PRINCIPLES OF PROGRAMMING LANGUAGES LAB
ETCS - 458

Faculty Name: Ms Ruchi Goel

Name: Varun Negi

Roll No: 13314802719

Semester: 8th

Group: 8C7

Maharaja Agrasen Institute of Technology, PSP area, Sector - 22, Rohini, New Delhi - 110085

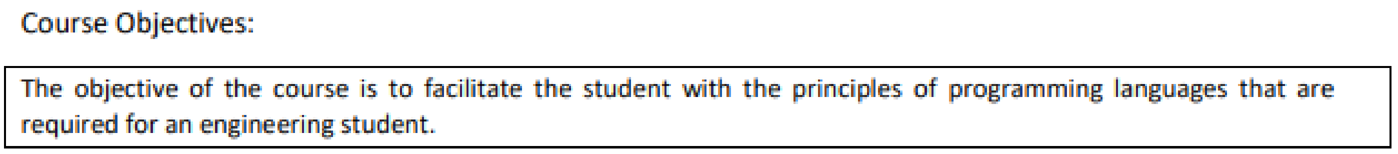
(Affiliated to Guru Gobind Singh Indraprastha University, New Delhi)

MAHARAJA AGRASEN INSTITUTE OF TECHNOLOGY

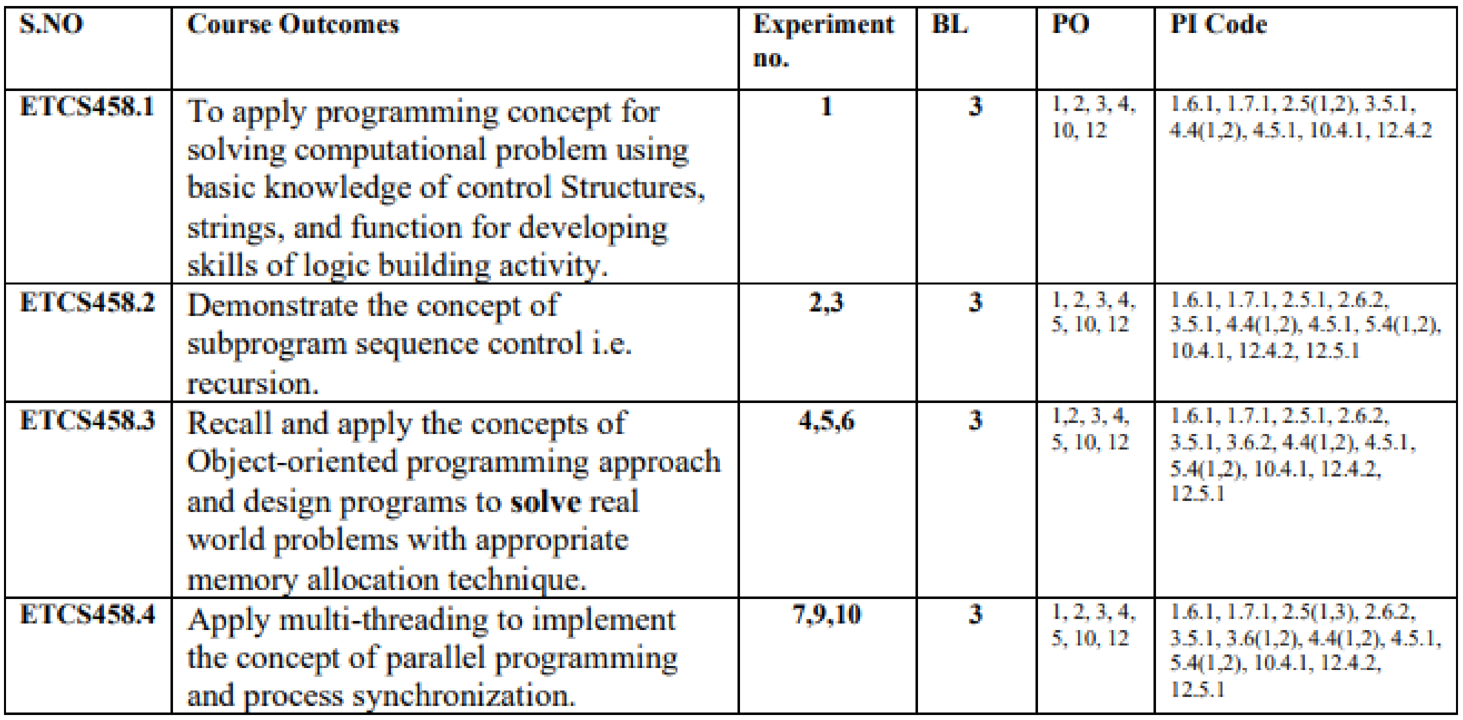
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Outcome Based Learning

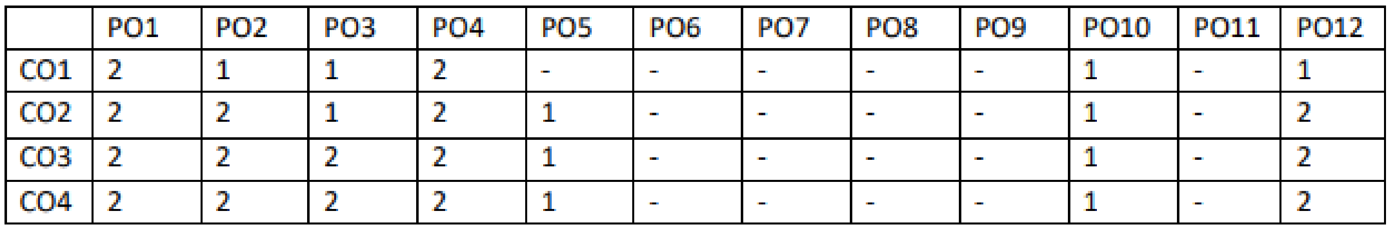
Course Outcomes (Revision)


Course Objectives:

The objective of the course is to facilitate the student with the principles of programming languages that are required for an engineering student.



S.NO	Course Outcomes	Experiment no.	BL	PO	PI Code
ETCS458.1	To apply programming concept for solving computational problem using basic knowledge of control Structures, strings, and function for developing skills of logic building activity.	1	3	1, 2, 3, 4, 10, 12	1.6.1, 1.7.1, 2.5(1,2), 3.5.1, 4.4(1,2), 4.5.1, 10.4.1, 12.4.2
ETCS458.2	Demonstrate the concept of subprogram sequence control i.e. recursion.	2,3	3	1, 2, 3, 4, 5, 10, 12	1.6.1, 1.7.1, 2.5.1, 2.6.2, 3.5.1, 4.4(1,2), 4.5.1, 5.4(1,2), 10.4.1, 12.4.2, 12.5.1
ETCS458.3	Recall and apply the concepts of Object-oriented programming approach and design programs to **solve** real world problems with appropriate memory allocation technique.	4,5,6	3	1,2, 3, 4, 5, 10, 12	1.6.1, 1.7.1, 2.5.1, 2.6.2, 3.5.1, 3.6.2, 4.4(1,2), 4.5.1, 5.4(1,2), 10.4.1, 12.4.2, 12.5.1
ETCS458.4	Apply multi-threading to implement the concept of parallel programming and process synchronization.	7,9,10	3	1, 2, 3, 4, 5, 10, 12	1.6.1, 1.7.1, 2.5(1,3), 2.6.2, 3.5.1, 3.6(1,2), 4.4(1,2), 4.5.1, 5.4(1,2), 10.4.1, 12.4.2, 12.5.1



	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	1	1	2	-	-	-	-	-	1	-	1
CO2	2	2	1	2	1	-	-	-	-	1	-	2
CO3	2	2	2	2	1	-	-	-	-	1	-	2
CO4	2	2	2	2	1	-	-	-	-	1	-	2

Varun Negi 13314802719

Department of Computer Science and Engineering

Rubrics for Lab Assessment

Rubrics		0	1	2	3
		Missing	**Inadequate**	**Needs Improvement**	**Adequate**
R1	Is able to identify the problem to be solved and define the objectives of the experiment.	No mention is made of the problem to be solved.	An attempt is made to identify the problem to be solved but it is described in a confusing manner, objectives are not relevant, objectives contain technical/ conceptual errors or objectives are not measurable.	The problem to be solved is described but there are minor omissions or vague details. Objectives are conceptually correct and measurable but may be incomplete in scope or have linguistic errors.	The problem to be solved is clearly stated. Objectives are complete, specific, concise, and measurable. They are written using correct technical terminology and are free from linguistic errors.
R2	Is able to design a reliable experiment that solves the problem.	The experiment does not solve the problem.	The experiment attempts to solve the problem but due to the nature of the design the data will not lead to a reliable solution.	The experiment attempts to solve the problem but due to the nature of the design there is a moderate chance the data will not lead to a reliable solution.	The experiment solves the problem and has a high likelihood of producing data that will lead to a reliable solution.
R3	Is able to communicate the details of an experimental procedure clearly and completely.	Diagrams are missing and/or experimental procedure is missing or extremely vague.	Diagrams are present but unclear and/or experimental procedure is present but important details are missing.	Diagrams and/or experimental procedure are present but with minor omissions or vague details.	Diagrams and/or experimental procedure are clear and complete.
R4	Is able to record and represent data in a meaningful way.	Data are either absent or incomprehensible.	Some important data are absent or incomprehensible.	All important data are present, but recorded in a way that requires some effort to comprehend.	All important data are present, organized and recorded clearly.
R5	Is able to make a judgment about the results of the experiment.	No discussion is presented about the results of the experiment .	A judgment is made about the results, but it is not reasonable or coherent.	An acceptable judgment is made about the result, but the reasoning is flawed or incomplete.	An acceptable judgment is made about the result, with clear reasoning. The effects of assumptions and experimental uncertainties are considered.

PRINCIPLES OF PROGRAMMING LANGUAGES LAB PRACTICAL RECORD

PAPER CODE: **ETCS – 458**

Name: **VARUN NEGI**

University Roll No.: **13314802719**

Group: **8C7**

Branch: **CSE**

S.No	**EXPERIMENT NAME**	**DATE**	**MARKS**					**Total Marks**	**Signature**
			R1	**R2**	**R3**	**R4**	**R5**		
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, strrev)								
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.								
8.	Write a program in c++ to prepare a list of 50 questions and their answers.								
9.	Write a program to display 10 questions at random out of exp.8-50 questions (do not display the answer of these questions to the user now).								
10.	Implement producer-consumer problem using threads.								

EXPERIMENT – 1

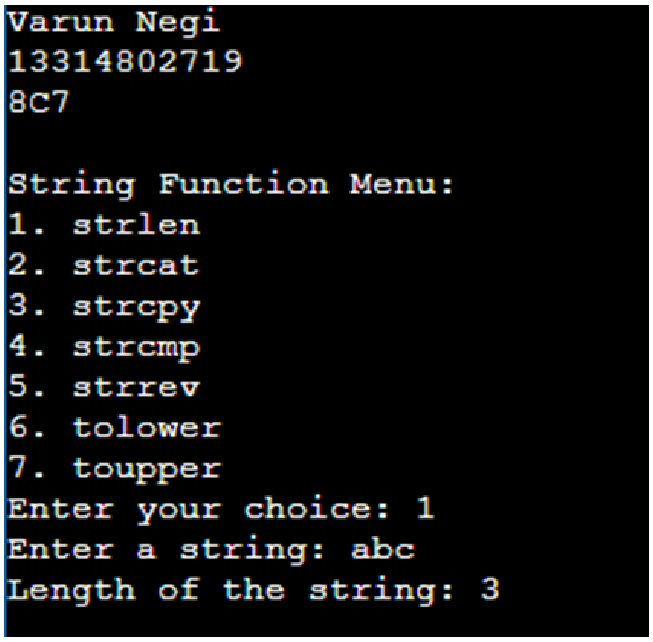
AIM: Implement all major functions of string.h in single C program using switch case to select specificfunction from user choice (like strlen, strcat, strcpy, strcmp, strrev).

SOURCE CODE:

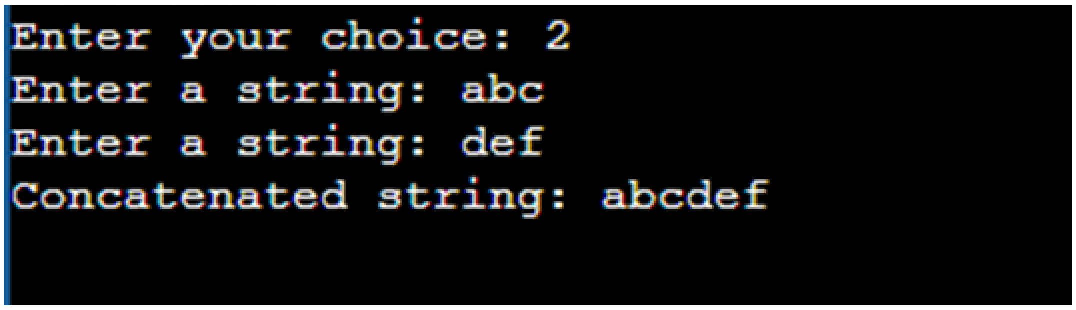
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void getString(char* str) {
   printf("Enter a string: ");
   fgets(str, 100, stdin);
   // Remove the trailing newline character
   str[strcspn(str, "\n")] = '\0';
}

int main() {
   printf("Varun Negi : 13314802719 : 8C7")
   int choice;
   char str1[100], str2[100], result[100];

   printf("String Function Menu:\n");
   printf("1. strlen\n");
   printf("2. strcat\n");
   printf("3. strcpy\n");
   printf("4. strcmp\n");
   printf("5. strrev\n");
   printf("6. tolower\n");
   printf("7. toupper\n");
   printf("Enter your choice: ");
   scanf("%d", &choice);
   getchar();  // Consume the newline character

   switch (choice) {
      case 1: {
         getString(str1);
```

```
            printf("Length of the string: %zu\n", strlen(str1));
            break;
        }
        case 2: {
            getString(str1);
            getString(str2);
            strcpy(result, str1);
            strcat(result, str2);
            printf("Concatenated string: %s\n", result);
            break;
        }
        case 3: {
            getString(str1);
            strcpy(str2, str1);
            printf("Copied string: %s\n", str2);
            break;
        }
        case 4: {
            getString(str1);
            getString(str2);
            int cmp = strcmp(str1, str2);
            if (cmp == 0)
                printf("Strings are equal\n");
            else if (cmp < 0)
                printf("String 1 is less than string 2\n");
            else
                printf("String 1 is greater than string 2\n");
            break;
        }
        case 5: {
            getString(str1);
            int len = strlen(str1);
            for (int i = len - 1; i >= 0; i--) {
                printf("%c", str1[i]);
            }
            printf("\n");
            break;
        }
        case 6: {
            getString(str1);
            for(int i=0;i<strlen(str1);i++){
                str1[i]=tolower(str1[i]);
            }
            printf("String in Lowercase : %s", str1);
            break;
        }
        case 7: {
            getString(str1);
            for(int i=0;i<strlen(str1);i++){
                str1[i]=toupper(str1[i]);
            }
            printf("String in Uppercase : %s", str1);
            break;
        }
        default:
            printf("Invalid choice!\n");
            break;
    }

    return 0;
}
```

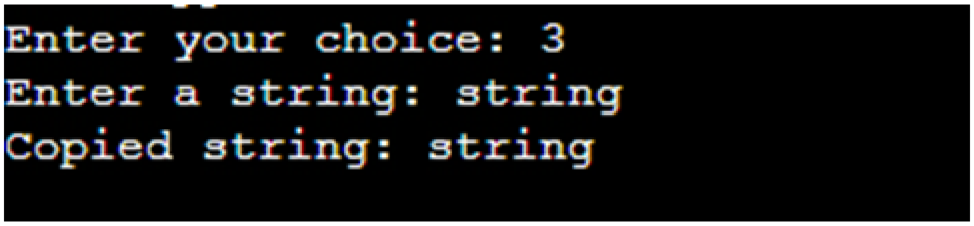
<u>OUTPUT</u> :



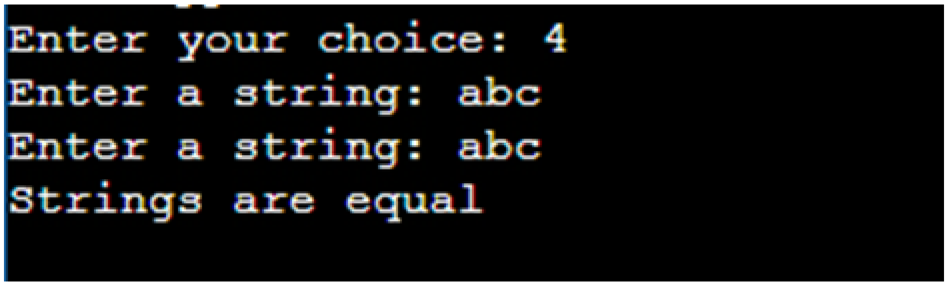
```
Varun Negi
13314802719
8C7

String Function Menu:
1. strlen
2. strcat
3. strcpy
4. strcmp
5. strrev
6. tolower
7. toupper
Enter your choice: 1
Enter a string: abc
Length of the string: 3
```



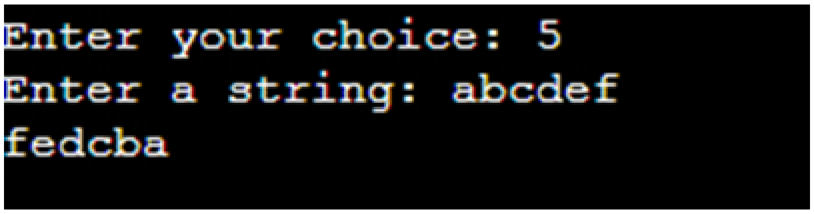
```
Enter your choice: 2
Enter a string: abc
Enter a string: def
Concatenated string: abcdef
```



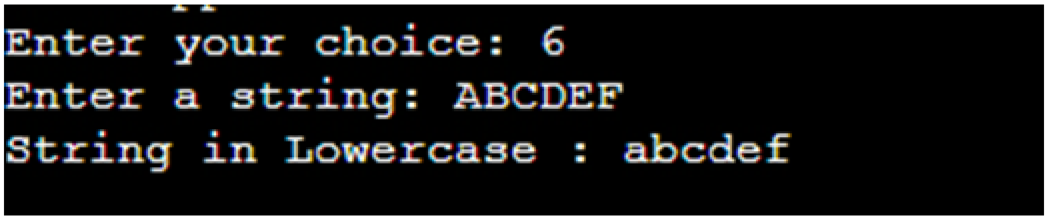
```
Enter your choice: 3
Enter a string: string
Copied string: string
```



```
Enter your choice: 4
Enter a string: abc
Enter a string: abc
Strings are equal
```

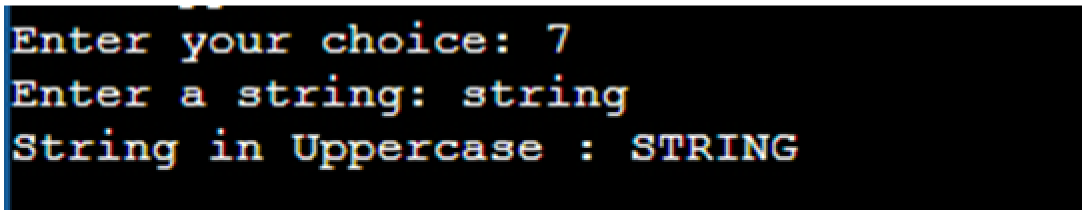


```
Enter your choice: 5
Enter a string: abcdef
fedcba
```



```
Enter your choice: 6
Enter a string: ABCDEF
String in Lowercase : abcdef
```

\



```
Enter your choice: 7
Enter a string: string
String in Uppercase : STRING
```

VIVA – VOCE

Q1. What is the use of string.h header while and where is this file stored.
Ans. The string.h header file in C (or <cstring> in C++) provides various string manipulation functions and character array operations. It is part of the C standard library and contains function declarations for common string operations like copying, concatenating, comparing, and searching.
The storage of the string.h header file, it is part of the C standard library or C++ Standard Library, depending on the language you are using. The header file is typically provided by the compiler or the standard library implementation you are using.

Q2. How can we create a header file?
Ans. To create a header file in C or C++, you can follow these steps:

1. Create a new file with a .h extension, such as myfile.h. This will be your header file.
2. Add the necessary include guards to prevent multiple inclusion of the header file. Include guards ensure that the contents of the header file are included only once during compilation.
3. Write the declarations for functions, structures, or variables that you want to expose to other source files.
4. Save the header file.

Q3. Write the function to find the length of a string.
Ans.
```
int len() {
   char s[] = "Programming is fun";
   int i;
   for (i = 0; s[i] != '\0'; ++i);
   printf("Length of the string: %d", i);
   return 0;
}
```

Q4. Write the function to concatenate two strings.
Ans.
```
char* concatenateStrings(const char* str1, const char* str2) {
   size_t len1 = strlen(str1);
   size_t len2 = strlen(str2);

   char* result = (char*)malloc((len1 + len2 + 1) * sizeof(char));

   strcpy(result, str1);
   strcat(result, str2);
   return result;
}
```

Q5. Explain the use of header file
Ans. Header files are used in C++ so that you don't have to write the code for every single thing. It helps to reduce the complexity and number of lines of code. It also gives you the benefit of reusing the functions that are declared in header files to different

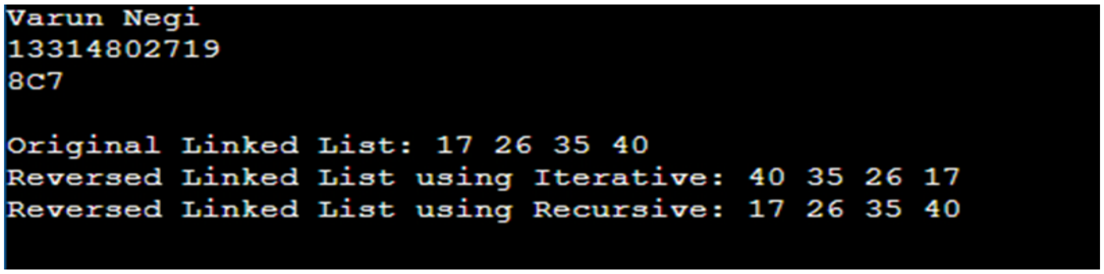
EXPERIMENT – 2

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

SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>

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

// Function to insert a new node at the beginning of the linked list
void insert(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
}

// Function to print the linked list
void display(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Function to reverse the linked list iteratively
struct Node* reverseIterative(struct Node* head) {
    struct Node* prev = NULL;
    struct Node* current = head;
    struct Node* next = NULL;

    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

// Function to reverse the linked list recursively
struct Node* reverseRecursive(struct Node* head) {
    if (head == NULL || head->next == NULL) {
        return head;
    }

    struct Node* rest = reverseRecursive(head->next);
    head->next->next = head;
    head->next = NULL;

    return rest;
}

int main() {
    printf("Varun Negi : 13314802719 : 8C7\n\n");
    struct Node* head = NULL;

    insert(&head, 4);
    insert(&head, 3);
    insert(&head, 2);
    insert(&head, 1);

    printf("Original Linked List: ");
    display(head);

    struct Node* reversedIterative = reverseIterative(head);
    printf("Reversed Linked List (Iterative): ");
    display(reversedIterative);

    struct Node* reversedRecursive = reverseRecursive(reversedIterative);
    printf("Reversed Linked List (Recursive): ");
    display(reversedRecursive);

    return 0;
}
```

<u>OUTPUT</u>:



```
Varun Negi
13314802719
8C7

Original Linked List: 17 26 35 40
Reversed Linked List using Iterative: 40 35 26 17
Reversed Linked List using Recursive: 17 26 35 40
```

VIVA VOCE QUESTIONS

Q1. What is the difference between iterative and recursive function call?
Ans. Recursion occurs when a statement in a function calls itself repeatedly. The iteration occurs when a loop repeatedly executes until the controlling condition becomes false

Q2. What are formal parameters in functions?
Ans. A formal parameter is a variable that you specify when you determine the subroutine or function. These parameters define the list of possible variables, their positions, and their data types.

Q3. How is the structure node declared?
Ans.
```
struct structName{

     // structure definition
    Data_type1 member_name1;
    Data_type2 member_name2;
    Data_type2 member_name2;

  } struct_var1, struct_var2;
```

Q4. Define a link list.
Ans. A linked list consists of nodes where each node contains a data field and a reference (link) to the next node in the list.

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

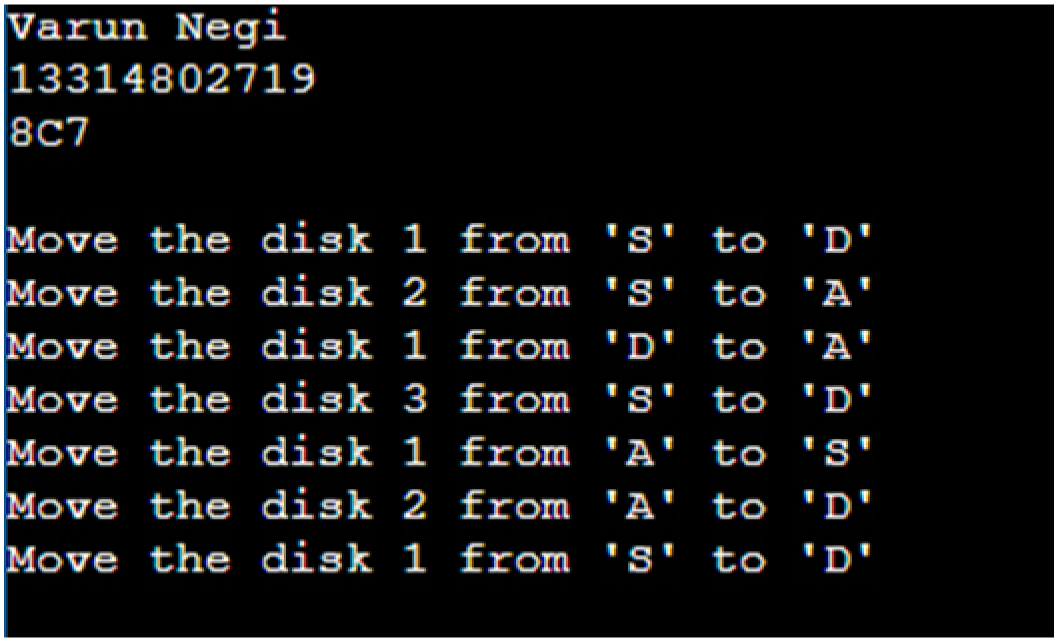
Source Code:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <limits.h>

struct Stack  { // A structure to represent a stack
        unsigned capacity;
        int top;
        int *array;
};

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;
}

int isFull(struct Stack* stack){
        return (stack->top == stack->capacity - 1);
}

int isEmpty(struct Stack* stack){
        return (stack->top == -1);
}

void push(struct Stack *stack, int item){
        if (isFull(stack))
                return;
        stack -> array[++stack -> top] = item;
}

int pop(struct Stack* stack){
        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);
	}
}

int main(){
	// 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:



```
Varun Negi
13314802719
8C7

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'
```

VIVA VOCE QUESTIONS

Q1. What is the tower of Hanoi problem?
Ans. The objective of this problem is to move the stack of disks from the initial rod to another rod, following these rules: A disk cannot be placed on top of a smaller disk. No disk can be placed on top of the smaller disk.

Q2. What are the various ways is stack created?
Ans. A stack can be implemented by means of Array, Structure, Pointer, and Linked List.

Q3. List the models of computation of language
Ans. There are six basic computational models such as Turing, von Neumann, dataflow, applicative, object-based, predicate logic-based, etc. These models are known as basic models because they can be declared using a basic set of abstractions.

Q4. What are objectives of principle of programming language?
Ans. The Objectives of Principle of programming Language are :
- Understand the concepts and terms used to describe languages that support the imperative, functional, object-oriented, and logic programming paradigms.
- Solve problems using the functional paradigm.
- Solve problems using the object-oriented paradigm.
- Solve problems using the logic programming paradigm.
- Critically evaluate what paradigm and language are best suited for a new problem.

Q5. What are the Paradigms of Programming?
Ans. Programming paradigms are a way to classify programming languages based on their features. Languages can be classified into multiple paradigms.
Common programming paradigms include:
- Imperative
- Procedural
- Object oriented
- Declarative
- Functional

EXPERIMENT – 4

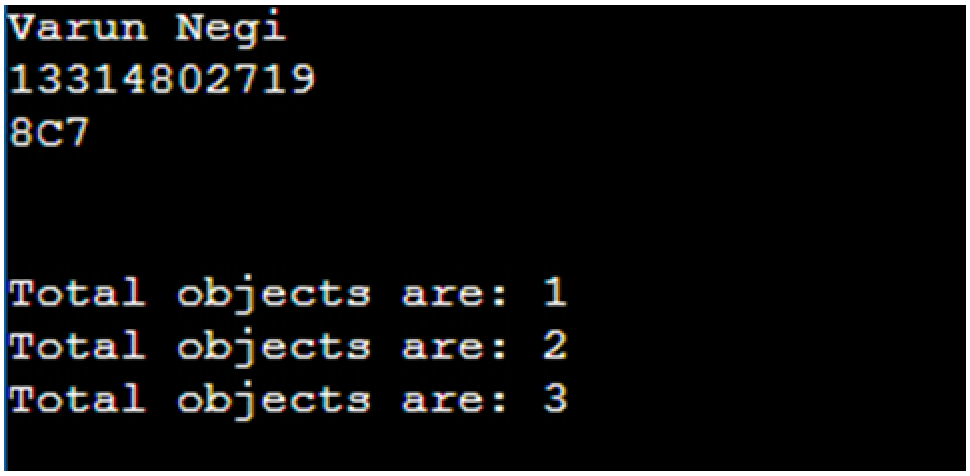
AIM: WAP in C++ to count the no. of object of a class with the help of static data member, function and constructor

Source Code:

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

class Counter{
   private:
   static int count;

   public:
   Counter(){  //default constructor
      count++;
   }

   static void Print(){
      cout<<"\nTotal objects are: "<<count;
   }
};

//count initialization with 0
int Counter :: count = 0;

int main(){
   printf("Varun Negi : 13314802719 : 8C7\n");
   Counter OB1;
   OB1.Print();
```

```
    Counter OB2;
    OB2.Print();

    Counter OB3;
    OB3.Print();

    return 0;
}
```

OUTPUT:



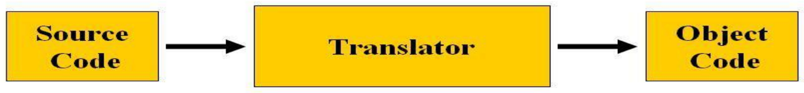
```
Varun Negi
13314802719
8C7


Total objects are: 1
Total objects are: 2
Total objects are: 3
```

VIVA VOCE QUESTIONS

Q1. List various type of languages.
Ans. Some different types of languages are
- Assembly Language
- Machine Language
- Compiled Language
- Scripting Language
- CLI Language
- Hardware Description Language

Q2. What are the issues for languages?
Ans. Some issues for programming languages are:
- Trade-off between efficiency of execution and ease of writing program.
- Type checking
- Many programming languages suffer from bad documentation, missing libraries, platform specific etc.
- Languages need to be portable and interoperable.

Q3. What is translation?
Ans. A computer system can only understand machine code. A program written for example in a high level language such as Java cannot be run directly. To execute a computer program written in any programming language, it must first be translated.
The source code which is written by the programmer needs to be translated. When translated, the source code becomes object code which is understandable by the computer system.



Q4. What are different types of translation and their roles?
Ans. **Assembler**: Assemblers convert assembly language mnemonics into machine code.
Interpreters: Interpreters convert each instruction of the source code into the object code as the program is being run. This gives a better interactive environment but is slower.
Compilers: A compiler converts the entire source code into machine code so that it can be run on the machine without further translation. However, error correction is tedious.

Q5. What is trade's off of translation
Ans. The most well-known tradeoff is between the direct and efficient control of computer hardware vs. generic, relatively simple way of expressing algorithms. This tradeoff can be also known as low level vs. high-level programming, and the underlying phenomenon is called the abstraction penalty, a well-established observation that optimized program code is often complicated and hard to maintain, while a concise, elegant program that solves the same task is usually less optimal.

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.

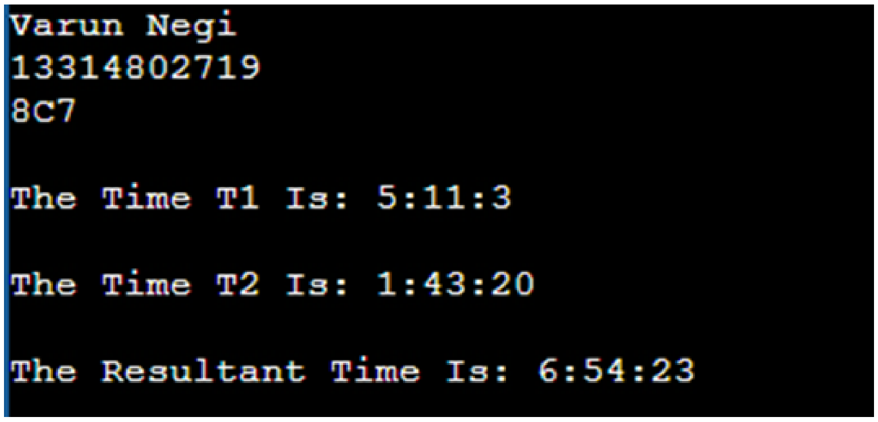
Source Code:

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(){
    cout<<"Varun Negi : 13314802719 : 8C7\n\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();
}
```

<u>**Output**</u>:



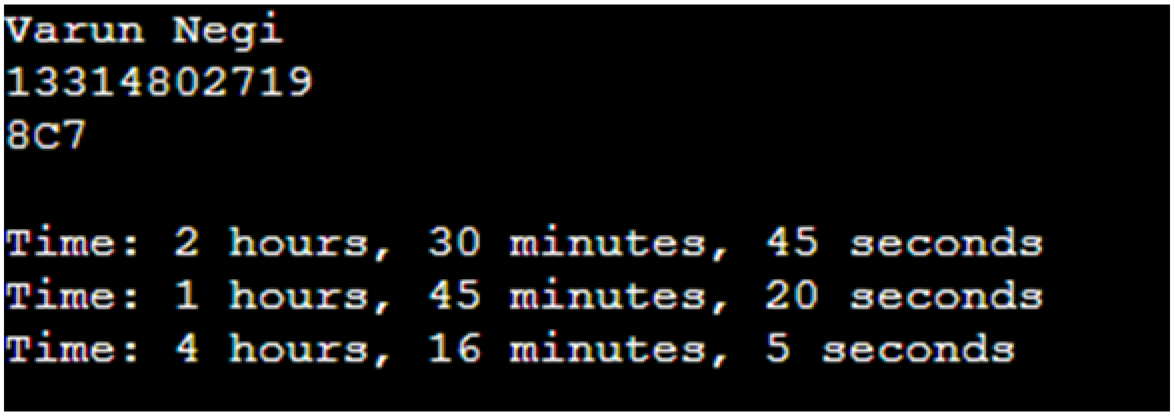
```
Varun Negi
13314802719
8C7

The Time T1 Is: 5:11:3

The Time T2 Is: 1:43:20

The Resultant Time Is: 6:54:23
```

Java :

```
class Time {
    private int hh, mm, ss;

    public Time(int hours, int minutes, int seconds) {
        hh = hours;
        mm = minutes;
        ss = seconds;
    }

    public Time addTime(Time t) {
        int totalSeconds = ss + t.ss;
        int carryMinutes = totalSeconds / 60;
        int remainingSeconds = totalSeconds % 60;

        int totalMinutes = mm + t.mm + carryMinutes;
        int carryHours = totalMinutes / 60;
        int remainingMinutes = totalMinutes % 60;

        int totalHours = hh + t.hh + carryHours;

        return new Time(totalHours, remainingMinutes, remainingSeconds);
    }

    public void displayTime() {
        System.out.println("Time: " + hh + " hours, " + mm + " minutes, " + ss + " seconds");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Time t1 = new Time(2, 30, 45);
        Time t2 = new Time(1, 45, 20);

        Time t3 = t1.addTime(t2);

        t1.displayTime();
        t2.displayTime();
        t3.displayTime();
    }
}
```

Output:



```
Varun Negi
13314802719
8C7

Time: 2 hours, 30 minutes, 45 seconds
Time: 1 hours, 45 minutes, 20 seconds
Time: 4 hours, 16 minutes, 5 seconds
```

VIVA VOCE QUESTIONS

Q1. Write any four important uses of programming languages.

- A programming language provides a structured mechanism for defining pieces of data, and the operations or transformations that may be carried out automatically on that data.
- Used in web development.
- Used to create Operating Systems.
- Used to maintain data.

Q2. The levels of acceptance of any language depend on the language description. Comment.
Ans. The acceptance of any language depends on the features and suitability according to the need. Languages like Java and Python are widely used to solve problems and web applications due to their ease of use and less-complexity.

Q3. Write the differences between lexical syntax and concrete syntax of the language.
Ans. Lexical Syntax: All the basic symbols of the language

Concrete Syntax: The rules of writing expressions, statements and programs.

Q4. List the design principles of imperative languages.
Ans. 1. It features close relation to machine architecture.
2. It is based on Von-Neumann architecture.

Q5. Write the differences between array and enumerated data types in imperative languages?
Ans. Array is a variable that can contain multiple elements with index starting from 0 whereas enum is a user defined datatype that contains a list of members for which an integer constant is assigned starting from 0. In case of enum the numbers starting from 0 are not indexes whereas in case of an array they are indexes. Also in case of enum you can assign your own constant values to the members that may or may not start from 0 and may or may not be in a sequence.

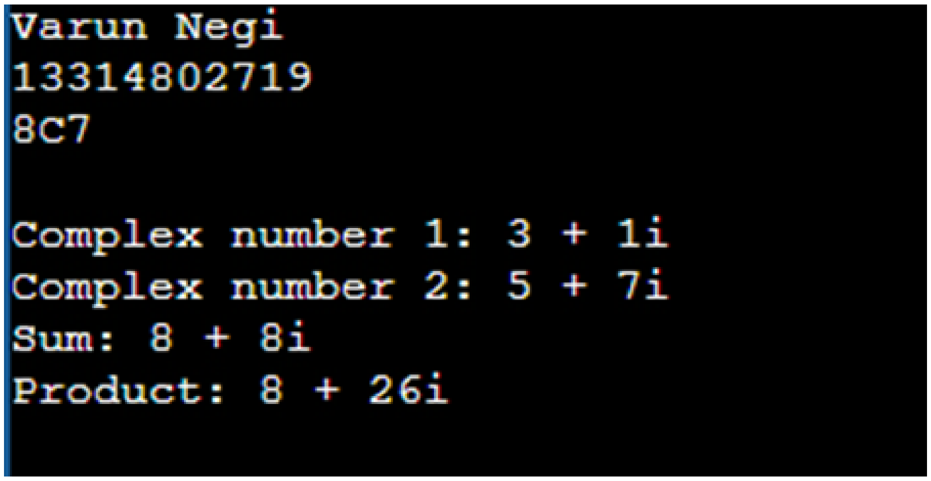
EXPERIMENT – 6

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

Source Code:

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

class Complex {
private:
    double real;
    double imag;

public:
    Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}

    // Overloading + operator using member function
    Complex operator+(const Complex& other) {
        Complex sum;
        sum.real = real + other.real;
        sum.imag = imag + other.imag;
        return sum;
    }

    double getReal() {
        return real;
    }

    double getImag() {
        return imag;
    }

    friend Complex operator*(const Complex& c1, const Complex& c2); // Forward declaration
of friend function
};
// Overloading * operator using friend function
Complex operator*(const Complex& c1, const Complex& c2) {
    Complex product;
    product.real = c1.real * c2.real - c1.imag * c2.imag;
    product.imag = c1.real * c2.imag + c1.imag * c2.real;
    return product;
}

int main() {
    cout<<"Varun Negi : 13314802719 : 8C7\n\n";
    Complex c1(2.0, 3.0);
    Complex c2(4.0, 5.0);

    Complex sum = c1 + c2;
    Complex product = c1 * c2;

    cout << "Complex number 1: " << c1.getReal() << " + " << c1.getImag() << "i" << std::endl;
    cout << "Complex number 2: " << c2.getReal() << " + " << c2.getImag() << "i" << std::endl;
    cout << "Sum: " << sum.getReal() << " + " << sum.getImag() << "i" << std::endl;
    cout << "Product: " << product.getReal() << " + " << product.getImag() << "i" << std::endl;

    return 0;
}
```

OUTPUT:



```
Varun Negi
13314802719
8C7

Complex number 1: 3 + 1i
Complex number 2: 5 + 7i
Sum: 8 + 8i
Product: 8 + 26i
```

Varun Negi 13314802719

VIVA VOCE QUESTIONS

Q1. Distinguish between dangling pointers and memory leakage.
Ans. A dangling pointer points to memory that has already been freed. The storage is no longer allocated. Trying to access it might cause a Segmentation fault.
A memory leak is memory which hasn't been freed, there is no way to access (or free it) now, as there are no ways to get to it anymore. (E.g. a pointer which was the only reference to a memory location dynamically allocated (and not freed) which points somewhere else now.)

Q2. List the benefits of modular development approach
Ans. Ease of use- This approach allows simplicity
Reusability- Allows user to reuse functions
Ease of maintenance- Helps in less collision at the time of working on modules

Q3. Give some reasons why computer scientists and professional software developers should study general concepts of language design and evaluation.
Ans. 1. It helps to increase ability to express ideas.
2. Improves background for choosing appropriate languages.
3. Understand significance of implementation.

Q4. What constitutes a programming environment?
Ans. Text Editor, Compiler, Interpreter, Machine (Hardware)

Q5. Give an example of how aliasing deters reliability.
Ans. Aliasing affects performance by preventing the compiler from doing certain optimizations.
For example:

```
void foo(int *array,int *size,int *value) {
        for(int i=0;i<*size;++i) {
                array[i] = 2 * *value;
        }
}
```

Looking at this code we might expect that the compiler could load *value once outside the loop and then set every element in the array to that value very quickly. But this isn't the case due to aliasing. Because *value could be an alias for an element of the array it could change on any given iteration. Therefore the code has to load the value every single iteration, resulting in a potentially large slowdown.

EXPERIMENT – 7

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

Source Code:

Server.java:

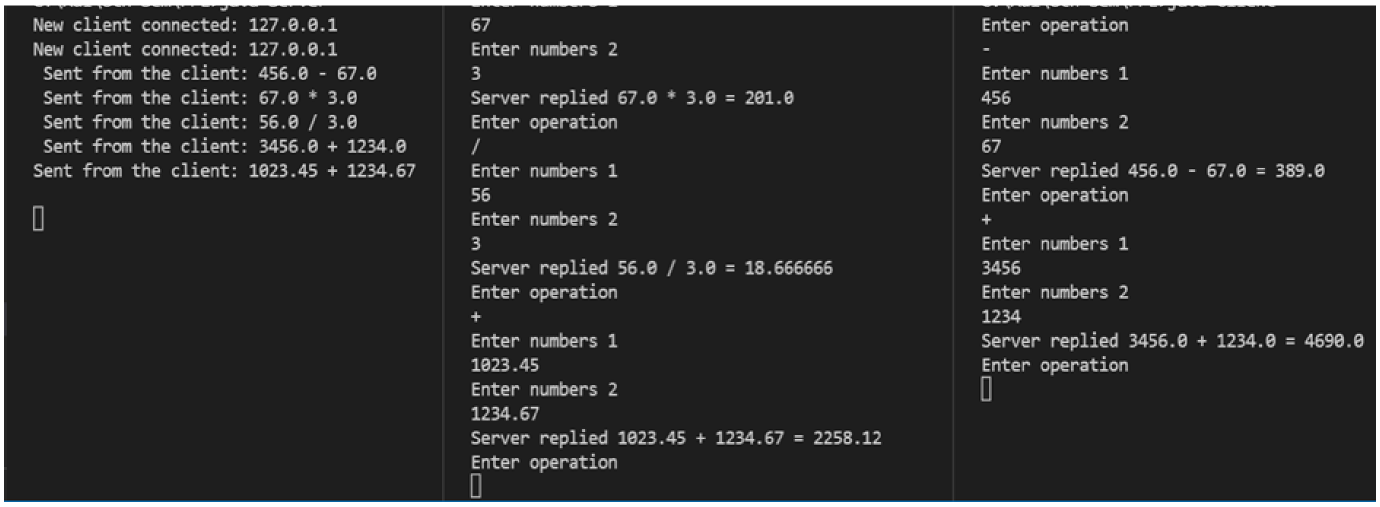
```
import java.io.*;
import java.net.*;
class Server {
    public static void main(String[] args) {
        ServerSocket server = null;
        try {
            server = new ServerSocket(1234);
            server.setReuseAddress(true);
            while (true) {
                Socket client = server.accept();
                System.out.println("New client connected"
                                        + client.getInetAddress()
                                                .getHostAddress());
                ClientHandler clientSock
                    = new ClientHandler(client);
                new Thread(clientSock).start();
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            if (server != null) {
                try {
                        server.close();
                }
                catch (IOException e) {
                        e.printStackTrace();
                }
            }
        }
}
private static class ClientHandler implements Runnable {
        private final Socket clientSocket;
        public ClientHandler(Socket socket) {
                this.clientSocket = socket;
        }

        public void run(){
                PrintWriter out = null;
                BufferedReader in = null;
                try {
                        out = new PrintWriter(clientSocket.getOutputStream(), true);
                        in = new BufferedReader(new InputStreamReader(
                                        clientSocket.getInputStream()));

                        String line;
                        while ((line = in.readLine()) != null) {
                                String del = "#";
                                String[] temp = line.split(del);
                                float x = Float.parseFloat(temp[1]);
                                float y = Float.parseFloat(temp[2]);
                                char operation = temp[0].charAt(0);
                                float result = 0;
                if(operation == '+'){
                    result = x + y;
                }else if(operation == '-'){
                    result = x- y;
                }else if(operation == '*'){
                    result = x * y;
                }else if(operation == '/'){
                    result = x / y;
                }else{

                }
                String req = "" + x + " " + operation +" " + y;
                System.out.printf(" Sent from the client: %s\n",req);
                String res = "" + x +" "+ operation +" "+ y + " = " + result;
                out.println(res);
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            try {
                if (out != null) {
                    out.close();
                }
                if (in != null) {
                    in.close();
                    clientSocket.close();
                }
            }
            catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Client.java:

```
import java.io.*;
import java.net.*;
import java.util.*;
class Client {
    public static void main(String[] args) {
        try (Socket socket = new Socket("localhost", 1234)) {
            PrintWriter out = new PrintWriter(
            socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new InputStreamReader(
                    socket.getInputStream()));
            Scanner sc = new Scanner(System.in);
            String line = null;

            while (!"exit".equalsIgnoreCase(line)) {

     System.out.println("Enter operation");
     char operation = sc.next().charAt(0);
     System.out.println("Enter numbers 1");
     float x = sc.nextFloat();
     System.out.println("Enter numbers 2");
     float y = sc.nextFloat();
     String req = operation + "#" + x + "#" + y;
                out.println(req);
```

```
                    out.flush();
                    System.out.println("Server replied " + in.readLine());
                }
                sc.close();
            }
            catch (IOException e) {
                e.printStackTrace();
            }
    }
}
```

OUTPUT:



```
New client connected: 127.0.0.1
New client connected: 127.0.0.1
 Sent from the client: 456.0 - 67.0
 Sent from the client: 67.0 * 3.0
 Sent from the client: 56.0 / 3.0
 Sent from the client: 3456.0 + 1234.0
Sent from the client: 1023.45 + 1234.67
```

```
67
Enter numbers 2
3
Server replied 67.0 * 3.0 = 201.0
Enter operation
/
Enter numbers 1
56
Enter numbers 2
3
Server replied 56.0 / 3.0 = 18.666666
Enter operation
+
Enter numbers 1
1023.45
Enter numbers 2
1234.67
Server replied 1023.45 + 1234.67 = 2258.12
Enter operation
```

```
Enter operation
-
Enter numbers 1
456
Enter numbers 2
67
Server replied 456.0 - 67.0 = 389.0
Enter operation
+
Enter numbers 1
3456
Enter numbers 2
1234
Server replied 3456.0 + 1234.0 = 4690.0
Enter operation
```

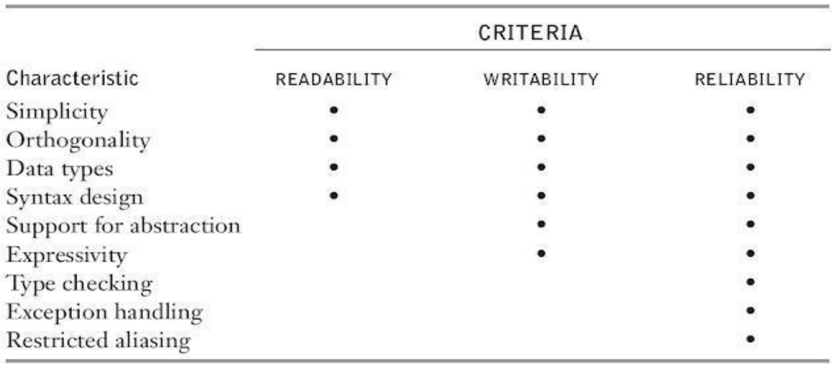
VIVA VOCE QUESTIONS

Q1. How do type declaration statements effect the readability of programming language?
Ans. The use of type declaration statements for simple scalar variables may have very little effect on the readability of programs. If a language has no type declarations at all, it may be an aid to readability, because regardless of where a variable is seen in the program text, its type can be determined without looking elsewhere. Unfortunately, most languages that allow implicitly declared variables also include explicit declarations. In a program in such a language, the declaration of a variable must be found before the reader can determine the type of that variable when it is used in the program.

Q2. Write the uses of constructor and destructors in OOP.
Ans. A constructor is needed so that when the class is instantiated all the variables and functions exist in memory. This allows you to use the class by name along with all the functions and variables.
The destructor is used to clean up memory and release the memory that was used for the class. Some believe that a destructor isn't needed because garbage collection will take care of it. It can, but it does a bad job of it. So USE the destructor it will help in the end.

Q3. Explain language evaluation criteria and the characteristics that affect them.
Ans. The most prominent language evaluation criteria are:

- Readability
- Writability
- Reliability



	CRITERIA		
Characteristic	READABILITY	WRITABILITY	RELIABILITY
Simplicity	•	•	•
Orthogonality	•	•	•
Data types	•	•	•
Syntax design	•	•	•
Support for abstraction		•	•
Expressivity		•	•
Type checking			•
Exception handling			•
Restricted aliasing			•

Q4. What Is Backus-Naur Form (BNF)?
Ans. In computer science, Backus–Naur form or Backus normal form (BNF) is a notation technique 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. They are applied wherever exact descriptions of languages are needed: for instance, in official language specifications, in manuals, and in textbooks on programming language theory.

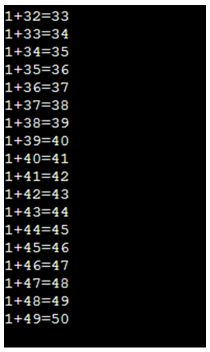
EXPERIMENT – 8

AIM: Write a program in c++ to prepare a list of 50 questions and their answers.

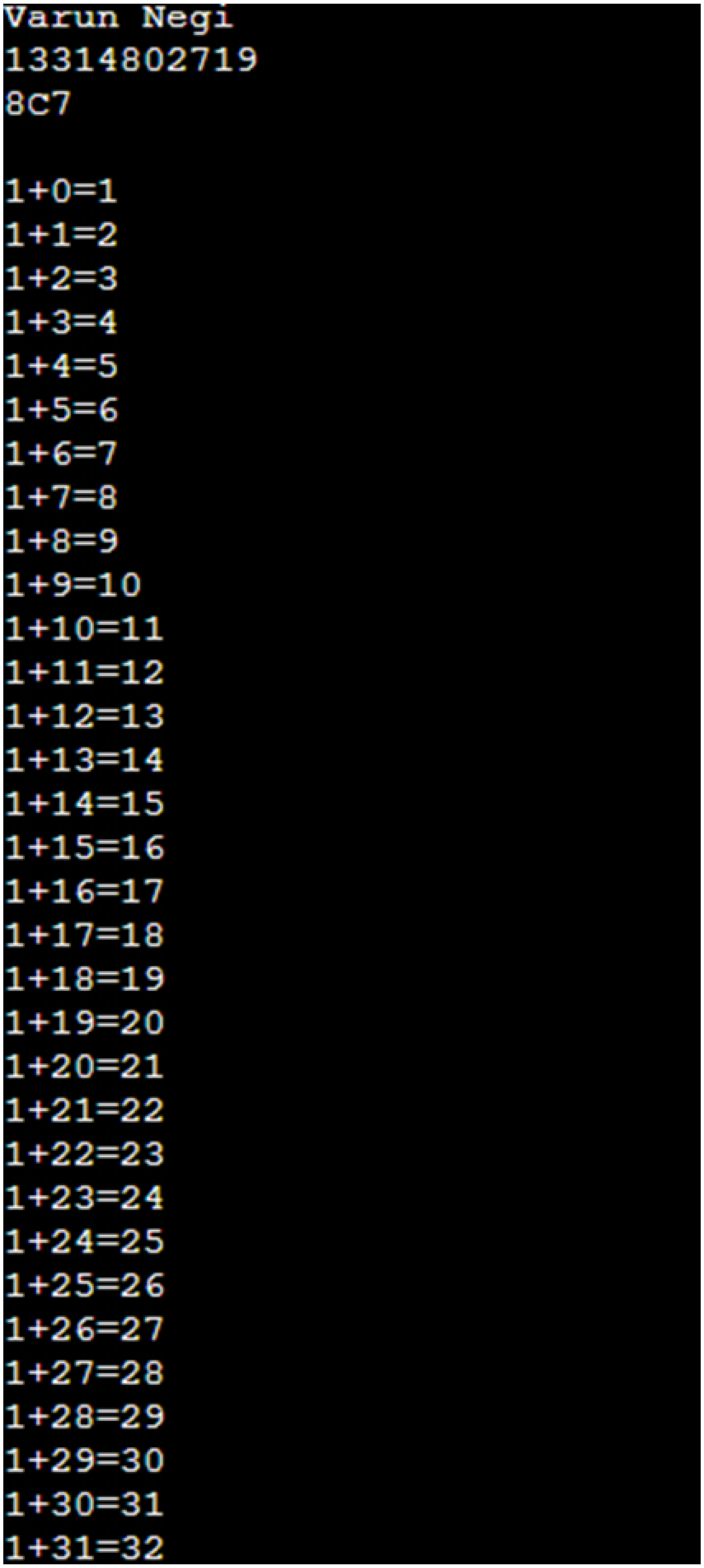
Source Code:

```
#include <bits/stdc++.h>
using namespace std;

int main(){
   cout<<"Varun Negi : 13314802719 : 8C7\n\n";
   string ques[50],ans[50];

   for(int i=0,j=1;i<50;i++,j++){
      stringstream s1,s2;

      s1 << i;
      s2 << j;

      ques[i]="1+";
      ques[i]+=s1.str();

      ans[i]=s2.str();
   }

   for(int i=0;i<50;i++){
      cout<<ques[i]<<"="<<ans[i]<<endl;
   }

   return 0;

}
```

OUTPUT:



```
1+32=33
1+33=34
1+34=35
1+35=36
1+36=37
1+37=38
1+38=39
1+39=40
1+40=41
1+41=42
1+42=43
1+43=44
1+44=45
1+45=46
1+46=47
1+47=48
1+48=49
1+49=50
```



```
Varun Negi
13314802719
8C7

1+0=1
1+1=2
1+2=3
1+3=4
1+4=5
1+5=6
1+6=7
1+7=8
1+8=9
1+9=10
1+10=11
1+11=12
1+12=13
1+13=14
1+14=15
1+15=16
1+16=17
1+17=18
1+18=19
1+19=20
1+20=21
1+21=22
1+22=23
1+23=24
1+24=25
1+25=26
1+26=27
1+27=28
1+28=29
1+29=30
1+30=31
1+31=32
```

VIVA VOCE QUESTIONS

Q1. How is memory allocated dynamically?
Ans. In C, dynamic memory is allocated from the heap using some standard library functions. The two key dynamic memory functions are malloc() and free(). The malloc() function takes a single parameter, which is the size of the requested memory area in bytes. It returns a pointer to the allocated memory.

Q2. What is the use of pointers?
Ans. Pointers are used extensively in both C and C++ for three main purposes:
to allocate new objects on the heap, to pass functions to other functions. to iterate over elements in arrays or other data structures.

Q3. List various types of pointers.

- NULL pointer
- Dangling pointer
- Generic pointer
- Wild pointer
- Complex pointer

Q4. What is a void pointer?
Ans. A void pointer is a pointer that has no associated data type with it. A void pointer can hold address of any type and can be typcasted to any type.

Q5. What is the use of functions? How are actual parameters different from formal parameters?
Ans. Functions provide a high degree of modularity for your application and also provide better code reusability and ease of maintenance.
Actual parameters are used in function calling statement. Formal parameters are used in function definition statement.

EXPERIMENT - 9

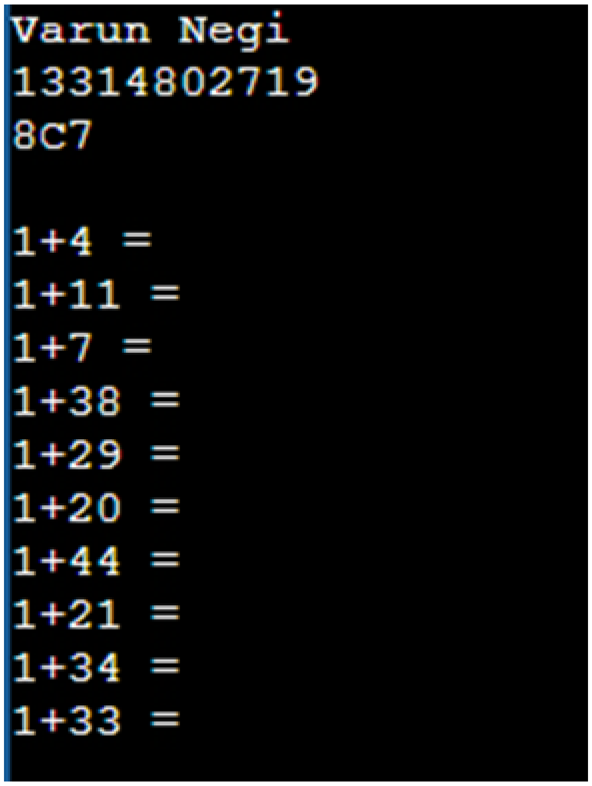
AIM: Write a program to display 10 questions at random out of exp.8 - 50 questions (do not display the answer of these questions to the user now).

Source Code:

```
#include <bits/stdc++.h>
using namespace std;

void generateSetOfNumbers(int arr[]){

                                                    int                p[50]              =
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34
,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49};

   for (int i=49; i>0; --i){
      int j = rand()%i;

      int temp = p[i];
      p[i] = p[j];
      p[j] = temp;
   }

   for (int i=0; i<10; ++i){
      arr[i] = p[i];
   }
}

int main(){
   cout<<"Varun Negi : 13314802719 : 8C7\n\n";
   string ques[50],ans[50];

   int r[10];

   for(int i=0,j=1;i<50;i++,j++){
      stringstream s1,s2;

      s1 << i;
      s2 << j;

      ques[i]="1+";
      ques[i]+=s1.str();
      ans[i]=s2.str();
```

```
    }

    srand(time(0));

    generateSetOfNumbers(r);

    for(int i=0;i<10;i++){
        int x=r[i];
        cout<<ques[x]<<" = "<<endl;
    }

    return 0;

}
```

<u>OUTPUT</u>:



```
Varun Negi
13314802719
8C7

1+4 =
1+11 =
1+7 =
1+38 =
1+29 =
1+20 =
1+44 =
1+21 =
1+34 =
1+33 =
```

VIVA VOCE QUESTIONS

Q1. Why is C called a mid-level programming language?
Ans. C is called a mid-level programming language because it binds the low level and high -level programming language. We can use C language as a System programming to develop the operating system as well as an Application programming to generate menu driven customer driven billing system.

Q2. What is static memory allocation?
Ans. In case of static memory allocation, memory is allocated at compile time, and memory can't be increased while executing the program. It is used in the array. The lifetime of a variable in static memory is the lifetime of a program. The static memory is allocated using static keyword. The static memory is implemented using stacks or heap.

Q3. What is the structure?
Ans. The structure is a user-defined data type that allows storing multiple types of data in a single unit. It occupies the sum of the memory of all members. The structure members can be accessed only through structure variables. Structure variables accessing the same structure but the memory allocated for each variable will be different.

Q4. What is a union?
Ans. The union is a user-defined data type that allows storing multiple types of data in a single unit. However, it doesn't occupy the sum of the memory of all members. It holds the memory of the largest member only. In union, we can access only one variable at a time as it allocates one common space for all the members of a union.

Q5. What is an auto keyword in C?
Ans. In C, every local variable of a function is known as an automatic (auto) variable. Variables which are declared inside the function block are known as a local variable. The local variables are also known as an auto variable. It is optional to use an auto keyword before the data type of a variable. If no value is stored in the local variable, then it consists of a garbage value.

EXPERIMENT - 10

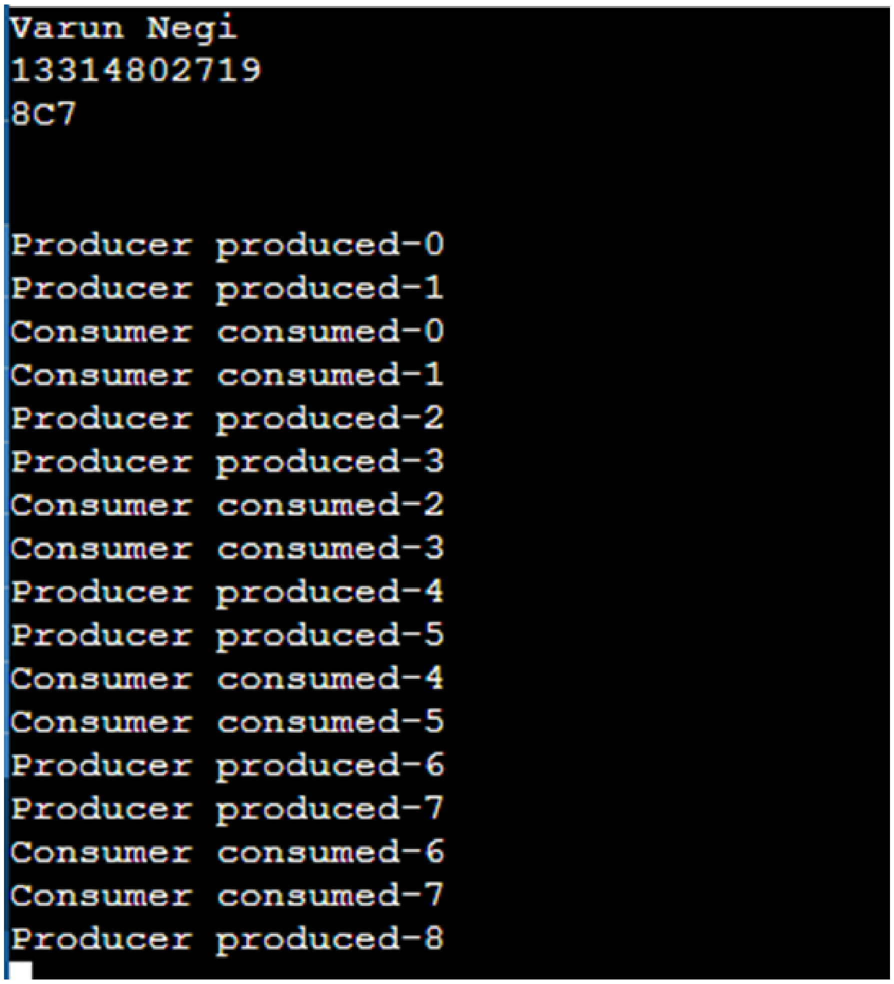
AIM: Implement producer-consumer problem using threads

Source Code:

```
import java.util.LinkedList;
public class Threadexample {
        public static void main(String[] args)
                throws InterruptedException {
                // Object of a class that has both produce() and consume() methods
                final PC pc = new PC();
                Thread t1 = new Thread(new Runnable() {    // Create producer thread
                        @Override
                        public void run(){
                                try {                                   pc.produce();}
                                catch (InterruptedException e) {
                                        e.printStackTrace();    }                       }           });
                Thread t2 = new Thread(new Runnable() {    // Create consumer thread
                        @Override
                        public void run(){
                                try {
                                        pc.consume();}
                                catch (InterruptedException e) {
                                        e.printStackTrace();
                                }}});
        t1.start();    // Start both threads
        t2.start();
        t1.join();     // t1 finishes before t2
        t2.join();
    // This class has a list, producer (adds items to list and consumer (removes items).
    public static class PC {
        // Create a list shared by producer and consumer Size of list is 2.
        LinkedList<Integer> list = new LinkedList<>();
        int capacity = 2;
        // Function called by producer thread
        public void produce() throws InterruptedException {
            int value = 0;
            while (true) {
                synchronized (this) {
                    // producer thread waits while list is full
                    while (list.size() == capacity)            wait();\
                    System.out.println("Producer produced-" + value);
                    list.add(value++);     // to insert the jobs in the list
                // notifies the consumer thread that now it can start consuming
                    notify();
                    // makes the working of program easier to understand
                    Thread.sleep(1000);                        }}}
        // Function called by consumer thread
        public void consume() throws InterruptedException {
            while (true) {
```

```
synchronized (this) {
        // consumer thread waits while list is empty
        while (list.size() == 0)        wait();
        // to retrieve the first job in the list
        int val = list.removeFirst();
        System.out.println("Consumer consumed-" + val);
        notify();    // Wake up producer thread
        Thread.sleep(1000);    // and sleep}}}}}
```

OUTPUT:



```
Varun Negi
13314802719
8C7


Producer produced-0
Producer produced-1
Consumer consumed-0
Consumer consumed-1
Producer produced-2
Producer produced-3
Consumer consumed-2
Consumer consumed-3
Producer produced-4
Producer produced-5
Consumer consumed-4
Consumer consumed-5
Producer produced-6
Producer produced-7
Consumer consumed-6
Consumer consumed-7
Producer produced-8
```

VIVA VOCE QUESTIONS

Q1. Differentiate between call by value and call by reference.

- In Call by value method original value is not modified whereas, in Call by reference method, the original value is modified.
- In Call by value, a copy of the variable is passed whereas in Call by reference, a variable itself is passed.
- In Call by value, actual and formal arguments will be created in different memory locations whereas in Call by reference, actual and formal arguments will be created in the same memory location.

Q2. What is the role of producer and consumer in producer-consumer problem?
Ans. We have a buffer of fixed size. A producer can produce an item and place it in the buffer. A consumer can pick items and consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item. In this problem, buffer is the critical section.

Q3. What is a semaphore?
Ans. Semaphore is simply a variable that is non-negative and shared between threads. A semaphore is a signaling mechanism, and a thread that is waiting on a semaphore can be signaled by another thread. It uses two atomic operations, 1) wait, and 2) signal for the process synchronization.
A semaphore either allows or disallows access to the resource, which depends on how it is set up.

Q4. How are threads created?
Ans. Java lets you create threads in one of two ways:

- By implementing the Runnable Interface
- By extending the Thread class

Q5. Explain Deadlock recovery?
Ans. Deadlock recovery is a critical process that is initiated after a deadlock has been detected in a computer system. This complex process involves a set of actions and procedures that are undertaken to resolve the deadlock by breaking the cycle of resource dependency between the processes involved in the deadlock.