Varuvan Vadivelan Institute of Technology

Dharmapuri – 636 703

LAB MANUAL

Regulation	**: 2013**
Branch	**: *B.E.* – CSE**
Year & Semester	**: I Year / II Semester**

CS6212- PROGRAMMING AND DATA STRUCTURES LABORATORY I

ANNA UNIVERSITY CHENNAI

REGULATION-2013

CS6212 PROGRAMMING AND DATA STRUCTURES LABORATORY I

OBJECTIVES:

The students should be made to:

- Be familiar with c programming
- Be exposed to implementing abstract data types
- Learn to use files
- Learn to implement sorting and searching algorithms.

1. C Programs using Conditional and Control Statements.
2. C Programs using Arrays, Strings and Pointers and Functions.
3. Representation of records using Structures in C – Creation of Linked List – Manipulation of records in a Linked List.
4. File Handling in C – Sequential access – Random Access.
5. Operations on a Stack and Queue – infix to postfix – simple expression evaluation using stacks -Linked Stack Implementation – Linked Queue Implementation.
6. Implementation of sorting algorithms.
7. Implementation of Linear search and Binary Search.

TOTAL: 45 PERIODS

INDEX

CONDITIONAL AND CONTROL STATEMENTS

The ***“if”*** statement is a two-way decision making statement. If a condition holds true, then corresponding true-block is executed; otherwise false-block is executed. The ***“else”*** part of an if statement is optional.

The ***“switch”*** statement tests expression value against a list of case values. When a match is found, statements associated with that case is executed until a ***“break”*** statement is encountered. The default block is executed when none of the case value matches.

The ***“while”*** is an entry-controlled loop, i.e., the condition is evaluated first. If condition is true then body of the loop is executed. The loop is executed repeatedly until the condition holds true. Minimum number of times the loop executed is 0.

The ***“do … while”*** construct provides an exit-controlled loop. Body of the loop is executed once and then condition is evaluated. If true, then the process is repeated. The loop is executed at least once.

The ***“for”*** statement is an entry and counter controlled loop. It contains three parts namely initialize condition and increment/decrement. The counter variable is initialized once and the condition is evaluated. If condition is true, then body of the loop is executed. Counter variable is incremented or decremented each time before testing the condition.

The ***“break”*** statement is used to exit from the loop in which it is contained. The ***“continue”*** statement skips remaining part of the loop for that iteration.

EX. NO: 1

DATE:

IMPLEMENTATION OF SIMPLE CALCULATOR

AIM

To implement a simple calculator using switch case statement.

ALGORITHM

```
1. Start
2. Display calculator menu
3. Read the operator symbol and operands n1, n2
4. If operator = + then
5. calculate result = n1 + n2
6. Else if operator = – then
7. calculate result = n1 – n2
8. Else if operator = * then
9. calculate result = n1 * n2
10. Else if operator = / then
11. calculate result = n1 / n2
12. Else if operator = % then
13. calculate result = n1 % n2
14. Else
15. print "Invalid operator"
16. Print result
17. Stop
```

PROGRAM **(IMPLEMENTATION OF SIMPLE CALCULATOR)**

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
 void main()
{
int n1, n2, result;
char op;
clrscr();
printf("\n Simple Calculator");
printf("\n + Summation");
printf("\n - Difference");
printf("\n * Product");
printf("\n / Quotient");
printf("\n % Remainder");
printf("\n Enter the operator :”);
op = getchar();
printf("Enter operand1 and operand2 :");
scanf("%d%d",&n1,&n2);
switch(op)
{
case '+':
result = n1 +n2;
break;
case '-':
result = n1 - n2;
break;
case '*':
result = n1 * n2;
break;
case '/':
result = n1 / n2;
break;
case '%':
result = n1 % n2;
break;
default:
printf("Invalid operator");
exit(-1);
}
printf("%d %c %d = %d", n1, op, n2,result);
getch();
}
```

OUTPUT

```
Simple Calculator

+Summation

- Difference

* Product

/ Quotient

% Remainder

Enter the operator : -

Enter operand1 and operand2 : 2 4

2 - 4 = -2


Simple Calculator

+Summation

 - Difference

* Product

/ Quotient

 % Remainder

Enter the operator : %

Enter operand1 and operand2 : 5 2

5 % 2 = 1
```

RESULT

Thus simple calculator functionality was executed using menu-oriented approach.

EX. NO: 2

DATE:

GENERATION OF PRIME NUMBER

AIM

To print first 'n' numbers using for loop.

ALGORITHM

1. Start
2. Read the value of n
3. Loop j to generate numbers from 2 to 10000
4. Loop k in the range 2 to j/2
5. Check if divisor exists
6. If j%k = 0 then
7. Examine next j
8. If there are no divisor for j then
9. Print j
10. Increment i by 1
11. If i = n then
12. Stop
13. Else
14. Examine next j
15. Stop.

PROGRAM **(GENERATION OF PRIME NUMBER)**

```
#include <stdio.h>
#include <conio.h>
Void main()
{
int i=0,j,k,n,flg;
clrscr();
printf("\n Enter value for n : ");
scanf("%d", &n);
printf("Prime numbers : ");
for(j=2; j<=10000; j++)
{
flg = 0;
for(k=2; k<=j/2; k++)
{
if (j%k == 0)
{
flg = 1;
break;
}
}
if (flg == 0)
{
printf("%d", j);
i++;
}
if (i == n)
break;
}
getch();
}
```

OUTPUT

```
Enter value for n:  9
Prime numbers: 2    3    5    7    11    13    17    19    23
```

RESULT

Thus first set of prime numbers is displayed.

EX. NO: 3

DATE:

SUMMATION OF N DIGITS

AIM

To find the sum of the digits of a given number using while statement.

ALGORITHM

1. Start
2. Read num
3. Initialize sum to 0.
4. Repeat until num = 0
5. Obtain last digit d = num % 10
6. Add d to sum
7. num = num / 10
8. Print sum
9. Stop

PROGRAM (SUMMATION OF N DIGITS)

```
#include <stdio.h>
#include <conio.h>
main()
{
int n, d, sum;
clrscr();
printf("Enter the numbers : ");
scanf("%d", &n);
sum = 0;
while(n)
{
d=n % 10;
sum=sum+d;
n=n/10;}
printf("Sum of digits : %d", sum);
getch();
}
```

OUTPUT

```
Enter the numbers:    5     8     3     4     9

Sum of digits: 29
```

RESULT

Thus digits of the given number were summed up.

EX. NO: 4

DATE:

GENERATION OF FIBONACCI SERIES

AIM

To print first N terms of the Fibonacci series assuming the first two terms as 0 and 1

ALGORITHM

1. Start
2. Read no. of terms n
3. Initialize f1 to 0 and f2 to 1
4. Print f1 and f2
5. Initialize i to 3
6. Repeat until i < n
7. Generate next term f3 = f1 + f2
8. Print f3
9. f1 = f2
10. f2 = f3
11. Increment i by 1
12. Stop

PROGRAM (**GENERATION OF FIBONACCI SERIES**)

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i, n;
long f1, f2, f3;
clrscr();
f1 = 0;
f2 = 1;
printf("Enter number of terms : ");
scanf("%d", &n);
printf("\n Fibonacci series \n");
printf("%ld \n %ld \n",f1,f2);
for(i=3; i<=n; i++)
{
f3 = f1 + f2;
printf("%ld",f3);
f1 = f2;
f2 = f3;
}
getch();
}
```

OUTPUT

```
Enter number of terms: 10
Fibonacci series        0       1       1       2       3       5       8       13      21      34
```

RESULT

Thus first ‘n’ terms of Fibonacci series was generated.

ARRAYS, STRINGS, POINTERS AND FUNCTIONS

A function is a block of code which is used or to achieve a specific task. Functions facilitate top-down modular programming and avoid code redundancy. When functions are used in a program, its prototype must be declared. Prototype specifies return type, function name, number and type of arguments that the function may take. ***“void”*** is used as return type, if the function does not return a value. Parameters can be passed to function either as value or reference (address). A function that calls itself is called recursive function.

An array is a collection of homogeneous elements i.e., of the same data-type. When an array is declared a contiguous memory is allocated. Array index always start with zero and the last index is size-1. Arrays can be either one dimensional or two dimensional. Array can be passed as an argument to a function only by reference. Memory allocation using arrays are static whereas using functions such as ***“malloc”*** is dynamic.

String is a sequence of characters terminated by a null character '\0'. String is declared as character array. C library provides string handling function under header file <string.h>. String variables are not preceded by a ***“&”*** in a “***scanf”*** statement. The function “***gets***" is used to read string with embedded whitespace.

Structure is a user-defined data type and is a collection of heterogeneous elements, i.e. elements can have dissimilar types. Elements in a structure are referred as members. Each member within a structure is assigned its own unique storage area. Memory allocation for a structure is sum of memory required for each member. It is generally obtained using ***“sizeof”*** operator. Members of a structure are accessed using dot (.) operator.

Pointer variable is a variable that can store the address of another variable. Pointer variables are prefixed with ***“*”*** operator in declaration statement. Address of a variable is obtained using & (address) operator. A pointer variable can point to elements of the same type only. A pointer variable can access value of the variable pointed to by using the ***“*”*** dereference operator. Arguments must be of pointer type for pass-by-reference. When arguments are passed by reference, changes made within the function are reflected in the calling function. Pointers can be used to voluminous data by passing an array (starting address).

EX. NO: 5

DATE:

FACTORIAL OF N NUMBER USING RECURSION

AIM

To find the factorial of a number using recursive function.

ALGORITHM

1. Start
2. Read the value of n
3. Call function factorial (n)
4. Print return value
5. Stop
6. Function factorial (n)
7. If n=1 then return 1
8. Else return n*factorial (n-1)

PROGRAM (FACTORIAL OF N NUMBER USING RECURSION)

```
#include <stdio.h>
#include <conio.h>
long factorial(int);
void main ()
{
int n;
long f;
clrscr();
printf("Enter a number : ");
scanf("%d", &n);
f = factorial(n);
printf("Factorial value : %ld",f);
getch();
}
long factorial(int n)
{
if (n <= 1)
return(1);
else
return (n * factorial(n-1));
}
```

OUTPUT

```
Enter a number:      3
Factorial value:     6
```

RESULT

Thus factorial value of a given number was obtained through recursive function call.

EX. NO: 6

DATE:

MAXIMUM OF N NUMBERS USING ARRAY

AIM

To find the greatest of ’N’ numbers stored in an array.

ALGORITHM

1. Start
2. Read number of array elements as n
3. Read array elements Ai, i = 0, 1, 2…n–1
4. Assume first element A0 to be max
5. Compare each array element Ai with max
6. If max <Ai then max = Ai
7. Print max
8. Stop

PROGRAM (MAXIMUM OF N NUMBERS USING ARRAY)

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a[10];
int i, max, n;
clrscr();
printf("Enter number of elements : ");
scanf("%d", &n);
printf("Enter Array Elements \n");
for(i=0; i<n; i++)
scanf("%d", &a[i]);
max = a[0];
for(i=1; i<n; i++)
{
if (max < a[i])
max = a[i];
}
printf("Maximum value = %d ",max);
getch();
}
```

OUTPUT

```
Enter number of elements: 6

Enter Array Elements

3

8

-7

11

-9

0

Maximum value = 11
```

RESULT

Thus maximum element of an array was determined.

EX. NO: 7

DATE:

COMPUTATION OF MATRIX MULTIPLICATION

AIM

To compute product of two matrices using two dimensional array.

ALGORITHM

1. Start
2. Read the order of matrix A as m and n
3. Read the order of matrix B as p and q
4. If np then print "Multiplication not possible" and Stop
5. Read matrix A elements Aij
6. Read matrix B elements Bij
7. Initialize matrix C elements Cij to 0
8. Compute product matrix Cij = Cij + Aik * Bkj where 0 i < m, 0 j < q and 0 k < n
9. Print matrix Cij,
10. Stop

PROGRAM (COMPUTATION OF MATRIX MULTIPLICATION)

```
#include <stdio.h>
#include <conio.h>
main()
{
int a[10][10], b[10][10], c[10][10]; int r1, c1, r2, c2; inti,
j, k;
clrscr();
printf("Enter order of matrix A : ");
scanf("%d%d", &r1, &c1);
printf("Enter order of matrix B : ");
scanf("%d%d", &r2, &c2);
if (c1 != r2)
{
printf("Matrix multiplication not possible");
getch();
exit(0);
}
printf("Enter matrix Aelements\n");
for(i=0; i<r1; i++)
 {
for(j=0; j<c1; j++)
{
scanf("%d", &a[i][j]);
}
}
printf("Enter matrix Belements\n");
for(i=0; i<r2; i++)
 {
for(j=0; j<c2; j++)
{
scanf("%d", &b[i][j]);
}
}
for(i=0; i<r1; i++)
{
for(j=0; j<c2; j++)
{
c[i][j] = 0;
for(k=0; k<c1; k++)
{
c[i][j] = c[i][j] + a[i][k] * b[k][j];
}
}
}
printf("Product matrix C\n");
for(i=0; i<r1; i++)
{
for(j=0; j<c2; j++)
{
printf("%-4d",c[i][j]);
}
printf("\n");
}
getch();
}
```

OUTPUT

```
Enter order of matrix A:    2    3

Enter order of matrix B:    3    2

Enter matrix A elements

1    1    1

1    1    1

Enter matrix B elements

2    2

2    2

2    2

Product matrix C

6    6

6    6
```

RESULT

Thus product of given two matrices was obtained using arrays.

EX. NO. 8

DATE:

DETERMINATION OF PALINDROME

AIM

To determine whether the input string is palindrome using string handling functions.

ALGORITHM

```
1.  Start
2.  Read the string, say str
3.  Copy str onto rev using strcpy function
4.  Reverse rev using strrev function
5.  Compare str and rev using strcmp function
6.  If outcome = 0 then
7.  Print "Given string is palindrome"
8.  Else
9.  Print "Given string is not palindrome"
10. Stop
```

PROGRAM (DETERMINATION OF PALINDROME)

```
#include <string.h>
#include <stdio.h>
#include <conio.h>
void main()
{
charstr[40], rev[40];
int x;
clrscr();
printf("Enter the string: ");
scanf("%s", str);
strcpy(rev, str);
strrev(rev);
printf("Reversed string is: %s \n",rev);
x = strcmpi(str, rev);
if (x == 0)
printf("Given string is a palindrome");
else
printf("Given string is not a palindrome");
getch();
}
```

OUTPUT

```
Enter the string: malayalam
Reversed string is: malayalam
Given string is a palindrome

Enter the string: Computer
Reversed string is: retupmoC
Given string is not a palindrome
```

RESULT

Thus the given input string is checked for palindrome using string handling functions.

STRUCTURES AND LINKED LISTS

C supports a constructed (user-defined) data type known as structure, which is a method of packing related data of different types i.e., heterogeneous elements. A single structure may contain integer, floating-point, character, arrays and pointer elements. The individual elements are referred as members.

Each member within a structure is assigned its own unique storage area. The amount of memory required to store structure is sum of storage capacity of all members. Each individual member of a structure is accessed by means of a member selector **“.”** operator.

Union is similar to structures. The members that compose a union all share the same storage area. The amount of memory required is same as that required for its largest member.

Self-Referential structure is a structure where one of its members is a pointer to the structure itself. Such structures are very useful in applications involving linked data structures such as list and trees.

A linked list is a set of nodes where each node has two field’s data and a link. The link field points to the next node by storing address of the next node. An operation on a linked list includes insertion and deletion of a node and traversal of the list. Backward traversal is not possible in a singly linked list.

Doubly linked list node contains an additional pointer that contains address of the previous node in the list. Both forward and backward traversal is possible in a doubly linked list.

EX. NO: 9

DATE:

GENERATION OF PAYROLL APPLICATION

AIM

To generate employee payroll for an organization using structure.

ALGORITHM

1. Start
2. Define employee structure with fields empid, ename, basic, hra, da, it, gross and netpay
3. Read number of employees n
4. Read empid, ename, and basic for n employees in an array of structure.
5. For each employee, compute
6. hra = 2% of basic
7. da = 1% of basic
8. gross = basic + hra + da
9. it = 5% of basic
10. netpay = gross - it
11. Print empid, ename, basic, hra, da, it, gross and netpay for all employees
12. Stop

PROGRAM (GENERATION OF PAYROLL APPLICATION)

```
#include <stdio.h>
#include <conio.h>
struct employee
{
int empid;
char name[15];
int basic;
float hra;
float da;
float it;
float gross;
float netpay;
};
void main()
{
struct employee emp[50];
inti, j, n;
clrscr();
printf("\n Enter No. of Employees : ");
scanf("%d", &n);
for(i=0; i<n ;i++)
{
printf("\n Enter Employee Details \n");
printf("Enter Employee Id : ");
scanf("%d", &emp[i].empid);
printf("Enter Employee Name : ");
scanf("%s", emp[i].ename);
printf("Enter Basic Salary : ");
scanf("%d", &emp[i].basic);
}
for(i=0; i<n; i++)
{
emp[i].hra = 0.02 * emp[i].basic;
emp[i].da= 0.01 * emp[i].basic;
emp[i].it= 0.05 * emp[i].basic;
emp[i].gross = emp[i].basic + emp[i].hra + emp[i].da;
emp[i].netpay = emp[i].gross - emp[i].it;
}
printf("\n\n\n\t\t\t\tXYZ& Co.Payroll\n\n");
for(i=0;i<80;i++)
printf("*");
printf("EmpId\tName\t\tBasic\t   HRA\t   DA\t   IT\tGross\t\tNet
ay\n");
for(i=0;i<80;i++)
printf("*");
for(i=0; i<n; i++)
{
printf("\n%d\t%-
15s\t%d\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f",emp[i].empid,emp[i].ena
me,    emp[i].basic,    emp[i].hra,    emp[i].da,    emp[i].it,
emp[i].gross, emp[i].netpay);
}
printf("\n");
for(i=0;i<80;i++)
printf("*");
getch();
}
```

OUTPUT

```
Enter No. of Employees: 2

Enter Employee Details
Enter Employee Id      : 436
Enter Employee Name: Gopal
Enter Basic Salary      : 10000

Enter Employee Details
Enter Employee Id      : 463
Enter Employee Name: Rajesh
Enter Basic Salary      : 22000

XYZ & Co. Payroll
**************************************************************************
EmpId Name   Basic   HRA    DA      IT        Gross   Net Pay
**************************************************************************
436      Gopal  10000  200.00 100.00 500.00  10300.00        9800.00
463      Rajesh 22000  440.00 220.00 1100.00 22660.00       21560.00
**************************************************************************
```

RESULT

Thus payroll for employees was generated using structure.

EX. NO: 10

DATE:

IMPLEMENTATION OF SINGLY LINKED LIST

AIM

To define a singly linked list node and perform operations such as insertions and deletions dynamically.

ALGORITHM

1. Start
2. Define single linked list node as self referential structure
3. Create Head node with label = -1 and next = NULL using
4. Display menu on list operation
5. Accept user choice
6. If choice = 1 then
7. Locate node after which insertion is to be done
8. Create a new node and get data part
9. Insert the new node at appropriate position by manipulating address Else if choice = 2
10. Get node's data to be deleted.
11. Locate the node and delink the node
12. Rearrange the links
13. Else
14. Traverse the list from Head node to node which points to null
15. Stop

PROGRAM (IMPLEMENTATION OF SINGLY LINKED LIST)

```
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>
#include <string.h>
struct node
{
int label;
struct node *next;
};
void main()
{
int ch, fou=0;
int k;
struct node *h, *temp, *head, *h1;
head = (struct node*) malloc(sizeof(struct node));
head->label = -1;
head->next = NULL;
while(-1)
{
clrscr();
printf("\n\n SINGLY LINKED LIST OPERATIONS \n");
printf("1->Add");
printf("2->Delete");
printf("3->View");
printf("4->Exit\n");
printf("Enter your choice : ");
scanf("%d", &ch);
switch(ch)
{
case 1:
printf("\n Enter label after which to add : ");
scanf("%d", &k);
h = head;
fou = 0;
if (h->label == k)
fou = 1;
while(h->next != NULL)
{
if (h->label == k)
{
fou=1;
break;
}
h = h->next;
}
if (h->label == k)
fou = 1;
if (fou != 1)
printf("Node not found\n");
else
{
temp=(struct node *)(malloc(sizeof(struct node)));
printf("Enter label for new node : ");
scanf("%d", &temp->label);
temp->next = h->next;
h->next = temp;
}
break;
case 2:
printf("Enter label of node to be deleted\n");
scanf("%d", &k);
fou = 0;
h = h1 = head;
while (h->next != NULL)
{
h = h->next;
if (h->label == k)
{
fou = 1;
break;
}
}
if (fou == 0)
printf("Sorry Node not found\n");
else
{
while (h1->next != h)
h1 = h1->next;
h1->next = h->next;
free(h);
printf("Node deleted successfully \n");
}
break;
case 3:
printf("\n\n HEAD -> ");
h=head;
```

```
while (h->next != NULL)
{
h = h->next;
printf("%d -> ",h->label);
}
printf("NULL");
break;
case 4:
exit(0);
}
}
}
```

OUTPUT

```
SINGLY LINKED LIST OPERATIONS

1->Add  2->Delete     3->View         4->Exit

Enter your choice       : 1

Enter label after which new node is to be added : -1

Enter label for new node : 23


SINGLY LINKED LIST OPERATIONS

1->Add  2->Delete     3->View         4->Exit

Enter your choice       : 1

Enter label after which new node is to be added : 23

Enter label for new node : 67


SINGLY LINKED LIST OPERATIONS

1->Add  2->Delete     3->View         4->Exit

Enter your choice       : 3

HEAD -> 23 -> 67      -> NULL
```

RESULT

Thus operation on single linked list is performed.

FILE HANDLING

Applications require information stored on auxiliary storage device. Such information is stored permanently as a data file that allows accessing and altering the information whenever necessary.

Prior to performing any activity of a file, the file should be opened. By opening a file, link between the program and the operating system is established. This link exists by means of a structure termed as FILE, which is specified in header file ***“<stdio.h>”***.

A file is opened using the standard function ***“fopen()”***. If a file could not be opened the ***“fopen()”*** returns a NULL. A file is opened in the given mode such as reading, writing, appending, etc. After performing file I/O, the file is closed.

File access could be either sequential or random. Sequential access starts with the first data set, fetch the next and so on until end-of-file is encountered. End-of-file condition can be checked using ***“feof”*** function.

Random access means moving file pointer to the desired byte. Pre-defined functions that facilitate random access are:

“ftell”—to know current position of the file pointer

“rewind”—to move file pointer to beginning of the file

“fseek”—used to move the file pointer to the desired byte.

File I/O is generally either character or block oriented. Functions ***“getc”*** and ***“putc”*** is used to read/write a single character from the given file. Functions **“fread”** and ***“fwrite”*** is used to perform I/O as blocks of data, where each block is a fixed number of contiguous bytes. A block is generally represented as a structure.

Command-line arguments allow parameters to be passed to the main function on execution. The two command-line arguments are ***“argc”***, an integer variable whose value is assigned to number of arguments given and ***“argv”***, a string array that contains the list of arguments. The main function prototype is ***“main(int argc, char *argv[])”*** to support command-line arguments.

EX.NO: 11

DATE:

SEQUENTIAL FILE ACCESS

AIM

To create a telephone directory and to locate a user details using sequential access.

ALGORITHM

1. Start
2. Open empseq.dat file in append mode
3. Add records to the file
4. Close the file
5. Open empseq.dat file in read mode
6. Get person name.
7. Check each record one by one from the first record
8. If person name matches then print the details.
9. Close the file
10. Stop

PROGRAM (SEQUENTIAL FILE ACCESS)

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
struct employee
{
char name[20];
long mno;
};
void main()
{
struct employee emp1, emp2;
FILE *fd1, *fd2;
char str[20];
int found = 0;
fd1 = fopen("empseq.dat", "a");
printf("\n\t Enter employee details \n");
while(1)
{
printf("\n Enter Name (\"xxx\" to quit) : ");
scanf("%s", emp1.name);
if (strcmp(emp1.name,"xxx") == 0)
break;
printf("Enter Contact No. : ");
scanf("%ld", &emp1.mno);
fwrite (&emp1, sizeof(emp1), 1, fd1);
}
fclose(fd1);
fd2 = fopen("empseq.dat", "r");
printf("\n Enter Employee name to get phone no. : ");
scanf("%s", str);
while(fread(&emp2, sizeof(emp2), 1, fd2))
{
if (strcmp(emp2.name, str) == 0)
{
printf("Telephone No. : %ld\n", emp2.mno);
 found = 1;
break;
}
}
fclose(fd2);
if(!found)
printf("\n Employee does not exist");
}
```

OUTPUT

```
Enter employee details

Enter Name ("xxx" to quit): vijai

Enter Contact No. : 1234567899

Enter Name ("xxx" to quit) : anand

Enter Contact No. : 9876543211

Enter Name ("xxx" to quit): xxx

Enter Employee name to get phone no: anand

Telephone No. : 9876543211
```

RESULT

Thus sequential access is performed to retrieve a person's contact details.

EX. NO : 12

Date:

RANDOM FILE ACCESS

AIM

To create a address book and to locate employee details using random access.

ALGORITHM

1. Start
2. Open emprand.dat file in append mode
3. Automatically generate employee id
4. Add records to the file
5. Close the file
6. Open emprand.dat file in read mode
7. Get employee id.
8. Move the file pointer to the desired record using fseek function
9. Print the details.
10. Close the file
11. Stop

PROGRAM (RANDOM FILE ACCESS)

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
struct employee
{
Int empid;
char name[20];
char desig[20];
};
void main()
{
struct employee emp1, emp2;
FILE *fd1, *fd2;
char str[20];
int id, eno, pos, size;
fd1 = fopen("emprand.dat", "a");
fseek(fd1, 0, 2);
size = ftell(fd1);
id = 100 + size / sizeof(emp1);
printf("\n Enter employee details \n");
while(1)
{
emp1.empid = id++;
printf("\n Employee Id : %d", emp1.empid);
printf("\n Enter Name (\"xxx\" to quit) : ");
scanf("%s", emp1.name);
if (strcmp(emp1.name,"xxx") = =0)
break;
printf("Enter Designation : ");
scanf("%s", emp1.desig);
fwrite (&emp1, sizeof(emp1), 1, fd1);
}
size = ftell(fd1);
fclose(fd1);
fd2 = fopen("emprand.dat", "r");
printf("\n Enter Employee id : ");
scanf("%d", &eno);
pos = (eno - 100) * sizeof(emp2);
if (pos < size)
{
fseek(fd2, pos, 0);
fread(&emp2, sizeof(emp2), 1, fd2);
printf("Employee Name : %s\n", emp2.name);
printf("Designation: %s\n", emp2.desig);
}
else
printf("\n Incorrect Employee Id \n");
fclose(fd2);
}
```

OUTPUT

```
Enter employee details
Employee Id: 100
Enter Name ("xxx" to quit): gopal
Enter Designation: AP

Employee Id: 101
Enter Name ("xxx" to quit) : arum
Enter Designation: AP

Employee Id: 102
Enter Name ("xxx" to quit): Raju
Enter Designation: Prof

Employee Id: 103
Enter Name ("xxx" to quit): xxx

Enter Employee id: 102
Employee Name: Raju
Designation:Prof
```

RESULT

Thus random access is performed to directly obtain employee details.

STACKS AND QUEUES

The stack is a list of elements placed on top of each other. A pointer always points to the topmost position of a stack called top. The two possible operations on stack is push and pop. It is implemented either using arrays or linked lists. The top is an indicator where both push and pop operations are performed. CDs in a disc spindle are an example of stack.

The insertion operation is termed as push. For each insertion the pointer top is incremented so that it retains the top-most position in the stack. The deletion operation of stack is termed as pop. Stack follows LIFO (Last In First Out) method i.e., the last inserted element would be the first element to be deleted. The pointer top is made to point the next element in order.

In a Queue, the set of elements are placed one after another, just like people standing in a queue at a reservation counter. In a Queue, there are two pointers namely front and rear. Pointer "***front*"** points the first element and ***"rear"*** the last element in the Queue.

Queue is represented as FIFO (First In First Out). Insertions are carried out at the REAR end the pointer is updated. Deletion is performed at the FRONT end removing the oldest element and the pointer is updated.

Applications of stack include infix to postfix conversion and evaluation of expression in postfix form. An infix arithmetic expression can be converted into a postfix expression if precedence of operators is known. Operators are sandwiched between operands in an infix expression whereas operators appear after operands in postfix form. Expression in postfix form is evaluated by applying operators on operands to its immediate left.

EX. NO. 13

DATE:

IMPLEMENTATION OF STACK USING LINKED LIST

AIM

To implement stack operations using linked list.

ALGORITHM

1. Start
2. Define a singly linked list node for stack
3. Create Head node
4. Display a menu listing stack operations
5. Accept choice
6. If choice = 1 then
7. Create a new node with data
8. Make new node point to first node
9. Make head node point to new node
10. If choice = 2 then
11. Make temp node point to first node
12. Make head node point to next of temp node
13. Release memory
14. If choice = 3 then
15. Display stack elements starting from head node till null
16. Stop

PROGRAM (IMPLEMENTATION OF STACK USING LINKED LIST)

```
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>
struct node
{
int label;
struct node *next;
};
void main()
{
int ch = 0;
int k;
struct node *h, *temp, *head;
head = (struct node*) malloc(sizeof(struct node));
head->next = NULL;
while(1)
{
printf("\n Stack using Linked List \n");
printf("1->Push ");
printf("2->Pop ");
printf("3->View");
printf("4->Exit \n");
printf("Enter your choice : ");
scanf("%d", &ch);
switch(ch)
{
case 1:
temp=(struct node *)(malloc(sizeof(struct node)));
printf("Enter label for new node : ");
scanf("%d", &temp->label);
h = head;
temp->next = h->next;
h->next = temp;
break;
case 2:
h = head->next;
head->next = h->next;
printf("Node %s deleted\n", h->label);
free(h);
break;
case 3:
printf("\n HEAD -> ");
h = head;
while(h->next != NULL)
{
h = h->next;
printf("%d -> ",h->label);
}
printf("NULL \n");
break;
case 4:
exit(0);
}
}
}
```

OUTPUT

```
Stack using Linked List
1->Push     2->Pop      3->View     4->Exit
Enter your choice : 1
Enter label for new node : 23

Stack using Linked List
1->Push     2->Pop      3->View     4->Exit
Enter your choice : 1
Enter label for new node : 34

Stack using Linked List
1->Push     2->Pop      3->View     4->Exit
Enter your choice : 3
HEAD -> 34 -> 23 -> NULL
```

RESULT

Thus operations of a stack were demonstrated using linked list.

EX. NO. 14

DATE:

IMPLEMENTATION OF QUEUE USING LINKED LIST

AIM

To implement queue operations using linked list.

ALGORITHM

1. Start
2. Define a singly linked list node for stack
3. Create Head node
4. Display a menu listing stack operations
5. Accept choice
6. If choice = 1 then
7. Create a new node with data
8. Make new node point to first node
9. Make head node point to new node
10. If choice = 2 then
11. Make temp node point to first node
12. Make head node point to next of temp node
13. Release memory
14. If choice = 3 then
15. Display stack elements starting from head node till null
16. Stop

PROGRAM **(IMPLEMENTATION OF QUEUE USING LINKED LIST)**

```
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>
struct node
{
int label;
struct node *next;
};
void main()
{
int ch=0;
int k;
struct node *h, *temp, *head;
head = (struct node*) malloc(sizeof(struct node)); head->next
= NULL;
while(1)
{
printf("\n Queue using Linked List \n");
printf("1->Insert ");
printf("2->Delete ");
printf("3->View ");
printf("4->Exit \n");
printf("Enter your choice : ");
scanf("%d", &ch);
switch(ch)
{
case 1:
temp=(struct node *)(malloc(sizeof(struct node)));
printf("Enter label for new node : ");
scanf("%d", &temp->label);
h = head;
while (h->next != NULL)
h = h->next;
h->next = temp;
temp->next = NULL;
break;
case 2:
h = head->next;
head->next = h->next;
printf("Node deleted \n");
free(h);
break;
case 3:
printf("\n\nHEAD -> ");
h=head;
while (h->next!=NULL)
{
h = h->next;
printf("%d -> ",h->label);
}
printf("NULL \n");
break;
case 4:
exit(0);
}
}
```

OUTPUT

```
Queue using Linked List

1->Insert  2- >Delete  3->View  4->Exit

Enter your choice : 1

Enter label for new node : 12


Queue using Linked

List 1->Insert  2- >Delete  3->View  4->Exit  Enter your choice : 1

Enter label for new node : 23


Queue using Linked List

1->Insert  2->Delete  3->View  4->Exit  Enter your choice : 3

HEAD -> 12 -> 23 -> NULL
```

RESULT

Thus operations of a Queue were demonstrated using linked list.

EX. NO. 15

DATE:

CONVERSION OF INFIX TO POSTFIX EXPRESSION

AIM

To convert infix expression to its postfix form using stack operations.

ALGORITHM

1. Start
2. Define a array stack of size max = 20
3. Initialize top = -1
4. Read the infix expression character-by-character If character is an operand print it
5. If character is an operator
6. Compare the operator’s priority with the stack[top] operator.
7. If the stack [top] operator has higher or equal priority than the input
8. operator,
9. Pop it from the stack and print it
10. Else
11. Push the input operator onto the stack
12. If character is a left parenthesis, then push it onto the stack.
13. If the character is a right parenthesis, pop all the operators from the stack and print it
14. Until a left parenthesis is encountered. Do not print the parenthesis.
15. Stop

PROGRAM (CONVERSION OF INFIX TO POSTFIX EXPRESSION)

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define MAX 20
int top = -1;
char stack[MAX];
char pop();
void push(char item);
int prcd(char symbol)
{
switch(symbol)
{
case '+':
case '-':
return 2;
break;
case '*':
case '/':
return 4;
break;
case '^':
case '$':
return 6;
break;
case '(':
case ')':
case '#':
return 1;
break;
} }
int isoperator(char symbol)
{
switch(symbol)
{
case '+':
case '-':
case '*':
case '/':
case '^':
case '$':
case '(':
case ')':
return 1;
```

```
break;
default:
return 0;
}
}
void convertip(char infix[],char postfix[])
{
int i,symbol,j = 0;
stack[++top] = '#';
for(i=0;i<strlen(infix);i++)
{
symbol = infix[i];
if(isoperator(symbol) == 0)
{
postfix[j] = symbol;
j++;
}
else
{
if(symbol == '(')
push(symbol);
else if(symbol == ')')
{
while(stack[top] != '(')
{
postfix[j] = pop();
j++;
}
pop();
}
else
{
if(prcd(symbol) > prcd(stack[top]))
push(symbol);
else
{
while(prcd(symbol) <= prcd(stack[top]))
{
postfix[j] = pop();
j++;
}
push(symbol);
}
}
}
}
while(stack[top] != '#')
{
postfix[j] = pop();
j++;
}
postfix[j] = '\0';
}
void main()
{
char infix[20],postfix[20];
clrscr();
printf("Enter the valid infix string: ");
gets(infix);
convertip(infix, postfix);
printf("The corresponding postfix string is: ");
 puts(postfix);
getch();
}
void push(char item)
{
top++;
stack[top] = item;
}
char pop()
{
char a;
a = stack[top];
top--;
return a;
}
```

OUTPUT

```
Enter the valid infix string: (a+b*c)/(d$e)
The corresponding postfix string is: abc*+de$/

Enter the valid infix string: a*b+c*d/e
The corresponding postfix string is: ab*cd*e/+

Enter the valid infix string: a+b*c+(d*e+f)*g
The corresponding postfix string is: abc*+de*f+g*+
```

RESULT

Thus the given infix expression was converted into postfix form using stack.

EX. NO: 16

DATE:

EVALUATION OF POSTFIX EXPRESSION

AIM

To evaluate the given postfix expression using stack operations.

ALGORITHM

1. Start
2. Define a array stack of size max = 20
3. Initialize top = -1
4. Read the postfix expression character-by-character
5. If character is an operand push it onto the stack
6. If character is an operator
7. Pop topmost two elements from stack.
8. Apply operator on the elements and push the result onto the stack,
9. Eventually only result will be in the stack at end of the expression.
10. Pop the result and print it.
11. Stop

PROGRAM (EVALUATION OF POSTFIX EXPRESSION)

```
#include <stdio.h>
#include <conio.h>
struct stack
{
int top;
float a[50];
}s;
void main()
{
char pf[50];
float d1,d2,d3;
int i;
clrscr();
s.top = -1;
printf("\n\n Enter the postfix expression: ");
gets(pf);
for(i=0; pf[i]!='\0'; i++)
{
switch(pf[i])
{
case '0':
case '1':
case '2':
case '3':
case '4':
case '5':
case '6':
case '7':
case '8':
case '9':
s.a[++s.top] = pf[i]-'0';
break;
case '+':
d1 = s.a[s.top--];
d2 = s.a[s.top--];
s.a[++s.top] = d1 + d2;
break;
case '-':
d2 = s.a[s.top--];
d1 = s.a[s.top--];
s.a[++s.top] = d1 - d2;
break;
case ‘*’:
```

```
d2= s.a[s.top--];
d1= s.a[s.top--];
s.a[++s.top] = d1*d2;
break;
case '/':
d2= s.a[s.top--];
d1= s.a[s.top--];
s.a[++s.top] = d1 / d2;
break;
}
}
printf("\n Expression value is %5.2f", s.a[s.top]);
getch();
}
```

OUTPUT

```
Enter the postfix expression: 6523+8*+3+*
Expression value is  288.00
```

RESULT

Thus the given postfix expression was evaluated using stack.

SORTING

Sorting algorithms take input an unsorted array and number of array elements says n. Sorting is basically based on comparison. Each stage in a sorting algorithm is called a pass.

Bubble sort is an older, easier and inefficient sorting algorithm. It works by comparing each element of the array with the element next to it and swapping if required. After each pass, smaller values are bubbled to top of the list.

Insertion sort is one of the simplest algorithms. It consists of n-1 passes. For any pass p, elements in positions 1 through p are in sorted order. In pass p, the p^{th} element left is moved until its correct place is found among the first p elements.

Shell sort works by comparing elements that are distant. The distance between comparisons decreases as the algorithm runs until the last phase, in which adjacent elements are compared. Shell sort uses a sequence, h_1, h_2, h_t, called the increment sequence. After a phase, using some increment h_k, all elements spaced h_k apart are sorted.

Merge sort algorithm merges two sorted lists. It is a recursive algorithm, where in merge sort is recursively applied to both halves of the original list. It is a classic divide-and-conquer strategy. Once the subsets are sorted, entries in both sets are compared, and whichever is less is put on to merged list.

Quick sort is the fastest known sorting algorithm. Like merge sort, quick sort is a divide-and-conquer recursive algorithm. It is based on selection of pivot element. Original list is partitioned with elements less than pivot and elements greater than pivot.

Selection sort is a very simple algorithm. It determines the minimum and swaps it with the element at the index where it is supposed to be. The process is repeated such that nth minimum of the list is swapped with the element at n-1th index of the array.

EX. NO. 17

DATE:

SORTING N NUMBERS USING QUICK SORT

AIM

To sort an array of N numbers using Quick sort.

ALGORITHM

1. Start
2. Read number of array elements n
3. Read array elements Ai
4. Select an pivot element x from Ai
5. Divide the array into 3 sequences: elements < x, x, elements > x
6. Recursively quick sort both sets (Ai < x and Ai > x)
7. Display the sorted array elements
8. Stop

PROGRAM(SORTING N NUMBERS USING QUICK SORT)

```
#include<stdio.h>
#include<conio.h>
void qsort(int arr[20], int fst, int last);
void main()
{
int arr[30];
int i, size;
printf("Enter total no. of the elements : ");
 scanf("%d", &size);
printf("Enter total %d elements : \n", size);
for(i=0; i<size; i++)
scanf("%d", &arr[i]);
qsort(arr,0,size-1);
printf("\n Quick sorted elements \n");
for(i=0; i<size; i++)
printf("%d\t", arr[i]);
getch();
}
void qsort(int arr[20], int fst, int last)
{
int i, j, pivot, tmp;
if(fst < last)
{
pivot = fst;
i = fst;
j = last;
while(i < j)
{
while(arr[i] <=arr[pivot] && i<last)
i++;
while(arr[j] > arr[pivot])
j--;
if(i <j )
{
tmp = arr[i];
arr[i] = arr[j];
arr[j] = tmp;
}
}
tmp = arr[pivot];
arr[pivot] = arr[j];
arr[j] = tmp;
qsort(arr, fst, j-1);
qsort(arr, j+1, last);
}
}
```

OUTPUT

```
Enter total no. of the elements: 8

Enter total 8 elements:

1

2

7

-1

0

4

-2

3


Quick sorted element

-2      -1      0       1       2       3       4       7
```

RESULT

Thus an array was sorted using quick sort's divide and conquers method.

EX. NO. 18

DATE:

SORTING N NUMBERS USING MERGE SORT

AIM

To sort an array of N numbers using Merge sort.

ALGORITHM

1. Start
2. Read number of array elements n
3. Read array elements Ai
4. Divide the array into sub-arrays with a set of elements
5. Recursively sort the sub-arrays
6. Display both sorted sub-arrays

PROGRAM (SORTING N NUMBERS USING MERGE SORT)

```
#include <stdio.h>
#include <conio.h>
void merge(int [],int ,int ,int );
void part(int [],int ,int );
int size;
void main()
{
int i, arr[30];
printf("Enter total no. of elements : ");
scanf("%d", &size);
printf("Enter array elements : ");
for(i=0; i<size; i++)
scanf("%d", &arr[i]);
part(arr, 0, size-1);
printf("\n Merge sorted list : ");
for(i=0; i<size; i++)
printf("%d ",arr[i]);
getch();
}
void part(int arr[], int min, int max)
{
int mid;
if(min < max)
{
mid = (min + max) / 2;
part(arr, min, mid);
part(arr, mid+1, max);
merge(arr, min, mid, max);
}
if (max-min == (size/2)-1)
{
printf("\n Half sorted list : ");
for(i=min; i<=max; i++)
printf("%d ", arr[i]);
}
}
void merge(int arr[],int min,int mid,int max)
{
int tmp[30];
int i, j, k, m;
j = min;
m = mid + 1;
for(i=min; j<=mid && m<=max; i++)
{
if(arr[j] <= arr[m])
{
tmp[i] = arr[j];
j++;
}
else
{
tmp[i] = arr[m];
m++;
}
}
if(j > mid)
{
for(k=m; k<=max; k++)
{
tmp[i] = arr[k];
i++;
}
}
else
{
for(k=j; k<=mid; k++)
{
tmp[i] = arr[k];
i++;
}
}
for(k=min; k<=max; k++)
arr[k] = tmp[k];
}
```

OUTPUT

```
Enter total no. of elements: 8
Enter array elements: 24    13    26    1    2    27    38    15
Half sorted list: 1 13 24 26
Half sorted list: 2 15 27 38
Merge sorted list: 1 2 13 15 24 26 27 38
```

RESULT

Thus array elements were sorted using merge sort's divide and conquer method.

SEARCHING

Searching is the process of locating a particular element in an array. The two searching techniques are:

1. Linear search

2. Binary search

Linear search involves each element to be compared against the key value starting from the first element. Linear search uses a ***“for”*** structure containing an ***“if”*** structure to compare each element of an array with a search key. If search key is not found, then value of –1 is returned. If the array being searched is not in any particular order, then half the elements of an array are likely to compared.

Binary search algorithm locates the middle element and compares with search key. If they are equal, the search key has been found and the subscript of that element is returned. If the search key is less than the middle array element, the first half of the array is searched; otherwise, the second half of the array is searched.

The binary search continues until the search key is equal to the middle element of a sub array or until the sub array consists of one element that is not equal to the search key. After each comparison, the binary search algorithm eliminates half of the elements in the array being searched.

EX. NO. 19

DATE:

SEARCHING AN ELEMENT USING LINEAR SEARCH

AIM

To perform linear search of an element on the given array.

ALGORITHM

1. Start
2. Read number of array elements n
3. Read array elements Ai, i = 0, 1, 2…n–1
4. Read search value
5. Assign 0 to found
6. Check each array element against search
7. If Ai = search then
8. found = 1
9. Print "Element found"
10. Print position i
11. Stop
12. If found = 0 then
13. print "Element not found"

PROGRAM (SEARCHING AN ELEMENT USING LINEAR SEARCH)

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a[50],i, n, val, found;
clrscr();
printf("Enter number of elements : ");
scanf("%d", &n);
printf("Enter Array Elements : \n");
for(i=0; i<n; i++)
scanf("%d", &a[i]);
printf("Enter element to locate : ");
scanf("%d", &val);
found = 0;
for(i=0; i<n; i++)
{
if (a[i] == val)
{
printf("Element found at position %d", i);
found = 1;
break;
}
}
if (found == 0)
printf("\n Element not found");
getch();
}
```

OUTPUT

```
Enter number of elements: 7
Enter Array Elements:
23      6       12      5       0       32      10
Enter element to locate: 5
Element found at position 3
```

RESULT

Thus an array was linearly searched for an element's existence.

EX. NO. 20

DATE:

SEARCHING AN ELEMENT USING BINARY SEARCH

AIM

To locate an element in a sorted array using Binary search method

ALGORITHM

```
1.  Start
2.  Read number of array elements, say n
3.  Create an array arr consisting n sorted elements
4.  Get element, say key to be located
5.  Assign 0 to lower and n to upper
6.  While (lower < upper)
7.  Determine middle element mid = (upper+lower)/2 If key = arr[mid] then
8.  Print mid
9.  Stop
10. Else if key > arr[mid] then
11. lower = mid + 1
12. else
13. upper = mid – 1
14. Print "Element not found"
15. Stop
```

PROGRAM (SEARCHING AN ELEMENT USING BINARY SEARCH)

```
#include <stdio.h>
void main()
{
int a[50],i, n, upper, lower, mid, val, found, att=0;
printf("Enter array size : ");
scanf("%d", &n);
for(i=0; i<n; i++)
a[i] = 2 * i;
printf("\n Elements in Sorted Order \n"); for(i=0; i<n; i++)
printf("%4d", a[i]);
printf("\n Enter element to locate : ");
scanf("%d", &val);
upper = n;
lower = 0;
found = -1;
while (lower <= upper)
{
mid = (upper + lower)/2;
att++;
if (a[mid] == val)
{
printf("Found at index %d in %d attempts", mid, att);
found = 1;
break;
}
else if(a[mid] > val)
upper = mid - 1;
else
lower = mid + 1;
}
if (found == -1)
printf("Element not found");
}
```

OUTPUT

```
Enter array size : 10
Elements in Sorted Order
0 2 4 6 8 10 12 14 16 18
Enter element to locate : 16
Found at index 8 in 2 attempts
```

RESULT

Thus an element is located quickly using binary search method