|  |
| --- |
|  |
| EXPERIMENT 7 |
|  |
|  |
| **Vansh Sukhija**  **12112021** |
|  |

|  |
| --- |
|  |

Ans 1(i)-

#include<stdio.h>

#include<stdlib.h>

typedef *struct* stack{

*int* size, top, \*s;

}stack;

*void* push(stack \**st*, *int* *data*){

    if(*st*->top == *st*->size-1)

        printf("Stack Overflown\n");

    else{

*st*->top++;

*st*->s[*st*->top] = *data*;

    }

}

*int* pop(stack \**st*){

*int* x = -1;

    if(*st*->top == -1)

        printf("Stack Underflown\n");

    else{

        x = *st*->s[*st*->top];

*st*->top--;

    }

    return x;

}

*int* peek(stack *st*, *int* *pos*){

*int* x = -1;

    if(*st*.top - *pos* + 1 < 0)

        printf("Invalid Position\n");

    else{

        x = *st*.s[*st*.top - *pos* + 1];

    }

    return x;

}

*int* stacktop(stack *st*){

    if(*st*.top == -1)

        return -1;

    else

        return *st*.s[*st*.top];

}

*int* isEmpty(stack *st*){

    if(*st*.top == -1)

        return 1;

    else

        return 0;

}

*int* isFull(stack *st*){

    if(*st*.top == *st*.size - 1)

        return 1;

    else

        return 0;

}

*void* display(stack *st*){

    printf("Stack: ");

    for(*int* i=0; i<=*st*.top; i++){

        printf("%d ", *st*.s[i]);

    }

    printf("\n");

}

*void* main(){

    stack st;

    printf("Enter size of the stack: ");

    scanf("%d", &st.size);

    st.top = -1;

    st.s = (*int* \*)malloc(sizeof(*int*)\*st.size);

    push(&st, 10);

    push(&st, 20);

    push(&st, 30);

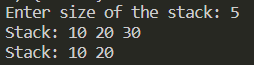
    display(st);

    pop(&st);

    display(st);

}

Output-



Ans 1(ii)-

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef *struct* Stack{

*int* size, top;

*int* \*s;

}stack;

*void* push(stack \**st*, *char* *data*){

    if(*st*->top == *st*->size-1)

        printf("Stack Overflown\n");

    else{

*st*->top++;

*st*->s[*st*->top] = *data*;

    }

}

*int* pop(stack \**st*){

*int* x = -1;

    if(*st*->top == -1)

        printf("Stack Underflown\n");

    else{

        x = *st*->s[*st*->top];

*st*->top--;

    }

    return x;

}

*int* peek(stack *st*, *int* *pos*){

*int* x = -1;

    if(*st*.top - *pos* + 1 < 0)

        printf("Invalid Position\n");

    else{

        x = *st*.s[*st*.top - *pos* + 1];

    }

    return x;

}

*int* stacktop(stack *st*){

    if(*st*.top == -1)

        return -1;

    else

        return *st*.s[*st*.top];

}

*int* isEmpty(stack *st*){

    if(*st*.top == -1)

        return 1;

    else

        return 0;

}

*int* isFull(stack *st*){

    if(*st*.top == *st*.size - 1)

        return 1;

    else

        return 0;

}

*int* isBalance(*char* \**exp*){

    stack st;

    st.size = sizeof(*exp*);

    st.top = -1;

    st.s = (*int* \*)malloc(sizeof(*int*)\*st.size);

    for(*int* i=0; *exp*[i]!=0; i++){

        if(*exp*[i]=='(')

            push(&st, *exp*[i]);

        else if(*exp*[i]==')'){

            if(isEmpty(st))

                return 0;

            pop(&st);

        }

    }

    return isEmpty(st)?1:0;

}

*int* prec(*char* *x*){

    if(*x*=='+' || *x*=='-')

        return 1;

    else if(*x*=='\*' || *x*=='/')

        return 2;

}

*int* isOperand(*char* *x*){

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

        return 0;

    else

        return 1;

}

*int* Eval(*char* \**postfix*){

    stack st;

    st.size = strlen(*postfix*);

    st.top = -1;

    st.s = (*int* \*)malloc(sizeof(*int*)\*st.size);

*int* i, x1, x2, r;

    for(i=0; *postfix*[i]!=0; i++){

        if(isOperand(*postfix*[i]))

            push(&st, *postfix*[i]-48);

        else{

            x2 = pop(&st);

            x1 = pop(&st);

            switch(*postfix*[i]){

                case '+':

                    r = x1+x2;

                    push(&st, r);

                    break;

                case '-':

                    r = x1-x2;

                    push(&st, r);

                    break;

                case '/':

                    r = x1/x2;

                    push(&st, r);

                    break;

                case '\*':

                    r = x1\*x2;

                    push(&st, r);

                    break;

            }

        }

    }

    return pop(&st);

}

*void* main(){

*char* postfix[20];

    printf("Enter expression: ");

    gets(postfix);

    printf("Evaluation: %d", Eval(postfix));

}

Output-



Ans 2(i)-

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef *struct* Stack{

*int* size, top;

*char* \*s;

}stack;

*void* push(stack \**st*, *char* *data*){

    if(*st*->top == *st*->size-1)

        printf("Stack Overflown\n");

    else{

*st*->top++;

*st*->s[*st*->top] = *data*;

    }

}

*int* pop(stack \**st*){

*int* x = -1;

    if(*st*->top == -1)

        printf("Stack Underflown\n");

    else{

        x = *st*->s[*st*->top];

*st*->top--;

    }

    return x;

}

*int* isEmpty(stack *st*){

    if(*st*.top == -1)

        return 1;

    else

        return 0;

}

*int* palindrome(*char* *str*[]){

    stack st;

    st.top = -1;

    st.size = strlen(*str*);

    st.s = (*char* \*)malloc(sizeof(*char*)\*st.size);

*int* i=0;

    while(i<strlen(*str*)/2){

        push(&st, *str*[i]);

        i++;

    }

    if(strlen(*str*)&1)

        i++;

    while(*str*[i]!=0){

        if(*str*[i]==st.s[st.top])

            pop(&st);

        else

            break;

        i++;

    }

    return isEmpty(st);

}

*void* main(){

*char* str[20];

    printf("Enter a string: ");

    gets(str);

    printf("String: ");

    puts(str);

    if(palindrome(str))

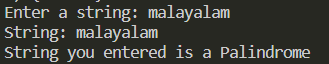
        printf("String you entered is a Palindrome\n");

    else

        printf("String you entered is not a Palindrome\n");

}

Output-



Ans 2(ii)-

#include<stdio.h>

#include<stdlib.h>

typedef *struct* Stack{

*int* size, top;

*char* \*s;

}stack;

*void* push(stack \**st*, *int* *data*){

    if(*st*->top == *st*->size-1)

        printf("Stack Overflown\n");

    else{

*st*->top++;

*st*->s[*st*->top] = *data*;

    }

}

*int* pop(stack \**st*){

*int* x = -1;

    if(*st*->top == -1)

        printf("Stack Underflown\n");

    else{

        x = *st*->s[*st*->top];

*st*->top--;

    }

    return x;

}

*int* peek(stack *st*, *int* *pos*){

*int* x = -1;

    if(*st*.top - *pos* + 1 < 0)

        printf("Invalid Position\n");

    else{

        x = *st*.s[*st*.top - *pos* + 1];

    }

    return x;

}

*int* stacktop(stack *st*){

    if(*st*.top == -1)

        return -1;

    else

        return *st*.s[*st*.top];

}

*int* isEmpty(stack *st*){

    if(*st*.top == -1)

        return 1;

    else

        return 0;

}

*int* isFull(stack *st*){

    if(*st*.top == *st*.size - 1)

        return 1;

    else

        return 0;

}

*int* isBalance(*char* \**exp*){

    stack st;

    st.size = sizeof(*exp*);

    st.top = -1;

    st.s = (*char* \*)malloc(sizeof(*char*)\*st.size);

    for(*int* i=0; *exp*[i]!=0; i++){

        if(*exp*[i]=='(')

            push(&st, *exp*[i]);

        else if(*exp*[i]==')'){

            if(isEmpty(st))

                return 0;

            pop(&st);

        }

    }

    return isEmpty(st)?1:0;

}

*void* main(){

*char* exp[20];

    printf("Enter the expression: ");

    gets(exp);

    if(isBalance(exp))

        printf("Correctly paranthesized\n");

    else

        printf("Not paranthesized correctly\n");

}

Output-



Ans 4(i)-

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef *struct* Stack{

*int* size, top;

*char* \*s;

}stack;

*void* push(stack \**st*, *char* *data*){

    if(*st*->top == *st*->size-1)

        printf("Stack Overflown\n");

    else{

*st*->top++;

*st*->s[*st*->top] = *data*;

    }

}

*char* pop(stack \**st*){

*char* x = 0;

    if(*st*->top == -1)

        printf("Stack Underflown\n");

    else{

        x = *st*->s[*st*->top];

*st*->top--;

    }

    return x;

}

*char* peek(stack *st*, *int* *pos*){

*char* x = 0;

    if(*st*.top - *pos* + 1 < 0)

        printf("Invalid Position\n");

    else{

        x = *st*.s[*st*.top - *pos* + 1];

    }

    return x;

}

*char* stacktop(stack *st*){

    if(*st*.top == -1)

        return -1;

    else

        return *st*.s[*st*.top];

}

*int* isEmpty(stack *st*){

    if(*st*.top == -1)

        return 1;

    else

        return 0;

}

*int* isFull(stack *st*){

    if(*st*.top == *st*.size - 1)

        return 1;

    else

        return 0;

}

*int* isBalance(*char* \**exp*){

    stack st;

    st.size = sizeof(*exp*);

    st.top = -1;

    st.s = (*char* \*)malloc(sizeof(*char*)\*st.size);

    for(*int* i=0; *exp*[i]!=0; i++){

        if(*exp*[i]=='(')

            push(&st, *exp*[i]);

        else if(*exp*[i]==')'){

            if(isEmpty(st))

                return 0;

            pop(&st);

        }

    }

    return isEmpty(st)?1:0;

}

*int* prec(*char* *x*){

    if(*x*=='+' || *x*=='-')

        return 1;

    else if(*x*=='\*' || *x*=='/')

        return 2;

}

*int* isOperand(*char* *x*){

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

        return 0;

    else

        return 1;

}

*char* \*intopost(*char* \**infix*){

*int* len = strlen(*infix*);

    stack st;

    st.top = -1;

    st.size = len;

    st.s = (*char* \*)malloc(sizeof(*char*)\*len);

*char* \*postfix = (*char* \*)malloc(sizeof(*char*)\*(len+1));

*int* i=0, j=0;

    while(*infix*[i]!=0){

        if(isOperand(*infix*[i]))

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

        else{

            if(prec(*infix*[i]) > prec(stacktop(st)))

                push(&st, *infix*[i++]);

            else

                postfix[j++] = pop(&st);

        }

    }

    while(!isEmpty(st)){

        postfix[j++] = pop(&st);

    }

    postfix[j] = 0;

    return postfix;

}

*void* main(){

*char* infix[20];

    printf("Enter expression: ");

    gets(infix);

*char* \*postfix = intopost(infix);

    puts(postfix);

}

Output-



Ans 4(ii)-

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

*struct* Stack{

*int* top, size;

*int* \*s;

};

*struct* Stack\* create(*int* *max*){

*struct* Stack\* stack = (*struct* Stack\*)malloc(sizeof(*struct* Stack));

    stack->size = *max*;

    stack->top = -1;

    stack->s = (*int*\*)malloc(stack->size \* sizeof(*int*));

    return stack;

}

*int* isFull(*struct* Stack\* *stack*){

    if(*stack*->top == *stack*->size - 1)

        printf("Will not be able to push size reached\n");

    return *stack*->top == *stack*->size - 1;

}

*int* isEmpty(*struct* Stack\* *stack*){

    return *stack*->top == -1;

}

*void* push(*struct* Stack\* *stack*, *char* *item*){

    if (isFull(*stack*))

        return;

*stack*->s[++*stack*->top] = *item*;

}

*int* pop(*struct* Stack\* *stack*){

    if (isEmpty(*stack*))

        return 0;

    return *stack*->s[*stack*->top--];

}

*int* peek(*struct* Stack\* *stack*){

    if (isEmpty(*stack*))

        return 0;

    return *stack*->s[*stack*->top];

}

*int* checkIfOperand(*char* *ch*){

    return (*ch* >= 'a' && *ch* <= 'z') || (*ch* >= 'A' && *ch* <= 'Z');

}

*int* precedence(*char* *ch*){

    switch(*ch*){

        case '+':

            return 1;

        case '-':

            return 1;

        case '\*':

            return 2;

        case '/':

            return 2;

        case '^':

            return 3;

    }

    return -1;

}

*int* getPostfix(*char*\* *expression*){

*int* i, j;

*struct* Stack\* stack = create(strlen(*expression*));

    if(!stack)

        return -1 ;

    for(i = 0, j = -1; *expression*[i]; ++i)    {

        if (checkIfOperand(*expression*[i]))

*expression*[++j] = *expression*[i];

        else if (*expression*[i] == '(')

            push(stack, *expression*[i]);

        else if (*expression*[i] == ')'){

            while (!isEmpty(stack) && peek(stack) != '(')

*expression*[++j] = pop(stack);

            if (!isEmpty(stack) && peek(stack) != '(')

                return -1;

            else

                pop(stack);

        }

        else{

            while (!isEmpty(stack) && precedence(*expression*[i]) <= precedence(peek(stack)))

*expression*[++j] = pop(stack);

            push(stack, *expression*[i]);

        }

    }

    while (!isEmpty(stack))

*expression*[++j] = pop(stack);

*expression*[++j] = '\0';

}

*void* reverse(*char* \**exp*){

*int* size = strlen(*exp*);

*int* j = size, i=0;

*char* temp[size];

    temp[j--]='\0';

    while(*exp*[i]!='\0'){

        temp[j] = *exp*[i];

        j--;

        i++;

    }

    strcpy(*exp*,temp);

}

*void* brackets(*char*\* *exp*){

*int* i = 0;

    while(*exp*[i]!='\0'){

        if(*exp*[i]=='(')

*exp*[i]=')';

        else if(*exp*[i]==')')

*exp*[i]='(';

        i++;

    }

}

*void* InfixtoPrefix(*char* \**exp*){

*int* size = strlen(*exp*);

    reverse(*exp*);

    brackets(*exp*);

    getPostfix(*exp*);

    reverse(*exp*);

}

*int* main(){

    printf("Infix: ");

*char* expression[] = "((a/b)+c)-(d+(e\*f))";

    printf("%s\n",expression);

    InfixtoPrefix(expression);

    printf("Prefix: ");

    printf("%s\n",expression);

    return 0;

}

Output-



Ans 5-

#include <bits/stdc++.h>

using *namespace* std;

*bool* check(stack<*int*> &*a*, stack<*int*> &*b*)

{

    if (*a*.empty())

    {

*a*.push(*b*.top());

*b*.pop();

        return false;

    }

    else if (*b*.empty())

    {

*b*.push(*a*.top());

*a*.pop();

        return true;

    }

    else if (*b*.top() > *a*.top())

    {

*b*.push(*a*.top());

*a*.pop();

        return true;

    }

    else

    {

*a*.push(*b*.top());

*b*.pop();

        return false;

    }

}

*int* main()

{

*int* n;

    cout << "Enter N: ";

    cin >> n;

    stack<*int*> a, b, c;

    for (*int* i = n; i >= 1; i--)

    {

        a.push(i);

    }

*int* i = 0;

    if (n % 2 == 0)

    {

        while (i < pow(2, n))

        {

            if (i % 3 == 0)

            {

                if (check(a, c))

                {

                    cout << "move from a to c" << endl;

                }

                else

                {

                    cout << "move from c to a" << endl;

                }

            }

            else if (i % 3 == 1)

            {

                if (check(a, b))

                {

                    cout << "move from a to b" << endl;

                }

                else

                {

                    cout << "move from b to a" << endl;

                }

            }

            else

            {

                if (check(b, c))

                {

                    cout << "move from b to c" << endl;

                }

                else

                {

                    cout << "move from c to b" << endl;

                }

            }

            i++;

        }

    }

    else

    {

        while (i < pow(2, n))

        {

            if (i % 3 == 0)

            {

                if (check(a, b))

                {

                    cout << "move from a to b" << endl;

                }

                else

                {

                    cout << "move from b to a" << endl;

                }

            }

            else if (i % 3 == 1)

            {

                if (check(a, c))

                {

                    cout << "move from a to c" << endl;

                }

                else

                {

                    cout << "move from c to a" << endl;

                }

            }

            else

            {

                if (check(b, c)){

                    cout << "move from b to c" << endl;

                }

                else{

                    cout << "move from c to b" << endl;

                }

            }

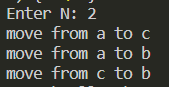
            i++;

        }

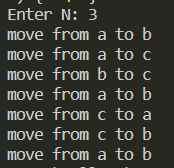
    }

}

Output(i)-



Output(ii)-



Ans 6(i)-

#include<stdio.h>

#include<stdlib.h>

typedef *struct* Queue{

*int* size, front, rear, \*q;

}queue;

queue create(*int* *size*){

    queue q;

    q.size = *size*;

    q.front = q.rear = -1;

    q.q = (*int* \*)malloc(sizeof(*int*)\*q.size);

    return q;

}

*void* enqueue(queue \**q*, *int* *x*){

    if(*q*->rear == *q*->size-1)

        printf("Queue is Full\n");

    else{

*q*->rear++;

*q*->q[*q*->rear] = *x*;

    }

}

*int* dequeue(queue \**q*){

*int* x = -1;

    if(*q*->front == *q*->rear)

        printf("Queue is Empty\n");

    else{

*q*->front++;

        x = *q*->q[*q*->front];

    }

    return x;

}

*int* isEmpty(queue *q*){

    return (*q*.front==*q*.rear);

}

*int* isFull(queue *q*){

    return (*q*.rear == *q*.size-1);

}

*void* display(queue *q*){

    printf("Queue: ");

    while(*q*.front < *q*.rear){

*q*.front++;

        printf("%d ", *q*.q[*q*.front]);

    }

    printf("\n");

}

*void* main(){

*int* n;

    printf("Enter the size of the queue: ");

    scanf("%d", &n);

    queue qu = create(n);

    enqueue(&qu, 10);

    display(qu);

    enqueue(&qu, 20);

    display(qu);

    enqueue(&qu, 30);

    display(qu);

    enqueue(&qu, 40);

    display(qu);

    dequeue(&qu);

    display(qu);

    dequeue(&qu);

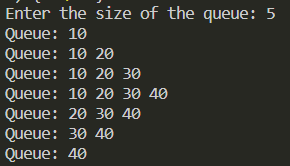
    display(qu);

    dequeue(&qu);

    display(qu);

}

Output-



Ans 6(ii)-

#include<stdio.h>

#include<stdlib.h>

typedef *struct* stack{

*int* size, top, \*s;

}stack;

stack st1, st2;

stack create(*int* *size*){

    stack st;

    st.size = *size*;

    st.top = -1;

    st.s = (*int* \*)malloc(sizeof(*int*)\*st.size);

    return st;

}

*void* push(stack \**st*, *int* *data*){

    if(*st*->top == *st*->size-1)

        printf("Stack Overflown\n");

    else{

*st*->top++;

*st*->s[*st*->top] = *data*;

    }

}

*int* pop(stack \**st*){

*int* x = -1;

    if(*st*->top == -1)

        printf("Stack Underflown\n");

    else{

        x = *st*->s[*st*->top];

*st*->top--;

    }

    return x;

}

*void* enqueue(*int* *key*){

    if(st2.top!=-1)

        while(st2.top!=-1)

            push(&st1, pop(&st2));

    push(&st1, *key*);

}

*int* dequeue(){

*int* x;

    if(st2.top==-1 && st1.top==-1)

        x=0;

    else if(st1.top==-1 && st2.top!=-1)

        x = pop(&st2);

    else{

        while(st1.top!=-1)

            push(&st2, pop(&st1));

        x = pop(&st2);

    }

    while(st2.top!=-1)

        push(&st1, pop(&st2));

    return x;

}

*void* display(stack *st*){

    printf("Stack: ");

    for(*int* i=*st*.top; i>=0; i--){

        printf("%d ", *st*.s[i]);

    }

    printf("\n");

}

*void* main(){

*int* n;

    printf("Enter the size of stack: ");

    scanf("%d", &n);

    st1 = create(n);

    st2 = create(n);

    enqueue(10);

    display(st1);

    enqueue(20);

    display(st1);

    enqueue(30);

    display(st1);

    enqueue(40);

    display(st1);

    dequeue();

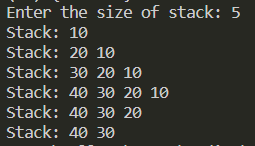
    display(st1);

    dequeue();

    display(st1);

}

Output-



Ans 7(ii)-

#include<stdio.h>

#include<stdlib.h>

typedef *struct* Queue{

*int* size, front, rear, \*q;

}queue;

queue create(*int* *size*){

    queue q;

    q.size = *size*;

    q.front = q.rear = -1;

    q.q = (*int* \*)malloc(sizeof(*int*)\*q.size);

    return q;

}

*void* enqueuelast(queue \**q*, *int* *x*){

    if(*q*->rear == *q*->size-1)

        printf("Queue is Full\n");

    else{

*q*->rear++;

*q*->q[*q*->rear] = *x*;

    }

}

*void* enqueuefront(queue \**q*, *int* *x*){

    if(*q*->front == -1)

        printf("Queue is full\n");

    else{

*q*->q[*q*->front] = *x*;

*q*->front--;

    }

}

*int* dequeuefront(queue \**q*){

*int* x = -1;

    if(*q*->front == *q*->rear)

        printf("Queue is Empty\n");

    else{

*q*->front++;

        x = *q*->q[*q*->front];

    }

    return x;

}

*int* dequeuelast(queue \**q*){

*int* x = -1;

    if(*q*->rear == *q*->front)

        printf("Queue is Empty\n");

    else{

        x = *q*->q[*q*->rear];

*q*->rear--;

    }

    return x;

}

*void* display(queue *q*){

    printf("Queue: ");

    while(*q*.front < *q*.rear){

*q*.front++;

        printf("%d ", *q*.q[*q*.front]);

    }

    printf("\n");

}

*void* main(){

*int* n;

    printf("Enter the size of the queue: ");

    scanf("%d", &n);

    queue qu = create(n);

    enqueuelast(&qu, 10);

    display(qu);

    enqueuelast(&qu, 20);

    display(qu);

    enqueuelast(&qu, 30);

    display(qu);

    dequeuefront(&qu);

    display(qu);

    dequeuefront(&qu);

    display(qu);

    enqueuefront(&qu, 40);

    display(qu);

    enqueuefront(&qu, 50);

    display(qu);

    dequeuelast(&qu);

    display(qu);

    dequeuelast(&qu);

    display(qu);

}

Output-

