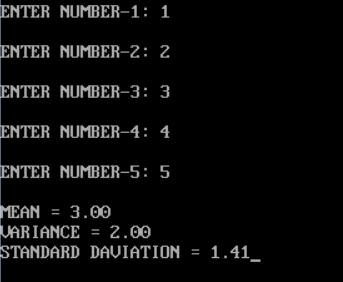
Q1: Array Operations

SOLUTION:

CODE:

1. #include <stdio.h>
2. #include <conio.h>
3. #include <math.h>
4. void main()
5. {
6. int a[10],i,n=5;
7. float mean,temp,var,sd,sum=0.0;
8. clrscr();
9. for(i=0;i<n;i++)
10. {
11. printf("\nENTER NUMBER-%d: ",i+1);
12. scanf("%d",&a[i]);
13. sum=sum+a[i];
14. }
15. mean=sum/n;
16. sum=0.0;
17. for(i=0;i<n;i++)
18. {
19. temp=a[i]-mean;
20. sum=sum+(temp\*temp);
21. }
22. var=sum/n;
23. sd=sqrt(var);
24. printf("\nMEAN = %0.2f",mean);
25. printf("\nVARIANCE = %0.2f",var);
26. printf("\nSTANDARD DAVIATION = %0.2f",sd);
27. getch();
28. }

O/P:

  
Q2: Linear Search

SOLUTION:

CODE:

#include <stdio.h>

int main()

{

int array[100], search, c, n;

printf("Enter number of elements in array\n");

scanf("%d", &n);

printf("Enter %d integer(s)\n", n);

for (c = 0; c < n; c++)

scanf("%d", &array[c]);

printf("Enter a number to search\n");

scanf("%d", &search);

for (c = 0; c < n; c++)

{

if (array[c] == search) /\* If required element is found \*/

{

printf("%d is present at location %d.\n", search, c+1);

break;

}

}

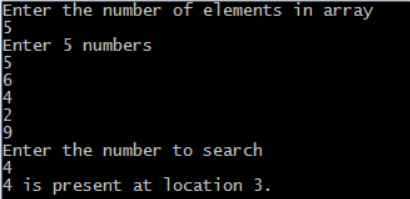
if (c == n)

printf("%d isn't present in the array.\n", search);

return 0;

}

O/P:

  
Q3: Binary Search  
SOLUTION:

CODE:

#include <stdio.h>

int main()

{

int i, low, high, mid, n, key, array[100];

printf("Enter number of elementsn");

scanf("%d",&n);

printf("Enter %d integersn", n);

for(i = 0; i < n; i++)

scanf("%d",&array[i]);

printf("Enter value to findn");

scanf("%d", &key);

low = 0;

high = n - 1;

mid = (low+high)/2;

while (low <= high) {

if(array[mid] < key)

low = mid + 1;

else if (array[mid] == key) {

printf("%d found at location %d.n", key, mid+1);

break;

}

else

high = mid - 1;

mid = (low + high)/2;

}

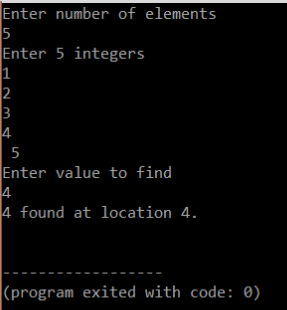
if(low > high)

printf("Not found! %d isn't present in the list.n", key);

return 0;

}

O/P:



Q4: Linked List

SOLUTION:

CODE:

#include <stdio.h>

#include <stdlib.h>

struct node{

int data;

struct node \*next;

};

struct node \*head, \*tail = NULL;

void addNode(int data) {

struct node \*newNode = (struct node\*)malloc(sizeof(struct node));

newNode->data = data;

newNode->next = NULL;

if(head == NULL) {

head = newNode;

tail = newNode;

}

else {

tail->next = newNode;

tail = newNode;

}

}

void display() {

struct node \*current = head;

if(head == NULL) {

printf("List is empty\n");

return;

}

printf("Nodes of singly linked list: \n");

while(current != NULL) {

printf("%d ", current->data);

current = current->next;

}

printf("\n");

}

int main()

{

addNode(1);

addNode(2);

addNode(3);

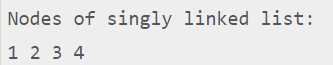
addNode(4);

display();

return 0;

}

O/P:



Q5: INFLIX TO POSTFLIX

SOLUTION:

CODE:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_EXPR\_SIZE 100

int precedence(char operator)

{

switch (operator) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

default:

return -1;

}

}

int isOperator(char ch)

{

return (ch == '+' || ch == '-' || ch == '\*' || ch == '/'

|| ch == '^');

}

char\* infixToPostfix(char\* infix)

{

int i, j;

int len = strlen(infix);

char\* postfix = (char\*)malloc(sizeof(char) \* (len + 2));

char stack[MAX\_EXPR\_SIZE];

int top = -1;

for (i = 0, j = 0; i < len; i++) {

if (infix[i] == ' ' || infix[i] == '\t')

continue;

if (isalnum(infix[i])) {

postfix[j++] = infix[i];

}

else if (infix[i] == '(') {

stack[++top] = infix[i];

}

else if (infix[i] == ')') {

while (top > -1 && stack[top] != '(')

postfix[j++] = stack[top--];

if (top > -1 && stack[top] != '(')

return "Invalid Expression";

else

top--;

}

else if (isOperator(infix[i])) {

while (top > -1

&& precedence(stack[top])

>= precedence(infix[i]))

postfix[j++] = stack[top--];

stack[++top] = infix[i];

}

}

while (top > -1) {

if (stack[top] == '(') {

return "Invalid Expression";

}

postfix[j++] = stack[top--];

} postfix[j] = '\0';

return postfix;

}

int main()

{

char infix[MAX\_EXPR\_SIZE] = "a+b\*(c^d-e)^(f+g\*h)-i";

char\* postfix = infixToPostfix(infix);

printf("%s\n", postfix);

free(postfix);

return 0;

}

O/P:



Q6: DUPLICATE ELEMENTS

SOLUTION:

CODE:

#include<stdio.h>

#define ARRAY\_SIZE(a) sizeof(a)/sizeof(a[0])

void findDuplicateElement(int arr[], const int size)

{

int i, j;

printf("Repeating elements are ");

for(i = 0; i < size; i++)

{

for(j = i+1; j < size; j++)

{

if(arr[i] == arr[j])

{

printf(" %d ", arr[i]);

}

}

}

}

int main()

{

int arr[] = {4, 2, 4, 5, 2, 3, 1};

const int N = ARRAY\_SIZE(arr);

findDuplicateElement(arr, N);

return 0;

}

O/P:

