SOP

&

Lab Manual

for

C PROGRAMMING

DEPARTMENT OF COMPUTER SCIENCE (AIDED)

Supported by

DBT Star College Scheme

Published by

KONGUNADU ARTS AND SCIENCE COLLEGE (Autonomous)

Re-accredited by NAAC with A^+ Grade – 4^{th} Cycle
52^{nd} Rank among Colleges in NIRF 2024
College of Excellence (UGC)
Coimbatore -641 029.
2024-2025

STANDARD OPERATING PROCEDURE

FOR

C PROGRAMMING

PREFACE

Welcome to the C Programming Lab Manual! This manual is designed to accompany your journey into the world of C programming, providing practical guidance, exercises, and insights to reinforce your learning experience.

C programming is a fundamental skill for aspiring programmers, offering a strong foundation in computer science and software development. This lab manual is designed to provide students with hands-on experience in C programming through structured exercises, real-world problem-solving, and practical applications.

The manual covers key concepts such as variables, data types, control structures, functions, arrays, pointers, and file handling, ensuring a comprehensive understanding of the language. Each lab session is structured to reinforce theoretical knowledge through practical implementation, enabling students to develop logical thinking and problem-solving skills.

By following this lab manual, students will gain confidence in writing efficient and optimized C programs. The exercises progressively enhance their coding abilities, preparing them for more advanced topics in software development, embedded systems, and algorithm design.

We hope this manual serves as a valuable resource for students, fostering a deeper understanding of C programming and its applications.

Standard Operating Procedure

Standard Operating Procedures (SOPs) in java programming refer to a set of guidelines and documented processes that outline the best practices, workflows, and steps involved in various aspects of developing java applications and some common components typically covered in SOPs for java programming

1. Requirements Gathering:

- Define the process for gathering and documenting client requirements for the mobile application.
- Specify methods for conducting interviews, surveys, or workshops to identify and prioritize features and functionalities.
- Outline the documentation format for capturing and validating requirements.

2. Design and Wireframing:

- Describe the process for creating the visual design and user interface (UI) of the mobile application.
- Explain how wireframing and prototyping tools are utilized to develop mockups and obtain feedback .
- Define guidelines for ensuring consistency in UI elements, navigation, and usability across different screens and devices.

3. Development Environment Setup:

- Specify the required development tools, software, and platforms for building the mobile application.
- Provide instructions for configuring integrated development environments (IDEs), emulators, and simulators for testing and debugging.
- Outline the process for setting up version control systems and collaborative tools for code management and team communication.

4. Coding Standards and Guidelines:

- Establish coding standards, naming conventions, and best practices for writing clean, maintainable, and efficient code.
- Define guidelines for organizing project files, folders, and code repositories.
- Specify documentation requirements, such as code comments or inline documentation.

5. Development Process:

- Outline the overall development process, such as Agile or waterfall methodologies, and how it applies to mobile app development.
- Define the roles and responsibilities of team members involved in the development process, including developers, testers, and project managers.

6. Testing and Quality Assurance:

- Specify the testing approaches and methodologies to be followed, including unit testing, integration testing, and user acceptance testing.
- Describe the process for identifying and reporting bugs or issues, including bug tracking tools and workflows.
- Outline the criteria for ensuring the quality and performance of the mobile application before release.

7. Deployment and Release:

- Define the process for packaging and deploying the mobile application to app stores (e.g., Apple App Store, Google Play Store).
- Specify the requirements and guidelines for creating app store listings, including app descriptions, screenshots, and promotional materials.
- Explain the process for obtaining necessary certificates, signing the app, and handling updates or version releases.

8. Maintenance and Support:

- Provide guidelines for post-release maintenance and support, including bug fixes, performance optimizations, and feature enhancements.
- Outline the process for gathering user feedback, conducting user surveys, and incorporating user suggestions into future updates.
- Describe the mechanisms for monitoring and analyzing the app's performance, crash reports, and analytics data.

These guidelines can serve as a starting point, but it's recommended to adapt and customize them to align with your organization's unique requirements and practices in C programming.

LIST OF PRACTICALS

C PROGRAMMING LAB EXPERIMENTS

1.SUM, AVERAGE, STANDARD DEVIATION

AIM:

To write a program to find the sum, average, and standard deviation for a given set of numbers.

SD=$\sqrt{1/n}\sum(x-\bar{x})^2$

ALGORITHM:

STEP 1: Start the program.
STEP 2: Get the number of inputs.
STEP 3: Get the values.
STEP 4: Find the sum for the given values.
STEP 5: Find the average using the formula:

$$\text{Average (ave)} = \frac{\sum x}{n}.$$

STEP 6: Find the standard deviation (SD) using the formula:

$$SD = \sqrt{1/n}\sum(x-\bar{x})^2$$

STEP 7: Display the sum, average, and standard deviation.
STEP 8: Terminate the program.

CODING

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

void main()
{
   clrscr();

   float x[100];
   float sum = 0, avg, tot = 0, var, sd;
   int n, i;

   printf("Enter the size of your array: ");
   scanf("%d", &n);

   for (i = 0; i < n; i++)
   {
      scanf("%f", &x[i]);
   }

   for (i = 0; i < n; i++)
   {
      sum = sum + x[i];
   }

   avg = sum / n;
```

```
    for (i = 0; i < n; i++)
    {
        tot = tot + pow((x[i] - avg), 2);
    }

    var = tot / (float)n;
    sd = sqrt(var);

    printf("\nSum = %f", sum);
    printf("\nAverage = %f", avg);
    printf("\nVariance = %f", var);
    printf("\nStandard Deviation = %f", sd);
}
```

OUTPUT:

Enter the size of your array:3

Enter the elements one after the other.....2

2

3

Sum = 7.000

Variance 0.22222

Standard deviation:0.471405

2. PRIME NUMBERS

AIM:

A program to generate 'n' prime numbers.

ALGORITHM:

STEP 1: Start the program.

STEP 2: **I**nclude the necessary header files.

STEP 3: Declare the variables.

STEP 4: Get the value of 'n' from the user.

STEP 5: Check whether a number is divisible by any other number up to the given number.

STEP 6: Increment the count if the number is not divisible by any other number.

STEP 7: Terminate the program.

CODING

```
#include <stdio.h>
#include<conio.h>
void main()
{
int i,j,n,r;
char prime;
clrscr();
printf("Enter the last number :");
scanf("%d",&n);
for(i=2;i<=n;i++)
{
prime = prime ‘y' ;
{
for (j = 2; j < i; j + +)
{
r = i% j
If ( r ==0)
{
prime =’ prime n';
 break;
}
}
If (prime==prime ‘y' )
printf("%d\n",i);
}
getch();}
```

OUTPUT:

Enter the last number : 7

2

3

5

7

3. BIGGEST NUMBER

AIM:

To write a program to find the biggest number among a set of numbers.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare i, n, big, and x[10] as integers.

STEP 3: Get the number of elements (n) from the user.

STEP 4: Input the numbers one by one into the array x[i].

STEP 5: Compare each number with big. If big < x[i], assign x[i] to big until the condition is false.

STEP 6: Stop the program.

CODING

```
#include<stdio.h>
#include<conio.h>
void main ()
{
 int i, n, big, x[10] ;
clrscr();
printf("Number of data :");
scanf ("%d",&n);
printf("\n Enter numbers one by one : “);
for ( i = 0; i < n ;i++)
scanf("%d", &x[i]);
big = x[0] ;
for ( i = 0; i < n ;i++)
{
If(big < x[i])
{
big =x[i];
}
}
printf("The biggest = %d ",big);
getch();}
```

OUTPUT:

Number of data: 5

Enter the numbers one by one:

2

4

6

3

7

The Biggest =7

4. BUBBLE SORTING

AIM:

A program to arrange a set of numbers in ascending order.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Include the necessary header files.

STEP 3: Declare the variables.

STEP 4: Get the values from the user and print the entered values.

STEP 5: Compare each number with the next number.

STEP 6: Interchange the numbers if the current number is greater than the next number.

STEP 7: Display the arranged order.

STEP 8: Terminate the program.

CODING

```
#include<stdio.h>
#include<conio.h>
void main ()
{
int x [20], i, j, t, n;
clrscr();
printf ("Enter no of elements:");
scanf("%d",&n);
for ( i = 0 i<n; i++)
{
printf ("Data :");
Scanf ("%d", &x [i]);
}
for( i = 0; i < n ;i++)
{
for( j = i + 1; j < n ;j++)
{
If(x[i] > x[j])
{
T=x[i];
X[i]=x[j];
}
}
printf ("Numbers in Ascending Order:\n”);
 for ( i = 0; i < n ;i++)
```

```
{
printf ("%d\n", x[i];
} getch ();}
```

OUTPUT:

Enter the number of elements :4

Data: 2

Data: 4

Data: 3

Data: 9

Numbers in ascending order

2

3

4

9

5. MERGING

AIM:

A program to merge two arrays.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Include the necessary header files.

STEP 3: Declare the variables as global.

STEP 4: Get the values for the first array and the second array.

STEP 5: Call the merge() function.

STEP 6: Assign the elements of the first array to the third array.

STEP 7: Assign the elements of the second array to the third array starting at the end of the first array's values.

STEP 8: Print the third array.

STEP 9: Terminate the program.

CODING

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

void main()
{
   int a1[10], a2[10], a3[20];
   int i, n1, n2, e1 = 0, e2 = 0, e3 = 0;

   clrscr();

   printf("Number of elements in Array1 and Array2: ");
   scanf("%d %d", &n1, &n2);

   printf("Enter data for Array1:\n");
   for (i = 0; i < n1; i++)
      scanf("%d", &a1[i]);

   printf("Enter data for Array2:\n");
   for (i = 0; i < n2; i++)
      scanf("%d", &a2[i]);

   int n = n1 + n2;
while (e1 < n1 && e2 < n2)
   {
      if (a1[e1] < a2[e2])
      {
```

```
            a3[e3++] = a1[e1++];
        }
        else
        {
            a3[e3++] = a2[e2++];
        }
    }

while (e1 < n1)
    {
        a3[e3++] = a1[e1++];
    }
    while (e2 < n2)
    {
        a3[e3++] = a2[e2++];
    }

    printf("\nMerged Array:\n");
    for (i = 0; i < n; i++)
        printf("%4d", a3[i]);

    getch();
}
```

OUTPUT

Number of elements in Array1 and Array2: 3 4

Enter data for Array1:

1 5 9

Enter data for Array2:

2 6 8 10

Merged Array:

1 2 5 6 8 9 10

6. SIN VALUE

AIM:

To write a program to calculate a sine value and compare it with the built-in functions.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare variables n, P, j, and fact.

STEP 3: Input values for x (in degrees) and n (number of terms).

STEP 4: Convert x to radians using the formula (x * 3.14) / 180.

STEP 5: Initialize S = 1 and sum = 0.

STEP 6: Use a for loop to generate the value of fact (factorial).

STEP 7: Display the output.

STEP 8: Terminate the program.

CODING

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
Void main()
{
int x deg,n,s=1,I,j;
 float x rad, sum =0;
long int fact;
clrscr();
printf ("\n Enter angle in degree:");
scanf("%d", &xdeg);
 xrad= ((3.14/180)*xdeg);
printf ("\n Upto which power of x :");
scanf("%d",&n);
for (i=1;i<n;i++)
{
fact=1;
for (j=0;j=i;j++)
{
fact = fact*j;
}
}
Sum = sum+((s*pow(xrad,i)/fact));
S=-s;
printf("\n result = %f",sum);
printf(\n Library value = %f", sin (xrad));
```

```
getch();
}
```

OUTPUT:

Enter angle in degree 180

Upto which power of x: 2

Result = 9.859601

Library value =0.001593

7. NCR VALUES

AIM:

To write a recursive program to calculate factorial values and compute the NCR value.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the function fact() as long.

STEP 3: Declare F1, F2, F3, and F4 as long, and n as an integer.

STEP 4: Input the value of n.

STEP 5: In the recursive function, check the condition if n == 1. If true, return 1.

STEP 6: If the condition is false, calculate and return the value as fact(n) = n * fact(n - 1).

STEP 7: Terminate the program.

CODING

```
#include<stdio.h>
#include<conio.h>
int fact(int k);
void main()
{
Int n, r, ncr, nf, nrf,rf;
printf(“ Enter n&r:”);
scanf(“%d %d”, &n &r);
If(r>n)
{
printf(“Invalid input: r should not be greater than n.\n”);
return;
}
nf=fact(n);
nrf=fact(n-r);
rf=fact®;
ncr=nf/(nrf*rf);
printf(“\n nCr value=%d\n”, nCr);
getch();
}
Int fact(int k)
{
If (k==0||k==1)
{
Return 1;
}
```

```
Else
{
Return k*fact(k-1); }}
```

OUTPUT:

Enter n&r:5 2

ncr value =10

8. PALINDROME

AIM:

To check if the given string is a palindrome.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Include the necessary header files.

STEP 3: Declare the variables.

STEP 4: Get the string input using the gets() function.

STEP 5: Find the length of the string using the strlen() function.

STEP 6: Check for blank spaces or null characters in the string.

STEP 7: Copy the string to another variable.

STEP 8: Compare the original string with the reversed string to check if they are equal.

STEP 9: Display whether the string is a palindrome or not.

STEP 10: Terminate the program.

CODING

```
#include <stdio.h>
#include<conio.h>
#include<string.h>
void main ()
{
int i, l, count =0, j = 0;
char txt[50], word [50], rword[50];
clrscr();
printf("Enter the string :");
flushall ();
gets (txt);
l = strlen(txt);
for ( i = 0 ; i <= l i++)
{
if (txt [i]!=" && txt [i]! = ‘0’)
{
word [j] = txt[i]
j++;
}
else
{
word [j]=’\0’;
printf ("%s\n", word );
strcpy (rword, word );
strrev (rword );
if (strcmp (word, rword )==0)
```

```
{
count++;0
}
j=0;
}
}
Printf ("Number of palindromes =%d", count);
getch ();
}
```

OUTPUT:

Enter the string :mom called me

Numbers of palindrome =1

9. STRING OPERATIONS

AIM:

To copy, concatenate, reverse, and find the length of a string.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Include the necessary header files.

STEP 3: Declare the variables.

STEP 4: Print the menu and ask the user to select an option.

STEP 5: Perform the selected operation: copy, concatenate, reverse, or find the length of the string.

STEP 6: Use the strcpy() function to copy the string.

STEP 7: Use the strcat() function to concatenate strings.

STEP 8: Use the strlen() function to find the length of the string and strrev() function to reverse the string.

STEP 9: Terminate the program.

CODING

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main ()
{
int n;
char s1[100], s2[100], s3[100], choice;
clrscr();
printf ("1. Reversing string.\n");
printf ("2. String concatenation.\n");
printf("3. Coping string.\n");
printf ("4. Exit.\n");
printf ("Enter your choice:");
scanf ("%d", & choice);
switch(choice)
{
case1:
printf("Enter the string to reverse:");
scanf ("%s", s1);
printf("Given string is %s\n",s1)"
strrev (s1);
printf("Reersed string is %s", s1);
break;
case2:
printf("Enter the 1st string:");
scanf("%s", s1);
strlwr(s1);
```

```
printf("\n Enter the 2nd string:");

scanf("%s",s2);
strcat(s1,s2);
printf ("\n concatenation string:%s", s1);
break;
case3:
printf("Enter string to copied :");
scanf("%s",s1);
strcpy(s2,s1);
printf("Given string is:%s\n",s1);
printf("\n copied string is %s",s2);
break;
case4:
break;
default:
printf ("Invalid choice");
}
getch();
}
```

OUTPUT:

1. Reversing the String
2. String concatenation.
3.Coping string
4. Exit.

Enter your choice 1

Enter the string to reverse: Book

Given string is book

Reversed string is koob

1. Reversing the String
2. String concatenation.
3. Coping string
4. Exit.

Enter your choice :2

Enter the 1stString: note

Enter the 2ndstring: book

The concatenated string: note book

1. Reversing string.
2. String concatenation.
3. Coping string.
4. Exit.

Enter your choice: 3

Enter the string to copied :book

Given string is book

Copied string: book

1.Reversing String.

2.String concatenation.

3.Coping sting.

4.Exit.

Enter your choice: 6

Invalid choice

10. LINEAR SEARCH

AIM:

To implement the linear search to find a particular name in a list of names.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Include the necessary header files.

STEP 3: Declare the variables.

STEP 4: Display the menu.

STEP 5: Perform a linear search if ch == 1.

STEP 6: If i == n, display "Not Found".

STEP 7: Repeat the loop until the search is complete.

STEP 8: Terminate the program.

CODING

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main ()
{
char name [10] [20];
int i, n, d;
charsrch[20];
printf ("\n Enter the number of names:");
scanf("%d", &n);
printf ("\n Enter the name one by one:");
for(i=0;i<n;i=i+1)
{
scanf("%s", name[i]);
}
printf("Enter the name to search:");
scanf ("%s", srch); For(i=0;i<n;i=i+1)
{
d = strcmp(name[i],srch);
if(d==0)
{
printf("\n Name Found at position %d,i);
}
}
getch();
}
```

OUTPUT:

Sorted names:

Dhiyana

Vino

Ram

Menu

1.linear sort

2. Exit

Enter your choice :1

Enter the names to be searched: ram

Name found at 3 positions

Sorted name

Dhiya

Ram

Vino

Menu

1. Linear sort

2. Exit

Enter your choice : 2

11. STACK OPERATION

AIM:

To write a program with functions for the following stack operations:

1. Push
2. Pop
3. Display

ALGORITHM:

STEP 1: Start the program.

STEP 2: Include the necessary header files.

STEP 3: Declare the variables.

STEP 4: Display the menu with the stack operations.

STEP 5: Perform the selected operation:

If ch == 1, perform the Push operation to add an element to the stack.

If ch == 2, perform the Pop operation to remove the top element from the stack.

If ch == 3, perform the Display operation to show all elements in the stack.

STEP 6: Check if the stack is empty or full during operations and display appropriate messages.

STEP 7: Repeat the loop until the user decides to exit.

STEP 8: Terminate the program.

CODING

```
#include <stdio.h>
#include <conio.h>
int st [10], top=-1;
void push (int data)
{
top++;
st [top] = data;
}
int pop();
{
int item;
item = st[ top ];
top--;
return item;
}
Void display ()
{
int i;
printf ("Items in stack:");
for (i= top; i>-1; i--)
printf ("\n %d", st [i]);
}
void main ()
{
Int num, choice = 0;
```

```
while (choice <4)
{
clrscr();
printf("\n MENU ");
printf("\n -------"):
printf ("\n 1. PUSH");
printf ("\n 2. POP");
printf ("\n3. DISPLAY ALL ");
printf ("\n 4. EXIT\n");
printf ("\n YOUR CHOICE: ");
scanf("%d", &choice);
// printf("Press any key....");
getch();
switch (choice)
{
case1:
if (top>=9)
{
printf ("\n STACK IS FULL");
}
else
{
printf ("\n Enter number to be pushed :");
scanf("%d",&num);
push(num);
}
break;
```

```
case2;
if (top<0)
{
printf("\n STACK IS EMPTY");
}
else
{
num=pop();
printf ("\n The value %d is poped", num");
}
break;
case 3:
If (top<0)
{
printf("\n STACK IS EMPTY");
}
else
{
display ();
}
break;
case 4:
break;
}
getch();
}
}
```

OUTPUT:

Menu

****

1.push

2.pop

3. display all

4.exit

Enter your choice :1

Enter the nmber to be pushed: 6

1.push

2.pop

3.display all

4.exit

Enter your choice2

The value of 6 is poped

Menu

……..

1.push

2.pop

3.display all

4.exit

Enter your choice :3

Stack is empty

Menu

****

1.push

2.pop

3.display all

4.exit

Enter your choice 3

Exiting

12. MATRIX ADDITION

AIM:

To write a program to perform matrix addition.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Include the necessary header files.

STEP 3: Declare the variables.

STEP 4: Use a for loop to perform the matrix addition.

STEP 5: Use a for loop to print the resulting matrix.

STEP 6: Terminate the program.

CODING

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

void main() {
   int i, j, r, c, a[10][10], b[10][10], x[10][10];
   clrscr();

   printf("Enter order of matrix (rows and columns): ");
   scanf("%d %d", &r, &c);

   printf("Data for Matrix A:\n");
   for (i = 0; i < r; i++) {
      for (j = 0; j < c; j++) {
         scanf("%d", &a[i][j]);
      }
   }

   printf("Data for Matrix B:\n");
   for (i = 0; i < r; i++) {
      for (j = 0; j < c; j++) {
         scanf("%d", &b[i][j]);
}
   }

   for (i = 0; i < r; i++) {
      for (j = 0; j < c; j++) {
```

```
            x[i][j] = a[i][j] + b[i][j];
        }
    }

    printf("The resultant matrix is:\n");
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            printf("%7d", x[i][j]);
        }
        printf("\n");
    }

    getch();
}
```

OUTPUT:

Enter order of matrix (rows and columns): 2 2

Data for Matrix A:

1 2

3 4

Data for Matrix B:

5 6

7 8

The resultant matrix is:

6 8

10 12

13. UNIVERSITY MARKLIST

AIM:

To write a program to print the marksheet for the semester of the university.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Get the name and roll number of the student.

STEP 3: Get the marks in subjects such as Tamil, English, and Moral Education.

STEP 4: Check the marks for pass/fail status and display the result.

STEP 5: Terminate the program.

CODING

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

struct Student {
    char regno[10];
    char name[20];
    int marks[3];
};

void printline(int n);

int main() {
    Student s[10];
    int n, i;
    char result[10];

    cout << "Number of students: ";
    cin >> n;

    for (i = 0; i < n; i++) {
        cout << "Register no: ";
        cin >> s[i].regno;
        cout << "Name: ";
        cin >> s[i].name;
        cout << "Enter 3 subject marks: ";
```

```
        cin >> s[i].marks[0] >> s[i].marks[1] >> s[i].marks[2];
    }

    for(i = 0; i < n; i++) {
        cout << "\n BHARATHIAR UNIVERSITY, Coimbatore ";
        cout << "\n Kongunadu Arts and Science College, Coimbatore-641029\n";
        printline(80);
        cout << "\n Register no: " << s[i].regno << "\t\t Name: " << s[i].name;
        cout << "\nSubject\t\t\tMax Marks\tMarks Awarded\tResult\n";
        printline(80);

        // English
        if (s[i].marks[0] >= 40)
            strcpy(result, "PASS");
        else
            strcpy(result, "FAIL");
        cout << "\n English\t\t100\t\t" << s[i].marks[0] << "\t\t" << result;

        if (s[i].marks[1] >= 40)
            strcpy(result, "PASS");
        else
            strcpy(result, "FAIL");
        cout << "\n Mathematics\t\t100\t\t" << s[i].marks[1] << "\t\t" << result;
        if (s[i].marks[2] >= 40)
strcpy(result, "PASS");
        else
            strcpy(result, "FAIL");
```

```
        cout << "\n C Programming\t\t100\t\t" << s[i].marks[2] << "\t\t" << result;

        printline(80);
        cout << "\nPress any key to continue...\n";
        cin.ignore();
        cin.get();
    }
 return 0;
}

void printline(int n) {
    for (int i = 0; i < n; i++) {
        cout << "*";
    }
    cout << "\n";
}
```

OUTPUT:

```
Number of students: 2

 BHARATHIAR UNIVERSITY, Coimbatore
 Kongunadu Arts and Science College, Coimbatore-641029
****************************
 Register no: 101              Name: Alice
Subject          Max Marks      Marks Awarded   Result
****************************
 English              100          85                  PASS
 Mathematics      100          76                  PASS
 C Programming   100          90                  PASS
****************************
Press any key to continue...
```

14. ARRAY USING POINTERS

AIM:

To write a program to display the array using pointers.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the variables.

STEP 3: Get the number of elements (data) from the user.

STEP 4: Assign the array to the pointer variable.

STEP 5: Display all numbers using the pointer variable.

STEP 6: Terminate the program.

CODING

```
#include <stdio.h>
#include <conio.h>
void main ()
{
int i, n;
float x[10], *ptr;
clrscr();
printf ("How many numbers: ");
scanf ("%d", &n);
printf ("Enter the data one by one\n");
for (i=0; i<n; i++)
{
scanf ("%f",&x [i]);
}
ptr = &x [0];
printf ("The numbers are: \n");
for (i=0; i<n; i++)
{
printf ("%f\n", *ptr);
ptr++;
}
getch();
}
```

OUTPUT:

How many numbers: 4

Enter the data one by one

3

4

5

1

The numbers are:

3.000000

4.000000

5.000000

1.000000

15. FILE CONCEPT

AIM:

To write a program to display the contents of an array using pointers.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the variables.

STEP 3: Get the number of elements (data) from the user.

STEP 4: Assign the array to the pointer variable.

STEP 5: Display all numbers using the pointer variable.

STEP 6: Terminate the program.

CODING

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

int main(int argc, char *argv[])
{
 FILE *f1, *f2;
 char ch;
 if (argc < 3)
{
 printf("Insufficient arguments\n");
 exit(1); // Exit with error code
  }
  f1 = fopen(argv[1], "r");
  if (f1 == NULL) { // Check if file exists
  printf("File: %s not found\n", argv[1]);
  exit(1);
  }
  f2 = fopen(argv[2], "w");
  if (f2 == NULL) {
  printf("Cannot create file: %s\n", argv[2]);
  fclose(f1);
  exit(1);
  }
  while ((ch = getc(f1)) != EOF)
 {
```

```
putc(ch, f2);
}
 fclose(f1);
fclose(f2); // Close target file
printf("Process Completed.....\n");
return 0;
}
```

OUTPUT:

Hello, World!

Welcome to File Handling in C.

Process Completed.....

Hello, World!

Welcome to File Handling in C

16. MONTH BY MONTH CALENDER

AIM:

To write a C program to display the calendar for each month of a given year.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Prompt the user to input a year.

STEP 3: Validate the input: If the year is less than 0, print "Invalid year" and exit.

STEP 4: Determine if the year is a leap year and adjust February's days accordingly.

STEP 5: For each month, calculate the starting day and print the month's name and layout (Sun-Sat), along with the days.

STEP 6: Print the days of each month in a calendar format, and repeat for all 12 months.

STEP 7: Terminate the program.

CODING

```
#include<stdio.h>
#include<stdbool.h>
Bool is leap Year (int year)
{
return (year 0/0 * 4 ==0 && year \% 100! = 0 ||year 0/0 * 400 ==0)
}
void display calendar (int year)
{
const char*months []= { "January", "February", "March", "April", "May",
"June", " July", "Au- gust" "September", "October", "November", "December"
}
int days in Months [] = {31,28,31,30,31,30,31,31,30,31,30,31};
 if(is Leap Year (year))
{
days in Months [1]=29
}
printf ("\n Calender for year %d\n", year);
for (int month =0; month < 12 ++month)
{
printf ("\n \t%s:\n", months [month]);
printf("Sun Mon Tue Wed Thu Fri Sat \n");
int starting Day = year +(year-1)/4- (year-1)/100 + (year-1)/400;
for (int i = 0 i<month;++i)
{
starting Day += days in Month [i];
}
starting day 0/0 = 7
```

```
for (int i = 0 <starting Day ;++i)
{
printf(" ");
}
for (int day = 1; day<=days in month [month]; ++day
{
printf ("%4d", day);
 if ((starting Day +day)%7==0|| day == days in month [month]
{
printf("\n");
}
}
}
}
int main ()
{
int year;
printf ("Enter a year to display the calendar (e.g.2023):");
scanf ("%d",&year);
if year<0)
{
printf (" Invalid Year !\n");
return 1;
}
if (year <100)
{
year += 2000;
```

```
}
if (year <1582 || year>4902)
{
printf ("Invalidyear !\n");
return 1;
}
display calendar (year);

return 0;
}
```

OUTPUT:

Enter a year to display the calendar : 2024

Calendar for year 2024

January:

Sun mon Tue Wed Thu Fri Sat

123456

7 8 9 10 11 12 13

14 15 1 17 18 19 20

21 22 23 24 25 26 27

28 29 30

February

Sun Mon Tue Wed Thu Fri Sat

123

45678910

11 12 13 14 15 16 17

18 19 20 21 22 23 24

25 26 27 28

March

Sun Mon Tue Wed Thu Fri Sat

12

345789

10 11 12 13 141516

17 18 19 20 21 22 23

24 25 2 27 28 29 30

31

April

Sun Mon Tue Wed Thu Fri Sat

123456

7 8 9 10 11 12 13

14 15 17 18 19 20

21 22 23 24 25 26 27 28

29 30

May

Sun Mon Tue Wed Thu Fri Sat

1234

5 7 8 9 10 11

12 13 14 15 1 17 18

19 20 21 22 23 24 25

26 27 28 29 30 31

June

Sun Mon Tue Wed Thu Fri Sat

1234567

8 9 10 11 12 13 14

15 16 17 18 19 20 21

22 23 24 25 26 27 28

29 30

July

Sun Mon Tue Wed Thu Fri Sat

12345

6 7 8 9 10 11 12

13 14 15 16 17 1819

20 21 22 23 24 25 26

27 28 29 30 31

August

Sun Mon Tue Wed Thu Fri Sat

123

45 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

September

Sun mon Tue Wed Thu Fri Sat

123457

8 9 10 11 12 13 14

15 16 17 18 19 20 21

22 23 24 25 26 27 28

29 30

October:

Sun mon Tue Wed Thu Fri Sat

12345

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

November

Sun Mon Tue Wed Thu Fri Sat

12

3456789

10 11 12 13 14

15 16 17 18 19 20 21

22 23 24 25 26 27

28 29 30

December

Sun Mon Tue Wed Thu Fri

1234567

8 9 10 11 12 13 14

15 16 17 18 19 20 21

22 23 24 25 26 27 28

29 30 31

17. READ/WRITE STRUCTURE TO A FILE

AIM:

To write a C program to read from and write to a file using structures.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Prompt the user to enter the filename to open for reading and store it.

STEP 3: Open the file for reading. If it cannot be opened, display an error and exit.

STEP 4: Prompt the user to enter the filename to open for writing and store it.

STEP 5: Open the file for writing. If it cannot be opened, display an error and exit.

STEP 6: Read the content from the first file character by character and write it to the second file until the end of the file is reached.

STEP 7: Terminate the program.

CODING

```
#include<stdio.h>
#include<stdlib.h>
int main ();
{
file*fptr1,*fptr2;
char filename[100]
int c;
printf("Enter the filename to open for reading :");
scanf("%s", filename);
fprt1 = fopen(filename, "r");
if (fprt1==NULL);
{
printf("Cannot open file %s\n", filename);
exit(1);
}
printf("Enter the filename to open for writing :");
scanf("%s", filename);
fprt2=fopen(filename, "w");
If(fprt2==NULL)
{
 printf("Cannot open file %s\n", filename”);
exit(1);
}
while(((c = fgetc(fprt1))!=EOF)
{
fputc(c,fptr2);
```

```
}
printf("Contents copied to %s\n",filename);
fclose (fptr1);
fclose(fptr2);
return 0;
}
```

OUTPUT:

Enter the filename to open for reading

a.txt

Enter the file name to open for writing

b.txt

18. PALINDROME USING POINTER

AIM:

To write a C program to check if a string is a palindrome using pointers.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Define a function to check if a string is a palindrome using pointers.

STEP 3: Set two pointers: one (ptr) starting at the beginning of the string, and another (rev) for reversing the string.

STEP 4: Move the first pointer (ptr) to the end of the string.

STEP 5: Move both pointers (ptr and rev) towards each other, comparing the characters at their positions.

STEP 6: If the characters match, continue moving the pointers closer to each other. If they don't match, stop.

STEP 7: If both pointers meet or cross, the string is a palindrome; otherwise, it is not.

STEP 8: Print whether the string is a palindrome or not based on the comparsion.

STEP 9: Terminate the program.

CODING

```
#include<stdio.h>
void is palindrome (char*string)
{
char*ptr,*rev;
ptr = string;
while (*ptr!='\0')
{
++ptr;
}
--ptr;

for (rev = string;ptr>=rev;)
{
If (*ptr==*rev)
{
--ptr;
rev++;
}
else
break;
}
If (rev >ptr)
printf("String is palindrome ");
else
printf ("String is not palindrome");
}
```

```
int main ()
{

charstr [1000] = "madam",is palindrome (str);

return0;

}
```

OUTPUT:

Input :str = "Madam"

Output: String is not palindrome

Inpt :str = "madam"

Output: String is palindrome

Input :str = "radar"

Output: String is palindrome.