1. **C program to check whether given expression is balanced expression or not using stack.**

**Pseudo Code:**

Input the expression, exp.

For i=0 with condition exp[i] not equals ‘\0’

If exp[i] is ‘(‘

Push ‘(‘ in the stack

Else if exp[i] is ‘)’

While top is not -1 and stack[top] is not ‘(‘

Pop from stack

Pop from stack

If top>=0

Print ‘Unbalanced’

Else

Print ‘Balanced’

**C Code:**

#include<stdio.h>

#include<stdlib.h>

char stack[50];

int top=-1;

void push(char c)

{

stack[++top]=c;

}

void pop()

{

top--;

}

int main()

{

char exp[50];

int i;

scanf("%s",exp);

for(i=0;exp[i]!='\0';i++)

{

if(exp[i]=='(')

push('(');

else if(exp[i]==')')

{

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

pop();

pop();

}

}

if(top>=0)

printf("Unbalanced");

else

printf("Balanced");

return 0;

}



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

**Pseudo Code:**

Enqueue()

If rear==max size

Print ‘Overflow’

Else

Input the element

Rear++

Queue[rear]=element

If front==-1

Front=rear

Print “Element entered!”

Dequeue()

If front==-1

Print “Underflow”

Else

Item = queue[front]

If front==rear

Front=rear=-1

Else

Front++

Print “Element deleted!”

Display()

If front==-1

Print “Underflow!”

Else

Item = queue[front]

Print item

Main()

While(1)

Input choice

Switch(choice)

Case 1: Enqueue

Case 2: Dequeue

Case 3: Display

Case 3: Exit

**C Code:**

#include<stdio.h>

#include<stdlib.h>

#define MAX\_SIZE 50

int queue[MAX\_SIZE];

int front=-1;

int rear=-1;

void enqueue()

{

int element;

if(rear==MAX\_SIZE)

printf("Overflow!\n");

else

{

printf("Enter the element you wish to enter: \n");

scanf("%d",&element);

rear++;

queue[rear]=element;

if(front==-1)

front++;

printf("%d entered!\n",element);

}

}

void dequeue()

{

int item;

if(front==-1)

printf("Underflow!\n");

else

{

item=queue[front];

if(front==rear)

{

front=-1;

rear=-1;

}

else

front++;

printf("%d deleted!\n",item);

}

}

void display()

{

if(front==-1)

printf("Underflow!\n");

else

{

printf("Front element is: %d\n",queue[front]);

}

}

int main()

{

int choice,item;

while(1)

{

printf("Enter your choice:\n");

printf("1. Enter an element.\n");

printf("2. Delete an element.\n");

printf("3. View the current element.\n");

printf("4. Exit.\n");

scanf("%d",&choice);

switch(choice)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

exit(0);

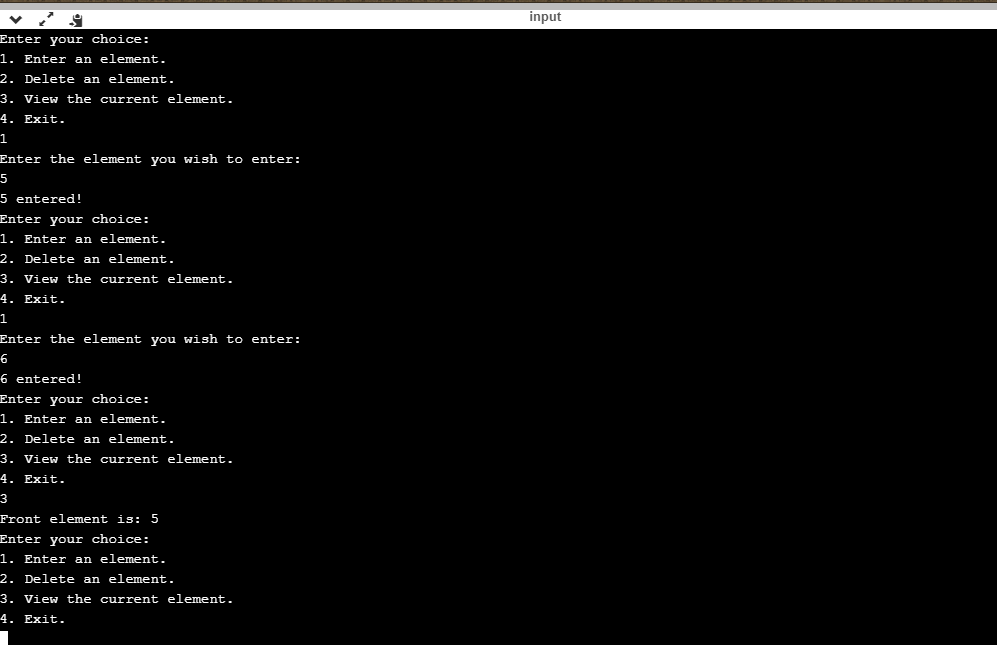
default:

printf("Invalid input! Try again: \n");

}

}

}



1. **Menu driven C program to implement circular queue using array. Perform enqueue, dequeue and display operations.**

**Pseudo Code:**

Enqueue()

If (rear==max\_size-1 and front==0) or (rear==front+1)

Print “Overflow”

Else

Input element

If rear==max\_size-1

Rear=0

Circular\_queue[rear]=element

Else

Rear++

Circular\_queue[rear]=element

If front==-1

Front++

Print “Element entered!”

Dequeue()

If front==-1

Print “Underflow!”

Else

Item=circular\_queue[front]

If front==rear

Front=-1

Rear=-1

Else if front==max\_size-1

Front=0

Else

Front++

Print item

Display()

If front==-1

Print “Underflow!”

Else

Print “Front element is:” , circular\_queue[front]

Main()

While(1)

Input choice

Switch(choice)

Case 1: Enqueue()

Case 2: Dequeue()

Case 3: Display()

Case 4: Exit()

**C Code:**

#include<stdio.h>

#include<stdlib.h>

#define MAX\_SIZE 50

int circular\_queue[MAX\_SIZE];

int front=-1;

int rear=-1;

void enqueue()

{

int element;

if((rear==MAX\_SIZE-1 && front==0)||(rear==front+1))

printf("Overflow!\n");

else

{

printf("Enter the element you wish to enter: \n");

scanf("%d",&element);

if(rear==MAX\_SIZE-1)

{

rear=0;

circular\_queue[rear]=element;

}

else

{

rear++;

circular\_queue[rear]=element;

}

if(front==-1)

front++;

printf("%d entered!\n",element);

}

}

void dequeue()

{

int item;

if(front==-1)

printf("Underflow!\n");

else

{

item=circular\_queue[front];

if(front==rear)

{

front=-1;

rear=-1;

}

else if(front==MAX\_SIZE)

front=0;

else

front++;

printf("%d deleted!\n",item);

}

}

void display()

{

if(front==-1)

printf("Underflow!\n");

else

{

printf("Front element is:%d\n",circular\_queue[front]);

}

}

int main()

{

int choice,item;

while(1)

{

printf("Enter your choice:\n");

printf("1. Enter an element.\n");

printf("2. Delete an element.\n");

printf("3. View the current element.\n");

printf("4. Exit.\n");

scanf("%d",&choice);

switch(choice)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

exit(0);

default:

printf("Invalid input! Try again: \n");

}

}

return 0;

}



1. **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**

**Pseudo Code:**

Struct node

{

Int data

Struct node \*next

}\*list=0;

Beg\_insert()

Input value

New\_node=malloc(sizeof(struct node))

New\_node->data=val

New\_node->next=0

If list==0

List=new\_node

Else

New\_node->next=list

List=new\_node

End\_indert()

Input val

Struct node \*temp,\*new\_node

New\_node=malloc(sizeof(struct node))

New\_node->data=val

New\_node->next=0

If list==0

List=new\_node

Else

Temp=list

While(temp->next != 0)

Temp=temp->next

Temp->next=new\_node

Int count\_node() // To count the number of nodes

Struct node \*temp

Count=0

If list==0

Return 0

Else

Temp=list

While(temp->next!=0)

Temp=temp->next

Count++

Return count

Pos\_insert()

Input element

Input position

Struct node \*temp

Current\_pos=1

If position>count\_node+1

Print “Not enough nodes.”

Else

Temp==list

While temp->next!=0 and current\_pos<pos-1

Temp=temp->next

Current\_pos++

New\_node=malloc(sizeof(struct node))

New\_node->data=element

New\_node->next=temp->next

Temp->next=new\_node

Beg\_del()

If list==0

Print “Empty list”

Else

Struct node \*s

S=list

Item=list->data

List=list->next

Print item

Free(s)

End\_del()

If list==0

Print “Empty list”

Else

Struct node \*temp,\*s

Temp=list

While temp->next!=0

S=temp

Temp=temp->next

Item=temp->data

Print item

s->next=0

free(temp)

Post\_del()

If list==0

Print “Empty list”

Else

Input value

Struct node \*temp,\*s

Temp=list

While(temp->data!=value and temp->next!=0)

S=temp

Temp=temp->next

If temp->data == value

S->next=temp->next

Free(temp)

Else

Print “Value not found!”

Search()

If list==0

Print “Empty list”

Else

Input element to be searched

Struct node \*temp

Temp=list

While temp->data!=element and temp->next!=0

Temp=temp->next

If temp->data = element

Print “Element found”

Else

Print “Element not found!”

Display()

Input position

Int current\_pos=1

If list==0

Print “Empty list”

Else

Struct node \*temp

Temp=list

If position>count\_nodes()

Print “Not enough nodes!”

Else

While(temp->next!=0 and current\_pos<positon)

Temp=temp->next

Current\_pos++

Print temp->data

**C Code:**

#include<stdio.h>

#include<stdlib.h>

#define MAX\_SIZE 50

struct node

{

int data;

struct node \*next;

}\*list=0;

void beg\_insert()

{

int val;

printf("Enter the element you wish to enter:\n ");

scanf("%d",&val);

struct node \*new\_node;

new\_node=malloc(sizeof(struct node));

new\_node->data=val;

new\_node->next=0;

if(list==0)

list=new\_node;

else

{

new\_node->next=list;

list=new\_node;

}

}

void end\_insert()

{

int val;

printf("Enter the element you wish to enter: \n");

scanf("%d",&val);

struct node \*new\_node,\*temp;

new\_node=malloc(sizeof(struct node));

new\_node->data=val;

new\_node->next=0;

if(list==0)

list=new\_node;

else

{

temp=list;

while(temp->next!=0)

temp=temp->next;

temp->next=new\_node;

}

}

int count\_node()

{

struct node \*temp;

int count=1;

if(list==0)

return 0;

else

{

temp=list;

while(temp->next!=0)

{

temp=temp->next;

count++;

}

return count;

}

}

void pos\_insert()

{

int val,position,current\_pos=1;

struct node \*new\_node,\*temp;

printf("Enter the element you wish to enter: \n");

scanf("%d",&val);

printf("Enter the position: \n");

scanf("%d",&position);

if(position>count\_node()+1)

printf("There are not enough nodes. Try again!\n");

else

{

temp=list;

while(temp->next!=0 && current\_pos<position-1)

{

temp=temp->next;

current\_pos++;

}

new\_node=malloc(sizeof(struct node));

new\_node->data=val;

new\_node->next=temp->next;

temp->next=new\_node;

}

}

void beg\_delete()

{

int item;

struct node\* s;

if(list==0)

printf("Empty list!\n");

else

{

s=list;

item=list->data;

list=list->next;

printf("%d deleted from beginning.\n",item);

free(s);

}

}

void end\_delete()

{

struct node \*temp,\*s;

int item;

if(list==0)

printf("Empty list!\n");

else

{

temp=list;

while(temp->next!=0)

{

s=temp;

temp=temp->next;

}

item=temp->data;

s->next=0;

printf("%d deleted from the end.\n",item);

free(temp);

}

}

void pos\_delete()

{

int val;

struct node \*temp,\*s;

if(list==0)

printf("Empty list!\n");

else

{

printf("Enter the value you wish to delete: \n");

scanf("%d",&val);

temp=list;

while(temp->data!=val && temp->next!=0)

{

s=temp;

temp=temp->next;

}

if(temp->data==val)

{

s->next=temp->next;

temp->next=0;

printf("%d deleted!\n",temp->data);

free(temp);

}

else

printf("Element not found!\n");

}

}

void search()

{

int val,count=1;

struct node\* temp;

if(list==0)

printf("Empty list!\n");

else

{

printf("Enter the value you want to search for: \n");

scanf("%d",&val);

temp=list;

while(temp->next!=0 && temp->data!=val)

{

temp=temp->next;

count++;

}

if(temp->data==val)

printf("%d found at node %d\n",val,count);

else

printf("Element not found!\n");

}

}

void display()

{

int position,current\_pos=1;

struct node\* temp;

if(list==0)

printf("Empty list!\n");

else

{

printf("Enter the position of the element you want to view.\n");

scanf("%d",&position);

if(position>count\_node())

printf("Not enough nodes!\n");

else

{

temp=list;

while(temp->next!=0 && current\_pos<position)

{

temp=temp->next;

current\_pos++;

}

printf("The element is: %d\n",temp->data);

}

}

}

int main()

{

int choice,type;

while(1)

{

printf("Enter your choice: \n");

printf("1. Insertion\n");

printf("2. Deletion\n");

printf("3. Search\n");

printf("4. Display\n");

printf("5. Exit\n");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter type:\n");

printf("1. Beginning Insertion\n");

printf("2. End Insertion\n");

printf("3. Position Insertion\n");

scanf("%d",&type);

switch(type)

{

case 1:

beg\_insert();

break;

case 2:

end\_insert();

break;

case 3:

pos\_insert();

break;

default:

printf("Invalid input!\n");

}

break;

case 2:

printf("Enter type:\n");

printf("1. Beginning Deletion\n");

printf("2. End Deletion\n");

printf("3. Position Deletion\n");

scanf("%d",&type);

switch(type)

{

case 1:

beg\_delete();

break;

case 2:

end\_delete();

break;

case 3:

pos\_delete();

break;

default:

printf("Invalid input!\n");

}

break;

case 3:

search();

break;

case 4:

display();

break;

case 5:

exit(0);

default:

printf("Invalid input!\n");

}

}

}



