(7,8), c3, c4;
    cout<<"The 1st no. is: ";
```

```
c1.show();
cout<<"\n\nThe 2nd no. is: ";
c2.show();
c3=c1+c2;
cout<<"\n\nSum is: ";
c3.show();
c4=c1*c2;
cout<<"\n\nMultiplication is: ";
c4.show();
}
```

Output :



```
C:\Users\pavit\Downloads\exp6.exe
Pavitra Walia 42414802718
The 1st no. is: 5 + i6

The 2nd no. is: 7 + i8

Sum is: 12 + i14

Multiplication is: -13 + i82
Process returned 0 (0x0) execution time : 3.352 s
Press any key to continue.
_
```


Viva Voce

Q1. List the benefits of the modular development approach.

The advantages of the modular design include:

- Consistency. The fact that the engine allows all applications to run using the same base RAD applications brings consistency to the Service Manager application suite.
- Reduced Development Times.
- Flexibility.

Q2. Give some reasons why computer scientists and professional software developers should study general concepts of language design and evaluation.

Reasons for Studying Concepts of Programming Languages

- Increased capacity to express ideas
- Improved ability to choose an appropriate language
- Increased ability to learn new languages
- A better understanding of implementation issues (i.e., how language constructs are implemented)

Q3. What constitutes a programming environment?

In a general sense, a programming environment combines hardware and software that allows a developer to build applications. Developers typically work in integrated development environments or IDEs. These connect users with all the features necessary to write and test their code correctly.

Q4. Give an example of how aliasing deters reliability.

Aliasing makes it particularly difficult to understand, analyze and optimize programs.

Q5. How do type declaration statements affect the readability of programming language?

Readability is a key element in any programming language and thus, very important. The addition of type declarations helps to improve code readability. This is because it makes it easy to identify and differentiate different variables by data types. A program with well-defined type declarations is easy to understand.

Experiment 7

Aim : Implement simple multi-threaded server to perform all mathematics operation parallel in Java.

Code :

```
//arithtcpclient.java

import java.io.*;
import java.net.*;
public class arithtcpclient
{
    public static void main(String[] args) throws IOException
    {
        System.out.println();
        System.out.println("ARITHMETIC CLIENT");
        System.out.println("*****");
        System.out.println("Enter the host name to connect");
        String str;
        DataInputStream inp=new DataInputStream(System.in);
        str=inp.readLine();
        Socket clientsoc = new Socket(str, 9);
        System.out.println("Enter the inputs");
        PrintWriter out = new PrintWriter(clientsoc.getOutputStream(), true);
        BufferedReader in = new BufferedReader(new
        InputStreamReader(clientsoc.getInputStream()));
        BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
        String userinput;
        try
        {
            while (true)
            {
                do
                {
                    userinput = stdin.readLine();
                    out.println(userinput);
                }while(!userinput.equals("."));
                System.out.println("Sever Says : " + in.readLine());
            }
        }

        catch(Exception e)
        {
            System.exit(0);
        }
    }
}
```

```

//arithtcpserver
import java.io.*;
import java.net.*;
public class arithtcpserver
{
    public static void main(String arg[]) throws Exception
    {
        System.out.println();
        System.out.println("ARITHMETIC SERVER");
        System.out.println("*****");
        System.out.println("Server is ready to accept inputs...");
        ServerSocket serversoc=new ServerSocket(9);
        Socket clientsoc = serversoc.accept();
        PrintWriter out = new PrintWriter(clientsoc.getOutputStream(), true);
        BufferedReader in = new BufferedReader(new
        InputStreamReader(clientsoc.getInputStream()));
        String inputline;
        BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
        try
        {
            while (true)
            {
                String s,op="",st;
                int i=0,c=0;
                int[] a=new int[2];
                while(true)
                {
                    s=in.readLine();
                    if(s.equals("+") || s.equals("-") || s.equals("*") || s.equals("/"))
                        op=s;
                    else if(s.equals("."))
                        break;
                    else
                    {
                        a[i]=Integer.parseInt(s);
                        i++;
                    }
                }
                if(op.equals("+"))
                    c=a[0]+a[1];
                else if(op.equals("-"))
                    c=a[0]-a[1];
                else if(op.equals("*"))
                    c=a[0]*a[1];
                else if(op.equals("/"))
                    c=a[0]/a[1];
                s=Integer.toString(c);
                out.println(s);
            }
        }
        catch(Exception e)
        {
            System.exit(0);
        }
    }
}

```

Output :

SERVER SIDE

```
C:\jdk1.5\bin>javac arithtcpserver.java
C:\jdk1.5\bin>java arithtcpserver
```

ARITHMETIC SERVER

Server is ready to accept inputs...

CLIENT SIDE

```
C:\JDK1.5\bin>javac arithtcpclient.java
C:\JDK1.5\bin>java arithtcpclient
```

ARITHMETIC CLIENT

Enter the host name to connect

p4-221

Enter the inputs

8

+

3

Sever Says : 11

9

-

11

Sever Says : -2

7

*

9

Sever Says : 63

12

/

3

Sever Says : 4

Viva Voce

Q1. Write the uses of constructors and destructors in OOP.

Constructor is used to initializing an object of the class and assign values to data members corresponding to the class. While destructor is used to deallocate the memory of an object of a class. There can be multiple constructors for the same class.

Q2. Explain language evaluation criteria and the characteristics that affect them.

Language Evaluation Criteria

- Criteria. Readability. Writability. Reliability. Cost.
- Affected by. Simplicity. Orthogonality. Data types and structures. Syntax. Support for Abstraction. Type checking. Exception handling. Aliasing. Control structures. Expressivity.

Q3. What Is Backus-naur Form (bnf)?

Backus–Naur form or Backus normal form is a metasyntax notation for context-free grammars, often used to describe the syntax of languages used in computing, such as computer programming languages, document formats, instruction sets, and communication protocols.

Q4. What is the use of pointers?

Pointers are used to store and manage the addresses of dynamically allocated blocks of memory. Such blocks are used to store data objects or arrays of objects.

Q5. What is a Void pointer?

The void pointer is a pointer that is not associated with any data types. It points to some data location in the storage. This means that it points to the address of variables. It is also called the general-purpose pointer.