ASSIGNMENT-05

Q1Write a C/C++ program to implement a circular linked list with the

following operations:

a) Insert an element at a specific position specified by the user.

b) Insert an element at the beginning of the list

c) Insert an element at the end of the list.

d) Delete an element from a specific position specified by the user.

e) Delete the first element from the list.

f) Delete the last element from the list.

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node\* link;

};

void add\_beg(int data, struct node\*\* head) {

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

if (temp == NULL) {

printf("Memory allocation failed\n");

return;

}

if (\*head == NULL) {

temp->data = data;

temp->link = temp;

\*head = temp;

return;

}

struct node\* tail = \*head;

while (tail->link != \*head) {

tail = tail->link;

}

temp->data = data;

temp->link = \*head;

tail->link = temp;

\*head = temp;

}

void add\_end(int data, struct node\* head) {

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

if (temp == NULL) {

printf("Memory not available\n");

return;

}

if (head == NULL) {

temp->data = data;

temp->link = temp;

return;

}

struct node\* tail = head;

while (tail->link != head) {

tail = tail->link;

}

temp->data = data;

tail->link = temp;

temp->link = head;

}

void print(struct node\* head) {

if (head == NULL) return;

struct node\* temp = head;

do {

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

temp = temp->link;

} while (temp != head);

printf("\n");

}

int count(struct node\* head) {

if (head == NULL) return 0;

int count = 1;

struct node\* temp = head->link;

while (temp != head) {

count++;

temp = temp->link;

}

return count;

}

void add\_pos(struct node\*\* head, int pos, int data) {

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

if (temp == NULL) {

printf("Memory allocation failed\n");

return;

}

if (\*head == NULL) {

temp->data = data;

temp->link = temp;

\*head = temp;

return;

}

int list\_size = count(\*head);

if (pos == 1) {

add\_beg(data, head);

free(temp);

return;

} else if (pos > list\_size + 1 || pos <= 0) {

printf("Invalid position\n");

free(temp);

return;

} else if (pos == list\_size + 1) {

add\_end(data, \*head);

free(temp);

return;

} else {

struct node\* current = \*head;

for (int i = 1; i < pos - 1; i++) {

current = current->link;

}

temp->data = data;

temp->link = current->link;

current->link = temp;

}

}

int main() {

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

if (head == NULL) {

printf("Memory not available \n");

return 1;

}

head->data = 1;

head->link = head;

printf("%d\n", head->data);

add\_beg(2, &head);

printf("After adding at the beginning: %d\n", head->data);

printf("Second element: %d\n", head->link->data);

print(head);

add\_end(0, head);

printf("\n");

printf("Adding the elemtns in the end\n");

print(head);

add\_pos(&head, 2, 7);

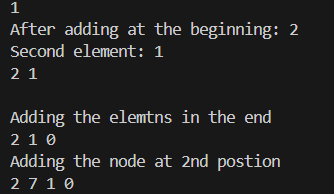
printf("Adding the node at 2nd postion\n");

print(head);

return 0;

}

OUTPUT:



#include <stdio.h>

#include <stdlib.h>

struct node{

int data;

struct node\*link;

};

void add\_beg(int data,struct node\*\*head){

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

if (temp==NULL){

printf("Memory allocation failed\n");

return;

}

if (\*head==NULL){

temp->data = data;

temp->link = temp;

\*head = temp;

return;

}

struct node\*tail = (\*head);

while (tail->link!=\*head){

tail = tail->link;

}

temp->data = data;

temp->link = \*head;

tail->link = temp;

\*head = temp;

}

int main(){

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

if (head==NULL){

printf("Memory not available \n");

return 1;

}

head->data = 1;

head->link = head;

struct node\*tail = head;

printf("%d\n",(tail->data));

add\_beg(2,&head);

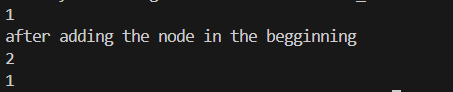
printf("after adding the node in the begginning\n");

printf("%d\n",(head->data));

printf("%d\n",(head->link->data));

}

OUTPUT:



#include <stdio.h>

#include <stdlib.h>

struct node{

int data;

struct node\*link;

};

void add\_beg(int data,struct node\*\*head){

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

if (temp==NULL){

printf("Memory allocation failed\n");

return;

}

if (\*head==NULL){

temp->data = data;

temp->link = temp;

\*head = temp;

return;

}

struct node\*tail = (\*head);

while (tail->link!=\*head){

tail = tail->link;

}

temp->data = data;

temp->link = \*head;

tail->link = temp;

\*head = temp;

}

void add\_end(int data,struct node\*head){

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

if (temp==NULL){

printf("Memory not available\n");

return;

}

struct node\*tail = head;

if(head==NULL){

add\_beg(data,&head);

return;

}

while(tail->link!=head){

tail = tail->link;

}

tail->link = temp;

temp->data = data;

temp->link = head;

}

void print(struct node\*head){

struct node\*temp = head;

while (temp->link!=head){

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

temp = temp->link;

}

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

}

int main(){

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

if (head==NULL){

printf("Memory not available \n");

return 1;

}

head->data = 1;

head->link = head;

struct node\*tail = head;

printf("%d\n",(tail->data));

add\_beg(2,&head);

printf("after adding the node in the begginning\n");

print(head);

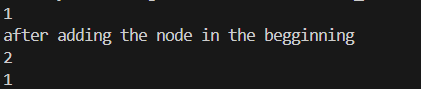
printf("after adding the node in the end\n");

add\_end(0,head);

print(head);

}

OUTPUT:



#include <stdio.h>

#include <stdlib.h>

struct node{

int data;

struct node\*link;

};

void add\_end(int data,struct node\*\*head){

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

if (temp == NULL) {

printf("Memory not available\n");

return;

}

temp->data = data;

if (\*head == NULL) {

temp->link = temp;

\*head = temp;

return;

}

struct node \*tail = \*head;

while (tail->link != \*head) {

tail = tail->link;

}

temp->link = \*head;

tail->link = temp;

}

void del\_beg(struct node\*\*head){

struct node\*tail = \*head;

while(tail->link!=(\*head)){

tail = tail->link;

}

tail->link = (\*head)->link;

free(\*head);

\*head = tail->link;

}

void del\_end(struct node\*\*head){

if (\*head == NULL) {

printf("List is empty, cannot delete.\n");

return;

}

if ((\*head)->link == \*head) {

free(\*head);

\*head = NULL;

return;

}

struct node\*current = \*head;

struct node\*previous = NULL;

while(current->link!=\*head){

previous = current;

current = current->link;

}

previous->link = \*head;

free(current);

}

void print(struct node\*head){

if (head == NULL) {

printf("List is empty\n");

return;

}

struct node \*temp = head;

do {

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

temp = temp->link;

} while (temp != head);

}

void del\_pos(struct node\*\*head,int pos,int n){

if (\*head==NULL){

printf("List is empty");

return;

}

if (pos==0 || pos>(n+1)){

printf("Enter an appropriate position \n");

return;

}

if (pos==1){

del\_beg(head);

return;

}

if (pos==n+1){

del\_end(head);

return;

}

struct node\*previous = NULL;

struct node\*current = (\*head);

for (int i = 1;i<=(pos-1);i++){

previous = current;

current = current->link;

}

previous->link = current->link;

free(current);

}

int main(){

int n;

printf("Enter the number of numbers you want to enter: ");

scanf("%d",&n);

int array[n];

struct node\*head = NULL;

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

printf("Enter a number: ");

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

}

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

add\_end(array[i],&head);

}

printf("Content of the linked list \n");

print(head);

free(array);

int pos;

printf("Enter the postion: ");

scanf("%d",&pos);

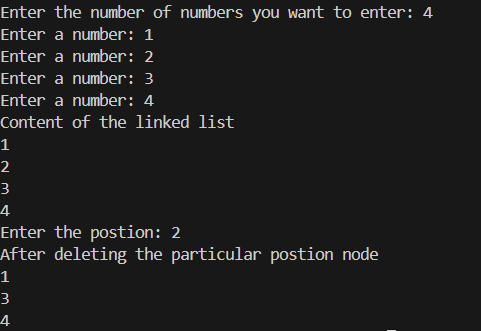
del\_pos(&head,pos,n);

printf("After deleting the particular postion node\n");

print(head);

}

OUTPUT:



#include <stdio.h>

#include <stdlib.h>

struct node{

int data;

struct node\*link;

};

void add\_beg(int data,struct node\*\*head){

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

if (temp==NULL){

printf("Memory allocation failed\n");

return;

}

if (\*head==NULL){

temp->data = data;

temp->link = temp;

\*head = temp;

return;

}

struct node\*tail = (\*head);

while (tail->link!=\*head){

tail = tail->link;

}

temp->data = data;

temp->link = \*head;

tail->link = temp;

\*head = temp;

}

void add\_end(int data,struct node\*\*head){

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

if (temp == NULL) {

printf("Memory not available\n");

return;

}

temp->data = data;

if (\*head == NULL) {

temp->link = temp;

\*head = temp;

return;

}

struct node \*tail = \*head;

while (tail->link != \*head) {

tail = tail->link;

}

temp->link = \*head;

tail->link = temp;

}

void print(struct node\*head){

if (head == NULL) {

printf("List is empty\n");

return;

}

struct node \*temp = head;

do {

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

temp = temp->link;

} while (temp != head);

}

void del\_beg(struct node\*\*head){

struct node\*tail = \*head;

while(tail->link!=(\*head)){

tail = tail->link;

}

tail->link = (\*head)->link;

free(\*head);

\*head = tail->link;

}

int main(){

int n;

printf("Enter the number of numbers you want to enter: ");

scanf("%d",&n);

int array[n];

struct node\*head = NULL;

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

printf("Enter a number: ");

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

}

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

add\_end(array[i],&head);

}

printf("Content of the linked list \n");

print(head);

free(array);

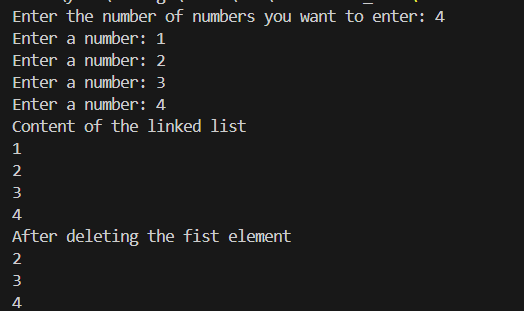
del\_beg(&head);

printf("After deleting the fist element\n");

print(head);

}

OUTPUT:



#include <stdio.h>

#include <stdlib.h>

struct node{

int data;

struct node\*link;

};

void add\_beg(int data,struct node\*\*head){

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

if (temp==NULL){

printf("Memory allocation failed\n");

return;

}

if (\*head==NULL){

temp->data = data;

temp->link = temp;

\*head = temp;

return;

}

struct node\*tail = (\*head);

while (tail->link!=\*head){

tail = tail->link;

}

temp->data = data;

temp->link = \*head;

tail->link = temp;

\*head = temp;

}

void add\_end(int data,struct node\*\*head){

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

if (temp == NULL) {

printf("Memory not available\n");

return;

}

temp->data = data;

if (\*head == NULL) {

temp->link = temp;

\*head = temp;

return;

}

struct node \*tail = \*head;

while (tail->link != \*head) {

tail = tail->link;

}

temp->link = \*head;

tail->link = temp;

}

void print(struct node\*head){

if (head == NULL) {

printf("List is empty\n");

return;

}

struct node \*temp = head;

do {

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

temp = temp->link;

} while (temp != head);

}

void del\_end(struct node\*\*head){

if (\*head == NULL) {

printf("List is empty, cannot delete.\n");

return;

}

if ((\*head)->link == \*head) {

free(\*head);

\*head = NULL;

return;

}

struct node\*current = \*head;

struct node\*previous = NULL;

while(current->link!=\*head){

previous = current;

current = current->link;

}

previous->link = \*head;

free(current);

}

int main(){

int n;

printf("Enter the number of numbers you want to enter: ");

scanf("%d",&n);

int array[n];

struct node\*head = NULL;

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

printf("Enter a number: ");

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

}

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

add\_end(array[i],&head);

}

printf("Content of the linked list \n");

print(head);

free(array);

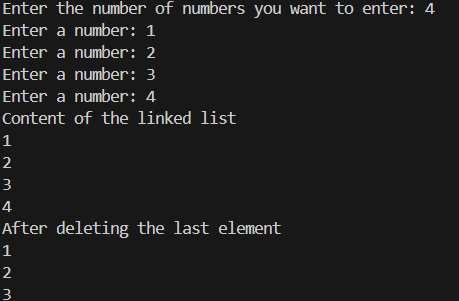
del\_end(&head);

printf("After deleting the last element\n");

print(head);

}

OUTPUT:



Q2: Write a C/C++ code to implement stack with following operations using

array.

a) create () = Create a stack.

b) push() = Pushing (storing) an element on the stack

c) pop() = Removing (accessing) an element from the stack.

d) peek() = Get the top data element of the stack, without removing it

e) isFull() = Check if stack is full.

f) isEmpty() = Check whether the stack is empty, and return true or false.

#include <stdio.h>

#include <stdbool.h>

#define MAX 100

struct stack {

int stArr[MAX];

int top;

};

typedef struct stack STACK;

void create(STACK\* s) {

s->top = -1;

}

void push(STACK\* s, int value) {

if (s->top == MAX - 1) {

printf("Stack Overflow Cannot push element.\n");

return;

}

(s->top)+=1;

s->stArr[s->top] = value;

printf("%d pushed onto the stack.\n", value);

}

int pop(STACK\* s) {

if (s->top == -1) {

printf("Stack Underflow Cannot pop element.\n");

return -1;

}

(s->top)-=1;

int poppedValue = s->stArr[s->top];

printf("%d popped from the stack.\n", poppedValue);

return poppedValue;

}

int peek(STACK\* s) {

if (s->top == -1) {

printf("Stack is empty.\n");

return -1;

}

return s->stArr[s->top];

}

bool isFull(STACK\* s) {

return s->top == MAX - 1;

}

bool isEmpty(STACK\* s) {

return s->top == -1;

}

int main() {

STACK myStack;

create(&myStack);

push(&myStack, 10);

push(&myStack, 20);

push(&myStack, 30);

printf("Top element: %d\n", peek(&myStack));

pop(&myStack);

pop(&myStack);

if (isFull(&myStack)){

printf("Stack overflow\n");

}

else{

printf("The stack is not overflowed \n");

}

if (isEmpty(&myStack)){

printf("The stack is underflowed \n");

}

else{

printf("The stack is not underflowed \n");

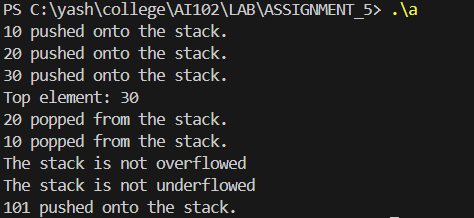
}

push(&myStack, 101);

return 0;

}

OUTPUT:



Q3 Q3: Write a C/C++ code to implement stack with all the operations defined in

Q2 using Linked list.

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node\* link;

};

void push(struct node\*\* top, int value) {

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

if (new\_node == NULL) {

printf("Stack overflow \n");

exit(1);

}

new\_node->data = value;

new\_node->link = \*top;

\*top = new\_node;

printf("%d pushed onto the stack.\n", value);

}

void pop(struct node\*\* top) {

if (\*top == NULL) {

printf("Stack underflow \n");

return;

}

struct node\* temp = \*top;

printf("%d popped from the stack.\n", temp->data);

\*top = (\*top)->link;

free(temp);

}

int peek(struct node\* top) {

if (top == NULL) {

printf("Stack is empty\n");

return -1;

}

return top->data;

}

int is\_empty(struct node\* top) {

return (top == NULL);

}

int main() {

struct node\* top = NULL;

if (is\_empty(top)) {

printf("Stack is empty\n");

}

push(&top, 1);

push(&top, 2);

printf("Top element: %d\n", peek(top));

pop(&top);

pop(&top);

if (is\_empty(top)) {

printf("Stack is empty\n");

}

}

OUTPUT:

