CENTRAL CALCUTTA POLYTECHNIC

21, Convent Road, Philips, Sealdah, Kolkata, West Bengal 700014

DEPT. : COMPUTER SCIENCE AND TECHNOLOGY

- NAME : SUMAN MONDAL
- ROLL : DCCPCSTS6
- NUMBER : 10005537
- REG NUMBER : D192005242
- SUBJECT : SYSTEM PROGRAMMING AND COMPILER DESIGN
- SESSION : 2021 - 2022
- EMAIL : SUMAN.MONDAL@OUTLOOK.IN

Contents

Chapter 1

Compiler Design Programs

1.1 Implement Linear Search using C language

Source Code :

```c
//Implement Linear Search using C language

#include <stdio.h>
#include <stdlib.h>
#define size 20

int main (int argc, char *argv[]) {
    int arr[size], num, i, n, found = 0, pos = -1;
    printf ("\n Enter the number of elements in the array: ");
    scanf ("%d", &n);
    printf ("\n Enter the elements: ");

    for (i = 0; i < n; i++)
        scanf ("%d", &arr[i]);

    printf ("\n Enter the number that has to be searched: ");
    scanf ("%d", &num);

    for (i = 0; i < n; i++) {
        if (arr[i] == num) {
            found = 1;
            pos = i;
            printf ("\n %d is found in the array at position = %d",
             ↪  num, i+1);
            break;
        }
    }
    printf("\n");
    if (found == 0)
```

```
        printf ("\n %d does not exist in the array", num);
    return 0;

}
```

Program Output :

```
➜ gcc Q01.c && ./a.out

 Enter the number of elements in the array: 6

 Enter the elements: 12 69 87 2 45 96

 Enter the number that has to be searched: 69

 69 is found in the array at position = 2
```

1.2 Implement Binary Search using C language

Source Code :

```
//Implement Binary Search using C language

#include <stdio.h>
int binarySearch(int arr[], int l, int r, int x)
{
        if (r >= l) {
                int mid = l + (r - l) / 2;
                if (arr[mid] == x)
                        return mid;

                if (arr[mid] > x)
                        return binarySearch(arr, l, mid - 1, x);

                return binarySearch(arr, mid + 1, r, x);
        }

        return -1;
}

int main(void)
{
        int arr[] = { 2, 3, 4, 10, 40 };
```

```
    printf ("Array is: ");
    for (int i = 0; i < 5; i++)
        printf ("%d ", arr[i]);
    printf ("\n");
        int n = sizeof(arr) / sizeof(arr[0]);
        int x = 10;

    printf ("Element to be searched: 10\n");
        int result = binarySearch(arr, 0, n - 1, x);
        (result == -1)
                ? printf("Element is not present in array")
                : printf("Element is present at position %d",
                ↪  result+1);
    printf ("\n");
        return 0;
}
```

Program Output :

```
→ gcc Q02.c && ./a.out
Array is: 2 3 4 10 40
Element to be searched: 10
Element is present at position 4
```

1.3 Implement Insertion Sort using C language

Source Code :

```
//Implement Insertion Sort using C language

#include <stdio.h>

void printArray(int array[], int size) {
  for (int i = 0; i < size; i++) {
    printf("%d ", array[i]);
  }
  printf("\n");
}
```

```c
void insertionSort(int array[], int size) {
  for (int step = 1; step < size; step++) {
    int key = array[step];
    int j = step - 1;
    while (key < array[j] && j >= 0) {
      array[j + 1] = array[j];
      --j;
    }
    array[j + 1] = key;
  }
}

int main() {
  int data[] = {9, 5, 1, 4, 3};
  printf ("Array before Insertion Sort: ");
  for (int i = 0; i < 5; i++)
    printf ("%d ", data[i]);
  printf ("\n");

  int size = sizeof(data) / sizeof(data[0]);
  insertionSort(data, size);
  printf("Sorted array in ascending order:\n");
  printArray(data, size);
}
```

Program Output :

```
→ gcc Q03.c && ./a.out
Array before Insertion Sort: 9 5 1
Sorted array in ascending order:
1 3 4 5 9
```

1.4 Implement Shell Sort using C language

Source Code :

```c
//Implement Shell Sort using C language
```

```c
#include <stdio.h>

void shellSort(int array[], int n) {
  for (int interval = n / 2; interval > 0; interval /= 2) {
    for (int i = interval; i < n; i += 1) {
      int temp = array[i];
      int j;
      for (j = i; j >= interval && array[j - interval] > temp; j -=
       ↪  interval) {
        array[j] = array[j - interval];
      }
      array[j] = temp;
    }
  }
}

void printArray(int array[], int size) {
  for (int i = 0; i < size; ++i) {
    printf("%d  ", array[i]);
  }
  printf("\n");
}

int main() {
    int data[] = {9, 8, 3, 7, 5, 6, 4, 1};
    printf ("Array before Shell Sort: ");
    for (int i = 0; i < 8; i++)
        printf ("%d ", data[i]);
    printf ("\n");

    int size = sizeof(data) / sizeof(data[0]);
    shellSort(data, size);
    printf("Sorted array: \n");
    printArray(data, size);
}
```

Program Output :

```
→ gcc Q04.c && ./a.out
Array before Shell Sort: 9 8 3 7 5 6
Sorted array:
1  3  4  5  6  7  8  9
```

1.5 Write a program to check balance parenthesis of a program.[a+(b*c) Balanced parenthesis ,Using Stack]

Source Code :

```c
//Write a program to check balance parenthesis of a program.
//[ a+(b*c) Balanced parenthesis ,Using Stack]

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

#define MAX 30
int top=-1;
int stack[MAX];

void push(char);
char pop();
int match(char a,char b);
int check(char []);

int main()
{
        char exp[MAX];
        int valid;
        printf("Enter an algebraic expression : ");
        gets(exp);
        valid=check(exp);
        if(valid==1)
                printf("Valid expression\n");
        else
                printf("Invalid expression\n");
```

```
                return 0;

}
int check(char exp[] )
{
        int i;
        char temp;
        for(i=0;i<strlen(exp);i++)
        {
                if(exp[i]=='(' || exp[i]=='{' || exp[i]=='[')
                        push(exp[i]);
                if(exp[i]==')' || exp[i]=='}' || exp[i]==']')
                        if(top==-1)     /*stack empty*/
                        {
                                printf("Right parentheses are more
                                 ↪  than left parentheses\n");
                                return 0;
                        }
                        else
                        {
                                temp=pop();
                                if(!match(temp, exp[i]))
                                {
                                        printf("Mismatched
                                         ↪  parentheses are : ");
                                        printf("%c and
                                         ↪  %c\n",temp,exp[i]);
                                        return 0;
                                }
                        }
        }
        if(top==-1) /*stack empty*/
        {
                printf("Balanced Parentheses\n");
                return 1;
        }
        else
        {
                printf("Left parentheses more than right
                 ↪  parentheses\n");
                return 0;
        }
```

```
}/*End of main()*/
int match(char a,char b)
{
        if(a=='[' && b==']')
                return 1;
        if(a=='{' && b=='}')
                return 1;
        if(a=='(' && b==')')
                return 1;
        return 0;
}/*End of match()*/

void push(char item)
{
        if(top==(MAX-1))
        {
                printf("Stack Overflow\n");
                return;
        }
        top=top+1;
        stack[top]=item;
}/*End of push()*/

char pop()
{
        if(top==-1)
        {
                printf("Stack Underflow\n");
                exit(1);
        }
        return(stack[top--]);
}/*End of pop()*/
```

Program Output :

```
Enter an algebraic expression : (
Balanced Parentheses
Valid expression
```