Q. WAP to convert given Infix expression into its equivalent Postfix expression and evaluate it using stack in C

#include<stdio.h>

#include<ctype.h>

char stack[100];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

if(top == -1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x == '(')

return 0;

if(x == '+' || x == '-')

return 1;

if(x == '\*' || x == '/')

return 2;

return 0;

}

int main()

{

char exp[100];

char \*e, x;

printf("Enter the expression : ");

scanf("%s",exp);

printf("\n");

e = exp;

while(\*e != '\0')

{

if(isalnum(\*e))

printf("%c ",\*e);

else if(\*e == '(')

push(\*e);

else if(\*e == ')')

{

while((x = pop()) != '(')

printf("%c ", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

printf("%c ",pop());

push(\*e);

}

e++;

}

while(top != -1)

{

printf("%c ",pop());

}return 0;

}

Q. WAP to implement two stack using array and perform following operations on it. A.PUSH, B.POP, C.StackFull, AND.Stack Empty.Display Stack. #include<stdio.h>

int stack[100],choice,n,top,x,i;

void push(void);

void pop(void);

void display(void);

int main()

{

top=-1;

printf("\n Enter the size of STACK[MAX=100]:");

scanf("%d",&n);

printf("\n\t STACK OPERATIONS USING ARRAY");

printf("\n\t--------------------------------");

printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");

do

{

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

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("\n\t EXIT POINT ");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");

}

}

}

while(choice!=4);

return 0;

}

void push()

{

if(top>=n-1)

{

printf("\n\tSTACK is over flow");

}

else

{

printf(" Enter a value to be pushed:");

scanf("%d",&x);

top++;

stack[top]=x;

}

}

void pop()

{

if(top<=-1)

{

printf("\n\t Stack is under flow");

}

else

{

printf("\n\t The popped elements is %d",stack[top]);

top--;

}

}

void display()

{

if(top>=0)

{

printf("\n The elements in STACK \n");

for(i=top; i>=0; i--)

printf("\n%d",stack[i]);

printf("\n Press Next Choice");

}

else

{

printf("\n The STACK is empty");

}

}

WAP to implement following by using stack.

A.Factorial of a given number B.Generation of Fibonacci series in C

A.

#include<stdio.h>

int stk[100]; // stack

int size = 100; // size of stack

int ptr = -1; // store the index of top element of the stack

// push x to stack

void push(int x){

if(ptr==size-1){

printf("OverFlow \n");

}

else{

++ptr;

stk[ptr] = x;

}

}

int top(){

if(ptr==-1){

printf("UnderFlow \n");

return -1;

}

else{

return stk[ptr];

}

}

void pop(){

if(ptr==-1){

printf("UnderFlow \n");

}

else{

--ptr;

}

}

int isempty(){

if(ptr==-1)

return 1;

else

return 0;

}

int main() {

int i, n;

printf("Enter a number: ");

scanf("%d", &n);

push(1);

for(i=2;i<=n;++i){

push(top() \* i);

}

printf("Factorial: %d", top());

return 0;

}

B.

#include <stdio.h>

// fibonacci() funtion definition

int fibonacci(int num)

{

// first base condition check

if (num == 0)

{

return 0; // retuning 0, if condition meets

}

// second base condition check

else if (num == 1)

{

return 1; // returning 1, if condition meets

}

// else calling the fibonacci() function recursively till we get to the base conditions

else

{

return fibonacci(num - 1) + fibonacci(num - 2); // recursively calling the fibonacc() function and then adding them

}

}

int main()

{

int num; // variable to store how many elements to be displayed in the series

printf("Enter the number of elements to be in the series : ");

scanf("%d", &num); // taking user input

int i;

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

{

printf("%d, ", fibonacci(i)); // calling fibonacci() function for each iteration and printing the returned value

}

return 0;

}

Q. Write a Program to implement circular double ended queue where user can add and remove the elements from both front and rear of the queue in C

#include <stdio.h>

#define MAX 100 // Maximum size of the queue

// Structure to represent a circular queue

typedef struct CQueue {

int A[MAX]; // Array to store the queue elements

int front; // Index of the front element

int rear; // Index of the rear element

} CQueue;

// Function to check if the queue is empty

int isEmpty(CQueue \*Q) {

return (Q->front == -1);

}

// Function to check if the queue is full

int isFull(CQueue \*Q) {

return ((Q->rear + 1) % MAX == Q->front);

}

// Function to insert an element at the rear of the queue

void enQueueRear(CQueue \*Q, int x) {

if (isFull(Q)) {

printf("Queue overflow\n");

return;

}

if (isEmpty(Q)) {

Q->front = 0;

}

Q->rear = (Q->rear + 1) % MAX;

Q->A[Q->rear] = x;

}

// Function to insert an element at the front of the queue

void enQueueFront(CQueue \*Q, int x) {

if (isFull(Q)) {

printf("Queue overflow\n");

return;

}

if (isEmpty(Q)) {

Q->front = 0;

Q->rear = 0;

} else {

Q->front = (Q->front - 1 + MAX) % MAX;

}

Q->A[Q->front] = x;

}

// Function to delete an element from the front of the queue

int deQueueFront(CQueue \*Q) {

if (isEmpty(Q)) {

printf("Queue underflow\n");

return -1;

}

int x = Q->A[Q->front];

if (Q->front == Q->rear) {

Q->front = -1;

Q->rear = -1;

} else {

Q->front = (Q->front + 1) % MAX;

}

return x;

}

// Function to delete an element from the rear of the queue

int deQueueRear(CQueue \*Q) {

if (isEmpty(Q)) {

printf("Queue underflow\n");

return -1;

}

int x = Q->A[Q->rear];

if (Q->front == Q->rear) {

Q->front = -1;

Q->rear = -1;

} else {

Q->rear = (Q->rear - 1 + MAX) % MAX;

}

return x;

}

// Function to display the queue

void displayQueue(CQueue \*Q) {

if (isEmpty(Q)) {

printf("Queue is empty\n");

return;

}

int i = Q->front;

while (i != Q->rear) {

printf("%d ", Q->A[i]);

i = (i + 1) % MAX;

}

printf("%d\n", Q->A[Q->rear]);

}

int main() {

CQueue Q;

Q.front = -1;

Q.rear = -1;

// Enqueue some elements

enQueueRear(&Q, 1);

enQueueRear(&Q, 2);

enQueueFront(&Q, 3);

// Display the queue

displayQueue(&Q);

// Dequeue from rear and front

int x = deQueueRear(&Q);

printf("Dequeued from rear: %d\n", x);

x = deQueueFront(&Q);

printf("Dequeued from front: %d\n", x);

// Display the queue

displayQueue(&Q);

return 0;

}

Q. Write a Program to implement multiple two queues using array and perform following operations on it. A.Add, B.Delete, C.DisplayQueue.

#include <stdio.h>

#define MAX\_SIZE 100 // Maximum size of each queue

#define NUM\_QUEUES 2 // Number of queues

typedef struct Queue {

int front;

int rear;

int size;

int elements[MAX\_SIZE];

} Queue;

Queue queues[NUM\_QUEUES];

void initializeQueues() {

for (int i = 0; i < NUM\_QUEUES; i++) {

queues[i].front = -1;

queues[i].rear = -1;

queues[i].size = 0;

}

}

int isQueueEmpty(int queueId) {

return queues[queueId].size == 0;

}

int isQueueFull(int queueId) {

return queues[queueId].size == MAX\_SIZE;

}

int enqueue(int queueId) {

int element;

// Get input from user

printf("Enter element to enqueue in queue %d: ", queueId);

scanf("%d", &element);

if (isQueueFull(queueId)) {

printf("Queue %d is full!\n", queueId);

return 0;

}

if (isQueueEmpty(queueId)) {

queues[queueId].front = 0;

queues[queueId].rear = 0;

} else {

queues[queueId].rear = (queues[queueId].rear + 1) % MAX\_SIZE;

}

queues[queueId].elements[queues[queueId].rear] = element;

queues[queueId].size++;

printf("Element %d enqueued successfully in queue %d.\n", element, queueId);

return 1;

}

int dequeue(int queueId) {

if (isQueueEmpty(queueId)) {

printf("Queue %d is empty!\n", queueId);

return -1;

}

int element = queues[queueId].elements[queues[queueId].front];

if (queues[queueId].front == queues[queueId].rear) {

queues[queueId].front = -1;

queues[queueId].rear = -1;

} else {

queues[queueId].front = (queues[queueId].front + 1) % MAX\_SIZE;

}

queues[queueId].size--;

printf("Element %d dequeued from queue %d.\n", element, queueId);

return element;

}

void displayQueue(int queueId) {

if (isQueueEmpty(queueId)) {

printf("Queue %d is empty!\n", queueId);

return;

}

int i = queues[queueId].front;

while (i != queues[queueId].rear) {

printf("%d ", queues[queueId].elements[i]);

i = (i + 1) % MAX\_SIZE;

}

printf("%d\n", queues[queueId].elements[queues[queueId].rear]);

}

int main() {

initializeQueues();

int choice, queueId, element;

while (1) {

printf("\n1. Enqueue\n2. Dequeue\n3. Display Queue\n4. Exit\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter queue ID (0-%d): ", NUM\_QUEUES - 1);

scanf("%d", &queueId);

enqueue(queueId);

break;

case 2:

printf("Enter queue ID (0-%d): ", NUM\_QUEUES - 1);

scanf("%d", &queueId);

dequeue(queueId);

break;

case 3:

printf("Enter queue ID (0-%d): ", NUM\_QUEUES - 1);

scanf("%d", &queueId);

displayQueue(queueId);

break;

case 4:

exit(0);

break;

default:

printf("Invalid choice!\n");

break;

}

}

return 0;

}

Q. WAP to perform addition of two polynomials using singly linked list. in C

#include <stdio.h>

#include <stdlib.h>

struct Node {

int coef;

int exp;

struct Node\* next;

};

typedef struct Node Node;

void insert(Node\*\* poly, int coef, int exp) {

Node\* temp = (Node\*) malloc(sizeof(Node));

temp->coef = coef;

temp->exp = exp;

temp->next = NULL;

if (\*poly == NULL) {

\*poly = temp;

return;

}

Node\* current = \*poly;

while (current->next != NULL) {

current = current->next;

}

current->next = temp;

}

void print(Node\* poly) {

if (poly == NULL) {

printf("0\n");

return;

}

Node\* current = poly;

while (current != NULL) {

printf("%dx^%d", current->coef, current->exp);

if (current->next != NULL) {

printf(" + ");

}

current = current->next;

}

printf("\n");

}

Node\* add(Node\* poly1, Node\* poly2) {

Node\* result = NULL;

while (poly1 != NULL && poly2 != NULL) {

if (poly1->exp == poly2->exp) {

insert(&result, poly1->coef + poly2->coef, poly1->exp);

poly1 = poly1->next;

poly2 = poly2->next;

} else if (poly1->exp > poly2->exp) {

insert(&result, poly1->coef, poly1->exp);

poly1 = poly1->next;

} else {

insert(&result, poly2->coef, poly2->exp);

poly2 = poly2->next;

}

}

while (poly1 != NULL) {

insert(&result, poly1->coef, poly1->exp);

poly1 = poly1->next;

}

while (poly2 != NULL) {

insert(&result, poly2->coef, poly2->exp);

poly2 = poly2->next;

}

return result;

}

int main() {

Node\* poly1 = NULL;

insert(&poly1, 5, 4);

insert(&poly1, 3, 2);

insert(&poly1, 1, 0);

Node\* poly2 = NULL;

insert(&poly2, 4, 4);

insert(&poly2, 2, 2);

insert(&poly2, 1, 1);

printf("First polynomial: ");

print(poly1);

printf("Second polynomial: ");

print(poly2);

Node\* result = add(poly1, poly2);

printf("Result: ");

print(result);

return 0;

}

Q. Write an iterative Reverse() function that reverses a list by rearranging all the next pointers and the head pointer. Ideally, Reverse() should only need to make one pass of the list. in C

#include <stdio.h>

typedef struct Node {

int data;

struct Node \*next;

} Node;

// Function to reverse a linked list iteratively

Node \*Reverse(Node \*head) {

Node \*prev = NULL, \*curr = head, \*next;

while (curr) {

// Store the next node before modifying it

next = curr->next;

// Reverse the next pointer

curr->next = prev;

// Update the previous and current pointers

prev = curr;

curr = next;

}

return prev; // The last node becomes the new head

// Utility function to print a linked list

void PrintList(Node \*head) {

Node \*temp = head;

while (temp) {

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

temp = temp->next;

}

printf("NULL\n");

}

int main() {

// Create a sample linked list

Node \*head = (Node \*)malloc(sizeof(Node));

head->data = 1;

head->next = (Node \*)malloc(sizeof(Node));

head->next->data = 2;

head->next->next = (Node \*)malloc(sizeof(Node));

head->next->next->data = 3;

head->next->next->next = NULL;

// Print the original list

printf("Original list: ");

PrintList(head);

// Reverse the list

head = Reverse(head);

// Print the reversed list

printf("Reversed list: ");

PrintList(head);

return 0;

}

Q. WAP to create doubly linked list and perform following operations on it. A)Insert (all case) 2.Delete(all cases). in C

// C program for the all operations in

// the Doubly Linked List

#include <stdio.h>

#include <stdlib.h>

// Linked List Node

struct node {

int info;

struct node \*prev, \*next;

};

struct node\* start = NULL;

// Function to traverse the linked list

void traverse()

{

// List is empty

if (start == NULL) {

printf("\nList is empty\n");

return;

}

// Else print the Data

struct node\* temp;

temp = start;

while (temp != NULL) {

printf("Data = %d\n", temp->info);

temp = temp->next;

}

}

// Function to insert at the front

// of the linked list

void insertAtFront()

{

int data;

struct node\* temp;

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

printf("\nEnter number to be inserted: ");

scanf("%d", &data);

temp->info = data;

temp->prev = NULL;

// Pointer of temp will be

// assigned to start

temp->next = start;

start = temp;

}

// Function to insert at the end of

// the linked list

void insertAtEnd()

{

int data;

struct node \*temp, \*trav;

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

temp->prev = NULL;

temp->next = NULL;

printf("\nEnter number to be inserted: ");

scanf("%d", &data);

temp->info = data;

temp->next = NULL;

trav = start;

// If start is NULL

if (start == NULL) {

start = temp;

}

// Changes Links

else {

while (trav->next != NULL)

trav = trav->next;

temp->prev = trav;

trav->next = temp;

}

}

// Function to insert at any specified

// position in the linked list

void insertAtPosition()

{

int data, pos, i = 1;

struct node \*temp, \*newnode;

newnode = malloc(sizeof(struct node));

newnode->next = NULL;

newnode->prev = NULL;

// Enter the position and data

printf("\nEnter position : ");

scanf("%d", &pos);

// If start==NULL,

if (start == NULL) {

start = newnode;

newnode->prev = NULL;

newnode->next = NULL;

}

// If position==1,

else if (pos == 1) {

// this is author method its correct but we can simply call insertAtfront() function for this special case

/\* newnode->next = start;

newnode->next->prev = newnode;

newnode->prev = NULL;

start = newnode; \*/

// now this is improved by Jay Ghughriwala on geeksforgeeks

insertAtFront();

}

// Change links

else {

printf("\nEnter number to be inserted: ");

scanf("%d", &data);

newnode->info = data;

temp = start;

while (i < pos - 1) {

temp = temp->next;

i++;

}

newnode->next = temp->next;

newnode->prev = temp;

temp->next = newnode;

temp->next->prev = newnode;

}

}

// Function to delete from the front

// of the linked list

void deleteFirst()

{

struct node\* temp;

if (start == NULL)

printf("\nList is empty\n");

else {

temp = start;

start = start->next;

if (start != NULL)

start->prev = NULL;

free(temp);

}

}

// Function to delete from the end

// of the linked list

void deleteEnd()

{

struct node\* temp;

if (start == NULL)

printf("\nList is empty\n");

temp = start;

while (temp->next != NULL)

temp = temp->next;

if (start->next == NULL)

start = NULL;

else {

temp->prev->next = NULL;

free(temp);

}

}

// Function to delete from any specified

// position from the linked list

void deletePosition()

{

int pos, i = 1;

struct node \*temp, \*position;

temp = start;

// If DLL is empty

if (start == NULL)

printf("\nList is empty\n");

// Otherwise

else {

// Position to be deleted

printf("\nEnter position : ");

scanf("%d", &pos);

// If the position is the first node

if (pos == 1) {

deleteFirst(); // im,proved by Jay Ghughriwala on GeeksforGeeks

if (start != NULL) {

start->prev = NULL;

}

free(position);

return;

}

// Traverse till position

while (i < pos - 1) {

temp = temp->next;

i++;

}

// Change Links

position = temp->next;

if (position->next != NULL)

position->next->prev = temp;

temp->next = position->next;

// Free memory

free(position);

}

}

// Driver Code

int main()

{

int choice;

while (1) {

printf("\n\t1 To see list\n");

printf("\t2 For insertion at"

" starting\n");

printf("\t3 For insertion at"

" end\n");

printf("\t4 For insertion at "

"any position\n");

printf("\t5 For deletion of "

"first element\n");

printf("\t6 For deletion of "

"last element\n");

printf("\t7 For deletion of "

"element at any position\n");

printf("\t8 To exit\n");

printf("\nEnter Choice :\n");

scanf("%d", &choice);

switch (choice) {

case 1:

traverse();

break;

case 2:

insertAtFront();

break;

case 3:

insertAtEnd();

break;

case 4:

insertAtPosition();

break;

case 5:

deleteFirst();

break;

case 6:

deleteEnd();

break;

case 7:

deletePosition();

break;

case 8:

exit(1);

break;

default:

printf("Incorrect Choice. Try Again \n");

continue;

}

}

return 0;

}

Q. WAP to merge two sorted Doubly linked lists and display their result. in C

#include <stdio.h>

#include <stdlib.h>

/\* Link list Node \*/

struct Node {

int key;

struct Node\* next;

};

struct Node\* newNode(int key) {

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

temp->key = key;

temp->next = NULL;

return temp;

}

int main() {

/\* Let us create two sorted linked lists to test

the above functions. Created lists shall be

a: 5->10->15->40

b: 2->3->20 \*/

struct Node\* a = newNode(5);

a->next = newNode(10);

a->next->next = newNode(15);

a->next->next->next = newNode(40);

struct Node\* b = newNode(2);

b->next = newNode(3);

b->next->next = newNode(20);

int v[100]; // Assuming a maximum size for the array

int i = 0;

while (a != NULL) {

v[i++] = a->key;

a = a->next;

}

while (b != NULL) {

v[i++] = b->key;

b = b->next;

}

int n = i;

// Sorting the array

for (i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (v[j] > v[j + 1]) {

int temp = v[j];

v[j] = v[j + 1];

v[j + 1] = temp;

}

}

}

struct Node\* result = newNode(-1);

struct Node\* temp = result;

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

temp->next = newNode(v[i]);

temp = temp->next;

}

temp = result->next;

printf("Resultant Merge Linked List Is :\n");

while (temp != NULL) {

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

temp = temp->next;

}

return 0;

}

Q. Implement Push and POP operations of STACK in Doubly linked lists in C

#include <stdio.h>

#define STACK\_SIZE 100

typedef struct Node {

int data;

struct Node \*next;

struct Node \*prev;

} Node;

Node \*top = NULL;

int isEmpty() {

return top == NULL;

}

void push() {

if (top == NULL) {

Node \*newNode = (Node \*)malloc(sizeof(Node));

printf("Enter data: ");

scanf("%d", &newNode->data);

newNode->next = NULL;

newNode->prev = NULL;

top = newNode;

} else {

Node \*newNode = (Node \*)malloc(sizeof(Node));

printf("Enter data: ");

scanf("%d", &newNode->data);

newNode->next = top;

newNode->prev = NULL;

top->prev = newNode;

top = newNode;

}

}

int pop() {

if (isEmpty()) {

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

return -1;

} else {

int data = top->data;

Node \*temp = top;

top = top->next;

if (top) {

top->prev = NULL;

}

free(temp);

return data;

}

}

int main() {

int choice;

while (1) {

printf("\n1. Push\n2. Pop\n3. Exit\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

push();

break;

case 2:

printf("Popped element: %d\n", pop());

break;

case 3:

exit(0);

break;

default:

printf("Invalid choice!\n");

break;

}

}

return 0;

}

Q. Implement ADD and DELETE operations of QUEUE on Doubly linked lists in C

#include <stdio.h>

#define QUEUE\_SIZE 100

typedef struct Node {

int data;

struct Node \*next;

struct Node \*prev;

} Node;

Node \*front = NULL;

Node \*rear = NULL;

int isEmpty() {

return front == NULL;

}

void add() {

int data;

printf("Enter data: ");

scanf("%d", &data);

Node \*newNode = (Node \*)malloc(sizeof(Node));

newNode->data = data;

newNode->next = NULL;

if (isEmpty()) {

front = newNode;

rear = newNode;

newNode->prev = NULL;

} else {

rear->next = newNode;

newNode->prev = rear;

rear = newNode;

}

}

int delete() {

if (isEmpty()) {

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

return -1;

} else {

Node \*temp = front;

int data = temp->data;

front = front->next;

if (front) {

front->prev = NULL;

} else {

rear = NULL;

}

free(temp);

return data;

}

}

int main() {

int choice;

while (1) {

printf("\n1. Add\n2. Delete\n3. Exit\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

add();

break;

case 2:

printf("Deleted element: %d\n", delete());

break;

case 3:

exit(0);

break;

default:

printf("Invalid choice!\n");

break;

}

}

return 0;

}