DATA STRUCTURES AND ALGORITHMS

LAB TASK 2

NAME: SPARSH ARYA
REGISTERATION NUMBER: 17BEC0656
TEACHER: GAYATRI P

Question1.

C program to check whether the given expression is balanced or not.

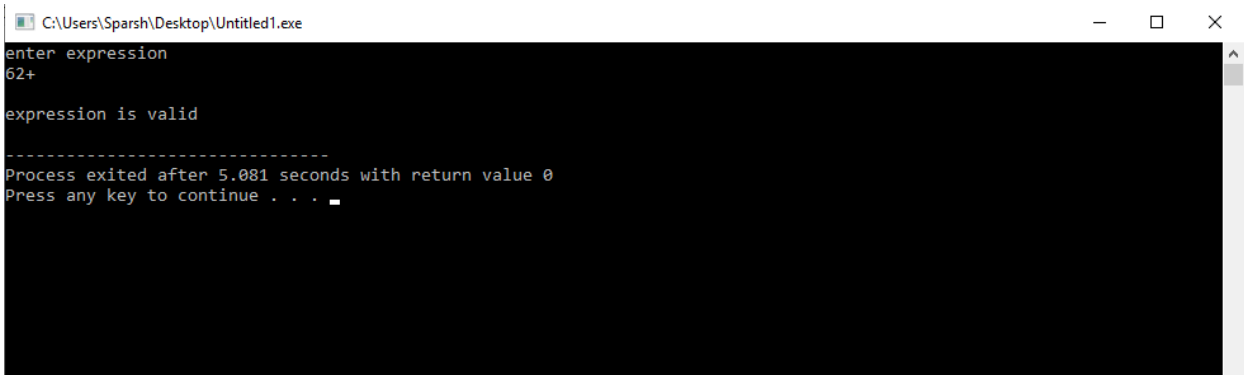
Pseudo code

```
1. Take a expression as input and store it in the array.
2. Check for the “(” and “)” in the expression.
3. If “(” encounters, then push it to the separate array. If “)” encounters, then pop the element of the array.
4. If the number of “(” and “)” are equal, then the expression is correctly parenthesized. Otherwise it is not.
```

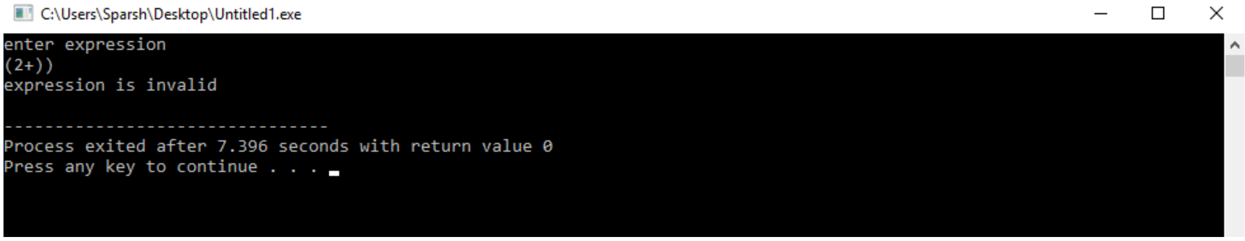
C Code

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

int top = -1;
char stack[100];

void push(char); void  pop(); void find_top();

int main()
{
int i;
char a[100];
printf("enter expression\n"); scanf("%s", &a);
for (i = 0; a[i] != '\0';i++)
{
if (a[i] == '(')
{
push(a[i]);
}
else if (a[i] == ')')
{
pop();
}
}
find_top();
return 0;
}

void push(char a)
{
stack[top] = a; top++;
}
void pop()
{
if (top == -1)
{
printf("expression is invalid\n"); exit(0);
}
else
{
top--;
}
}

void find_top()
{
if (top == -1)
printf("\nexpression is valid\n");
else
printf("\nexpression is invalid\n");

}
```

Output



```
enter expression
62+

expression is valid

--------------------------------
Process exited after 5.081 seconds with return value 0
Press any key to continue . . .
```



```
enter expression
(2+))
expression is invalid

--------------------------------
Process exited after 7.396 seconds with return value 0
Press any key to continue . . .
```

Question 2.

Menu-driven C program to implement queue ADT using array. Perform enqueue, dequeue and display operations.

Pseudo Code

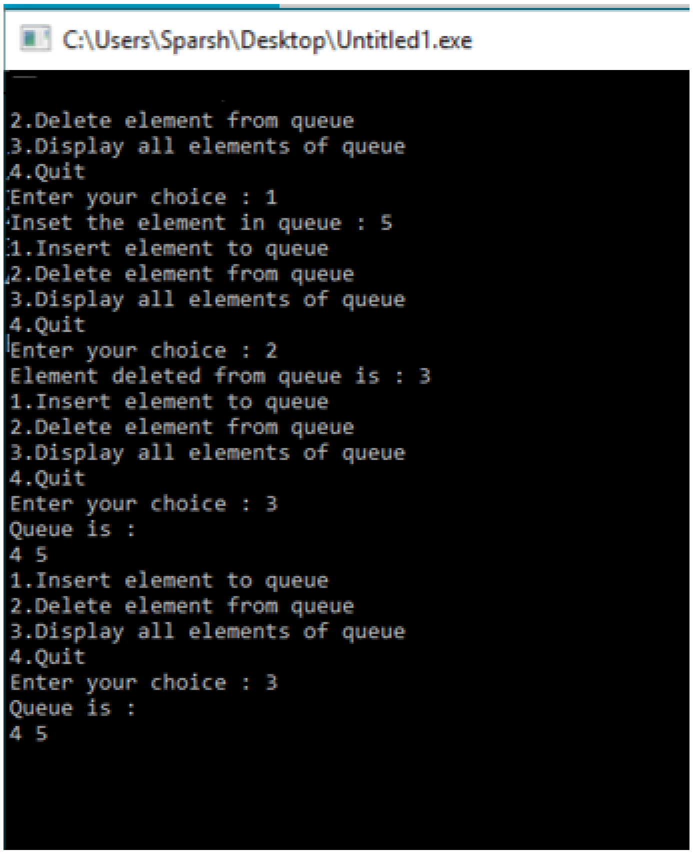
1. Use three functions for three operations like insert, delete and display.
2. Use switch statement to access these functions.
3. Exit.

C CODE

```
#include <stdio.h>
#define MAX 50

void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
   int choice;
   while (1)
   {
      printf("1.Insert element to queue \n");
      printf("2.Delete element from queue \n");
      printf("3.Display all elements of queue \n");
      printf("4.Quit \n");
      printf("Enter your choice : ");
      scanf("%d", &choice);
      switch (choice)
      {
         case 1:
         insert();
         break;
         case 2:
         delete();
         break;
         case 3:
         display();
         break;
         case 4:
         exit(1);
         default:
         printf("Wrong choice \n");
      } /* End of switch */
   } /* End of while */
} /* End of main() */

void insert()
{
   int add_item;
   if (rear == MAX - 1)
   printf("Queue Overflow \n");
   else
   {
      if (front == - 1)
      /*If queue is initially empty */
      front = 0;
      printf("Inset the element in queue : ");
      scanf("%d", &add_item);
      rear = rear + 1;
      queue_array[rear] = add_item;
   }
} /* End of insert() */

void delete()
{
   if (front == - 1 || front > rear)
   {
      printf("Queue Underflow \n");
      return ;
   }
   else
   {
      printf("Element deleted from queue is : %d\n", queue_array[front]);
      front = front + 1;
   }
```

```
    } /* End of delete() */

    void display()
    {
        int i;
        if (front == - 1)
            printf("Queue is empty \n");
        else
        {
            printf("Queue is : \n");
            for (i = front; i <= rear; i++)
                printf("%d ", queue_array[i]);
            printf("\n");
        }
}
```

OUTPUT-



```
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 5
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 2
Element deleted from queue is : 3
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 3
Queue is :
4 5
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 3
Queue is :
4 5
```

Q3.Menu-driven C program implement circular queue using array. Perform enqueue, dequeue and display operations.

Pseudo code

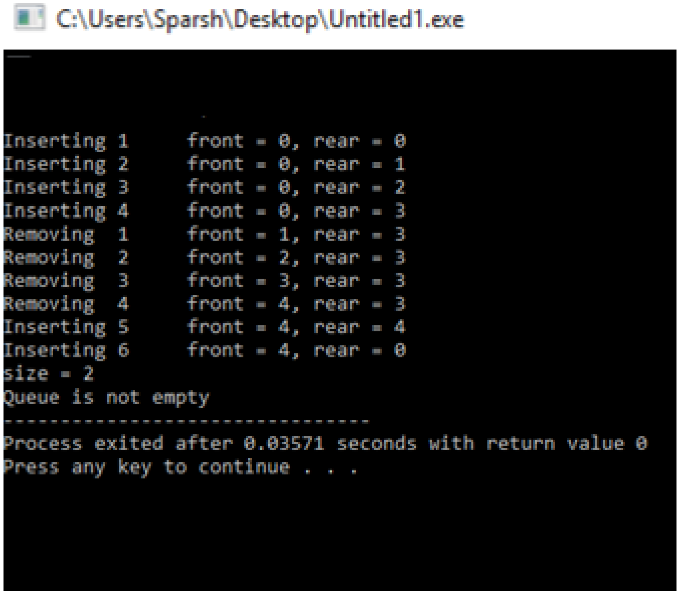
```
For Enqueue:
if (((front == 0) and (rear == n-1)) or (rear == front - 1)) //Check for overflow
    Print “Overflow”
    end Enqueue
 end if
 if (front == -1) //Inserting in an initially empty queue
     front = rear = 0
  end if
 else if (rear == n-1) // Inserting after the last element which is at n-1
      rear = 0
 end if
 else
      rear = rear +1 //Increment rear
 end else
 QUEUE [rear]  =   element //
 end Enqueue
For Dequeue:
 if(front == -1) //Check for overflow
    Print “Underflow”
    end Dequeue
 end if
 element = QUEUE[front] // Delete the front element
 if(front == rear) // The last element is deleted
     front = rear   = -1
 end if
 else if (front == n-1)
    front = 0
 end if
 else
     front = front + 1//Increment front
 end else
 end Dequeue
```

C code

```
#include <stdio.h>

// Data structure for queue
struct queue
{
    int *items;          // array to store queue elements
    int maxsize; // maximum capacity of the queue
    int front;           // front points to front element in the queue (if any)
    int rear;            // rear points to last element in the queue
    int size;            // current capacity of the queue
};

// Utility function to initialize queue
struct queue* newQueue(int size)
{
    struct queue *pt = NULL;
    pt = (struct queue*)malloc(sizeof(struct queue));

    pt->items = (int*)malloc(size * sizeof(int));
    pt->maxsize = size;
    pt->front = 0;
    pt->rear = -1;
    pt->size = 0;

    return pt;
}

// Utility function to return the size of the queue
int size(struct queue *pt)
{
    return pt->size;
}

// Utility function to check if the queue is empty or not
int isEmpty(struct queue *pt)
{
    return !size(pt);
}

// Utility function to return front element in queue
```

```
int front(struct queue *pt)
{
    if (isEmpty(pt))
    {
            printf("UnderFlow\nProgram Terminated\n");
            exit(EXIT_FAILURE);
    }

    return pt->items[pt->front];
}

// Utility function to add an element x in the queue
void enqueue(struct queue *pt, int x)
{
    if (size(pt) == pt->maxsize)
    {
            printf("OverFlow\nProgram Terminated\n");
            exit(EXIT_FAILURE);
    }

    printf("Inserting %d\t", x);

    pt->rear = (pt->rear + 1) % pt->maxsize;        // circular queue
    pt->items[pt->rear] = x;
    pt->size++;

    printf("front = %d, rear = %d\n", pt->front, pt->rear);
}

// Utility function to remove element from the queue
void dequeue(struct queue *pt)
{
    if (isEmpty(pt)) // front == rear
    {
            printf("UnderFlow\nProgram Terminated\n");
            exit(EXIT_FAILURE);
    }

    printf("Removing  %d\t", front(pt));

    pt->front = (pt->front + 1) % pt->maxsize;      // circular queue
    pt->size--;

    printf("front = %d, rear = %d\n", pt->front, pt->rear);
}

// main function
int main()
{
printf(“\n\n\n”);
struct queue *pt = newQueue(5);

    enqueue(pt, 1);
    enqueue(pt, 2);
    enqueue(pt, 3);
    enqueue(pt, 4);

    dequeue(pt);
    dequeue(pt);
    dequeue(pt);
    dequeue(pt);

    enqueue(pt, 5);
    enqueue(pt, 6);

    printf("size = %d\n", size(pt));

    if (isEmpty(pt))
            printf("Queue is empty");
    else
            printf("Queue is not empty");

    return 0;
}
```

<u>OUTPUT-</u>



C:\Users\Sparsh\Desktop\Untitled1.exe

```
Inserting 1     front = 0, rear = 0
Inserting 2     front = 0, rear = 1
Inserting 3     front = 0, rear = 2
Inserting 4     front = 0, rear = 3
Removing  1     front = 1, rear = 3
Removing  2     front = 2, rear = 3
Removing  3     front = 3, rear = 3
Removing  4     front = 4, rear = 3
Inserting 5     front = 4, rear = 4
Inserting 6     front = 4, rear = 0
size = 2
Queue is not empty
--------------------------------
Process exited after 0.03571 seconds with return value 0
Press any key to continue . . .
```

Q4. Menu driven C program to implement singly linked list. Menu should have the following operations:

a. Insertion

i. Beginning insertion

ii. End insertion

iii. Position insertion

b. Deletion

i. Beginning deletion

ii. End deletion

iii. Position deletion

c. Search

d. Display

e. Exit

C CODE-

```
#include<stdio.h>
#include<stdlib.h>
/*----Function Prototypes     */
void create();
void display();
void insert_begin();
void insert_end();
void insert_pos();
void delete_begin();
 void delete_end();
void delete_pos();

struct node
{
int info;
struct node *next;
};
struct node *start=NULL; int main()
{
int choice;
while(1){
printf("\n***SINGLE LINKED LIST OPERATIONS:****\n");
 printf("\n          MENU          \n");
printf("   \n");
printf("\n 0.Create      \n");
printf("\n 1.Beginning Insertion \n");
 printf("\n 2.End Insertion \n");
 printf("\n 3.Position Insertion        \n");
 printf("\n 4.Beginning Deletion                \n");
 printf("\n 5.End Deletion     \n");
 printf("\n 6.Position Deletion \n");
printf("\n 7.Display     \n");
printf("\n 8.Exit\n");
printf("\n          \n");
printf("Enter your choice:\t");
scanf("%d",&choice); switch(choice)
{
case 0:                 create(); break;

                        display(); break;
case 7:
                        insert_begin(); break;

case 1:                 insert_end(); break;

                        insert_pos(); break;
case 2:
                        delete_begin(); break;

case 3:                 delete_end(); break;

                        delete_pos(); break;
case 4:
                        exit(0); break;

case 5:                 printf("\n Wrong Choice:\n"); break;


case 6:


case 8:
default:
}
}
return 0;
}
void create()
{
struct node *temp,*ptr;
temp=(struct node *)malloc(sizeof(struct node)); if(temp==NULL)
{
printf("\nOut of Memory Space:\n"); exit(0);
}
printf("\nEnter the data value for the node:\t"); scanf("%d",&temp->info);
temp->next=NULL; if(start==NULL)
```

```
{
            start=temp;
}
else
{
            ptr=start;
            while(ptr->next!=NULL)
            {
ptr=ptr->next;
}
ptr->next=temp;
}
}
void display()
{
struct node *ptr; if(start==NULL)
{
            printf("\nList is empty:\n"); return;

}
else
{           ptr=start;
            printf("\nThe List elements are:\n"); while(ptr!=NULL)
            {
printf("%d\t",ptr->info ); ptr=ptr->next ;
}
}
}
void insert_begin()
{
struct node *temp;
temp=(struct node *)malloc(sizeof(struct node)); if(temp==NULL)
{
printf("\nOut of Memory Space:\n"); return;
}
printf("\nEnter the data value for the node:\t" ); scanf("%d",&temp->info);
temp->next =NULL; if(start==NULL)
{
start=temp;
}
```

```
else
{
	temp->next=start; start=temp;

}
}
void insert_end()
{
struct node *temp,*ptr;
temp=(struct node *)malloc(sizeof(struct node)); if(temp==NULL)
{
printf("\nOut of Memory Space:\n"); return;
}
printf("\nEnter the data value for the node:\t" ); scanf("%d",&temp->info );
temp->next =NULL; if(start==NULL)
{
	start=temp;
}
else
{
	ptr=start;
	while(ptr->next !=NULL)
	{
ptr=ptr->next ;
}
ptr->next =temp;
}
}
void insert_pos()
{
struct node *ptr,*temp; int i,pos;
temp=(struct node *)malloc(sizeof(struct node)); if(temp==NULL)
{
printf("\nOut of Memory Space:\n"); return;
}
printf("\nEnter the position for the new node to be inserted:\t");
scanf("%d",&pos);
printf("\nEnter the data value of the node:\t"); scanf("%d",&temp->info) ;

temp->next=NULL;
```

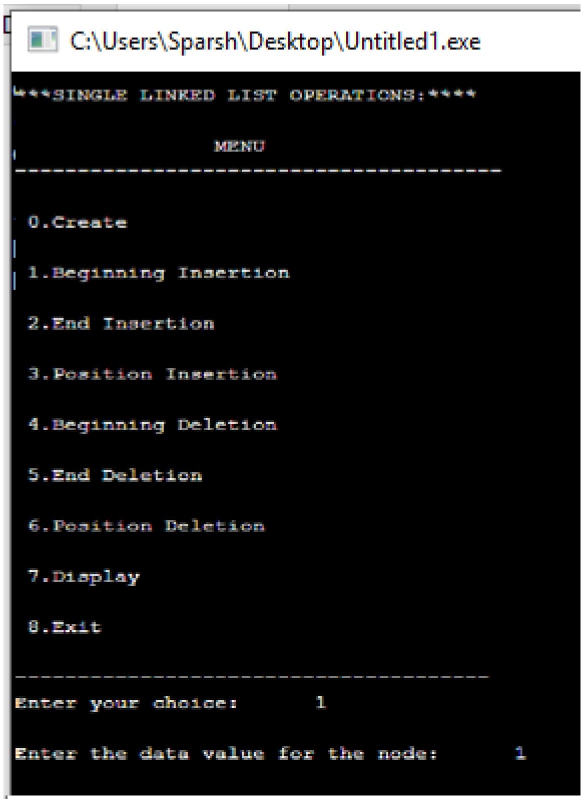
```
if(pos==0)
{
        temp->next=start; start=temp;

}
else
{       for(i=0,ptr=start;i<pos-2;i++)
        {
ptr=ptr->next; if(ptr==NULL)
{
printf("\nPosition not found:[Handle with care]\n"); return;
}
}
temp->next =ptr->next ; ptr->next=temp;
}
}
void delete_begin()
{
struct node *ptr; if(ptr==NULL)
{
        printf("\nList is Empty:\n"); return;

}
else
{       ptr=start; start=start->next ;
        printf("\nThe deleted element is :%d\t",ptr->info); free(ptr);



}
}
void delete_end()
{
struct node *temp,*ptr; if(start==NULL)
{
printf("\nList is Empty:"); exit(0);
}
else if(start->next ==NULL)
{
ptr=start; start=NULL;
```

```
            printf("\nThe deleted element is:%d\t",ptr->info); free(ptr);

}
else
{           ptr=start;
            while(ptr->next!=NULL)
            {
temp=ptr; ptr=ptr->next;
}
temp->next=NULL;
printf("\nThe deleted element is:%d\t",ptr->info); free(ptr);
}
}
void delete_pos()
{
int i,pos;
struct node *temp,*ptr; if(start==NULL)
{
            printf("\nThe List is Empty:\n"); exit(0);

}
else
{           printf("\nEnter the position of the node to be deleted:\t");
            scanf("%d",&pos);
            if(pos==0)
            {
                 ptr=start; start=start->next ;
                 printf("\nThe deleted element is:%d\t",ptr->info ); free(ptr);


}
else             ptr=start; for(i=0;i<pos-1;i++)
{                {
temp=ptr; ptr=ptr->next ; if(ptr==NULL)
{
printf("\nPosition not Found:\n"); return;
}
}
temp->next =ptr->next ;
printf("\nThe deleted element is:%d\t",ptr->info ); free(ptr);
}
}
}
```

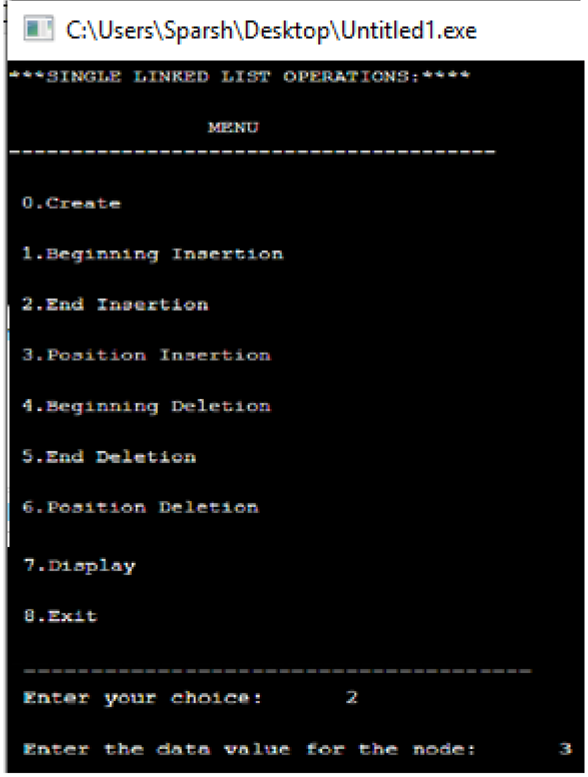
Pseudo code

```
1. Take input from user about option to choose.
2. Create functions for insering,deleting,searching and displaying
   the linked lists.
3. Based upon user specification the functions are called and the
   output is returned ti the main function.
4. Upon getting 8 as input the while loop terminates and the
   program comes out of the loop.
5. 0 is for create.
6. 1 is for beginning insertion
7. 2 is for end insertion
8. 3 is for position insertion
9. 4 is for beginning deletion
10. 5 is for end deletion
11. 6 is for position deletion.
12. 7 is for display
13. 8 is for exit.
```

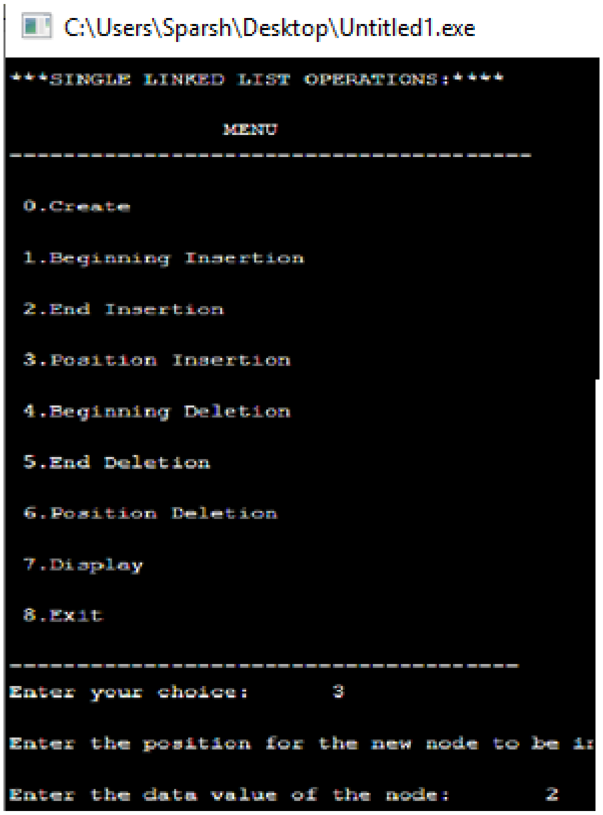
OUTPUT-



```
***SINGLE LINKED LIST OPERATIONS:****

                MENU
---------------------------------------

 0.Create

 1.Beginning Insertion

 2.End Insertion

 3.Position Insertion

 4.Beginning Deletion

 5.End Deletion

 6.Position Deletion

 7.Display

 8.Exit

--------------------------------------
Enter your choice:      1

Enter the data value for the node:      1
```

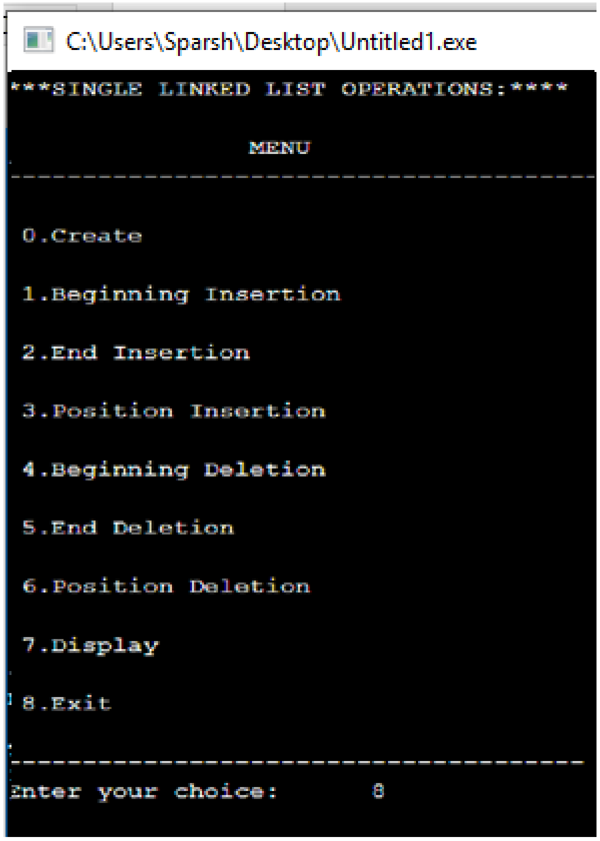


```
***SINGLE LINKED LIST OPERATIONS:****

                MENU
---------------------------------------

 0.Create

 1.Beginning Insertion

 2.End Insertion

 3.Position Insertion

 4.Beginning Deletion

 5.End Deletion

 6.Position Deletion

 7.Display

 8.Exit

 --------------------------------------
 Enter your choice:      2

 Enter the data value for the node:      3
```



C:\Users\Sparsh\Desktop\Untitled1.exe

```
***SINGLE LINKED LIST OPERATIONS:****

                MENU
---------------------------------------


 0.Create

 1.Beginning Insertion

 2.End Insertion

 3.Position Insertion

 4.Beginning Deletion

 5.End Deletion

 6.Position Deletion

 7.Display

 8.Exit

--------------------------------------
Enter your choice:      3

Enter the position for the new node to be i

Enter the data value of the node:       2
```



```
C:\Users\Sparsh\Desktop\Untitled1.exe

***SINGLE LINKED LIST OPERATIONS:****

                MENU
----------------------------------------

 0.Create

 1.Beginning Insertion

 2.End Insertion

 3.Position Insertion

 4.Beginning Deletion

 5.End Deletion

 6.Position Deletion

 7.Display

 8.Exit

---------------------------------------
Enter your choice:      8
```