LAB ASSIGNMENT 4

U24CS076

RUSHANG BAGADA

Q1:

Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression.

*#include* <stdio.h>

*#include* <stdlib.h>

*#include* <ctype.h>

*#include* <string.h>

*#define* *MAX* 100

*typedef* *struct*

{

*int* *top*;

*char* *items*[*MAX*];

} Stack;

*void* *initStack*(Stack *\**s)

{

    s->*top* *=* *-*1;

}

*int* *isFull*(Stack *\**s)

{

*return* s->*top* *==* *MAX* *-* 1;

}

*int* *isEmpty*(Stack *\**s)

{

*return* s->*top* *==* *-*1;

}

*void* *push*(Stack *\**s, *char* value)

{

*if* (*!isFull*(s))

    {

        s->*items*[*++*s->*top*] *=* value;

    }

}

*char* *pop*(Stack *\**s)

{

*if* (*!isEmpty*(s))

    {

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

    }

*return* '\0';

}

*char* *peek*(Stack *\**s)

{

*if* (*!isEmpty*(s))

    {

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

    }

*return* '\0';

}

*int* *precedence*(*char* operator)

{

*switch* (operator)

    {

*case* '+':

*case* '-':

*return* 1;

*case* '\*':

*case* '/':

*return* 2;

*case* '^':

*return* 3;

*default*:

*return* 0;

    }

}

*int* *isOperand*(*char* ch)

{

*return* *isalnum*(ch);

}

*// Converts infix to postfix*

*void* *infixToPostfix*(*const* *char* *\**infix, *char* *\**postfix)

{

    Stack *s*;

*initStack*(*&s*);

*int* *j* *=* 0;

*for* (*int* *i* *=* 0; infix[*i*] *!=* '\0'; *i++*)

    {

*if* (*isOperand*(infix[*i*]))

        {

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

        }

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

        {

*push*(*&s*, infix[*i*]);

        }

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

        {

*while* (*!isEmpty*(*&s*) *&&* *peek*(*&s*) *!=* '(')

            {

                postfix[*j++*] *=* *pop*(*&s*);

            }

*pop*(*&s*);

        }

*else*

        { *// Operator*

*while* (*!isEmpty*(*&s*) *&&* *precedence*(*peek*(*&s*)) *>=* *precedence*(infix[*i*]))

            {

                postfix[*j++*] *=* *pop*(*&s*);

            }

*push*(*&s*, infix[*i*]);

        }

    }

*while* (*!isEmpty*(*&s*))

    {

        postfix[*j++*] *=* *pop*(*&s*);

    }

    postfix[*j*] *=* '\0';

}

*int* *main*()

{

*char* *infix*[*MAX*], *postfix*[*MAX*];

*printf*("Enter an infix expression: ");

*scanf*("%s", *infix*);

*infixToPostfix*(*infix*, *postfix*);

*printf*("Postfix Expression: %s\n", *postfix*);

*return* 0;

}

OUTPUT:



Q2:

Write a program to implement a Queue data structure using two Stacks.

*#include* <stdio.h>

*#include* <stdlib.h>

*#define* *MAX* 100

*// Stack definition*

*typedef* *struct*

{

*int* *top*;

*int* *items*[*MAX*];

} Stack;

*// Queue definition*

*typedef* *struct*

{

    Stack *stack1*; *// Used for enqueue*

    Stack *stack2*; *// Used for dequeue*

} Queue;

*// Function prototypes*

*void* *initStack*(Stack *\**s);

*int* *isFull*(Stack *\**s);

*int* *isEmpty*(Stack *\**s);

*void* *push*(Stack *\**s, *int* value);

*int* *pop*(Stack *\**s);

*void* *enqueue*(Queue *\**q, *int* value);

*int* *dequeue*(Queue *\**q);

*void* *display*(Queue *\**q);

*int* *main*()

{

    Queue *q*;

*initStack*(*&q*.*stack1*);

*initStack*(*&q*.*stack2*);

*enqueue*(*&q*, 10);

*enqueue*(*&q*, 20);

*enqueue*(*&q*, 30);

*printf*("Dequeued: %d\n", *dequeue*(*&q*)); *// Should print 10*

*printf*("Dequeued: %d\n", *dequeue*(*&q*)); *// Should print 20*

*enqueue*(*&q*, 40);

*printf*("Dequeued: %d\n", *dequeue*(*&q*)); *// Should print 30*

*printf*("Dequeued: %d\n", *dequeue*(*&q*)); *// Should print 40*

*return* 0;

}

*// Initializes the stack*

*void* *initStack*(Stack *\**s)

{

    s->*top* *=* *-*1;

}

*// Checks if the stack is full*

*int* *isFull*(Stack *\**s)

{

*return* s->*top* *==* *MAX* *-* 1;

}

*// Checks if the stack is empty*

*int* *isEmpty*(Stack *\**s)

{

*return* s->*top* *==* *-*1;

}

*// Pushes an item onto the stack*

*void* *push*(Stack *\**s, *int* value)

{

*if* (*!isFull*(s))

    {

        s->*items*[*++*s->*top*] *=* value; *// Place value on stack*

    }

}

*// Pops an item from the stack*

*int* *pop*(Stack *\**s)

{

*if* (*!isEmpty*(s))

    {

*return* s->*items*[s->*top--*]; *// Return top item and remove*

    }

*return* *-*1; *// Return -1 if stack is empty (error case)*

}

*// Enqueues an item into the queue*

*void* *enqueue*(Queue *\**q, *int* value)

{

*push*(*&*q->*stack1*, value); *// Push onto stack1*

}

*// Dequeues an item from the queue*

*int* *dequeue*(Queue *\**q)

{

*if* (*isEmpty*(*&*q->*stack2*))

    { *// If stack2 is empty, move elements*

*while* (*!isEmpty*(*&*q->*stack1*))

        {

*push*(*&*q->*stack2*, *pop*(*&*q->*stack1*)); *// Move elements to stack2*

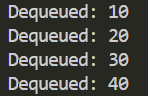
        }

    }

*return* *pop*(*&*q->*stack2*); *// Pop from stack2*

}

OUTPUT:



Q3:

Design, Develop and Implement a Program in C for Tower of Hanoi.

*#include* <stdio.h>

*#include* <math.h>

*void* *towerOfHanoiIterative*(*int* n, *char* source, *char* target, *char* auxiliary)

{

*int* *totalMoves* *=* (1 *<<* n) *-* 1;

*if* (n *%* 2 *==* 0)

    {

*char* *temp* *=* target;

        target *=* auxiliary;

        auxiliary *=* *temp*;

    }

*for* (*int* *i* *=* 1; *i* *<=* *totalMoves*; *i++*)

    {

*if* (*i* *%* 3 *==* 1)

        {

*printf*("Move disk from %c to %c\n", source, target);

        }

*else* *if* (*i* *%* 3 *==* 2)

        {

*printf*("Move disk from %c to %c\n", source, auxiliary);

        }

*else*

        {

*printf*("Move disk from %c to %c\n", auxiliary, target);

        }

    }

}

*int* *main*()

{

*int* *n*;

*printf*("Enter the number of disks: ");

*scanf*("%d", *&n*);

*printf*("The sequence of moves involved in the Tower of Hanoi are:\n");

*towerOfHanoiIterative*(*n*, 'A', 'C', 'B');

*return* 0;

}

OUTPUT:

Enter the number of disks: 5

The sequence of moves involved in the Tower of Hanoi are:

Move disk from A to C

Move disk from A to B

Move disk from B to C

Move disk from A to C

Move disk from A to B

Move disk from B to C

Move disk from A to C

Move disk from A to B

Move disk from B to C

Move disk from A to C

Move disk from A to B

Move disk from B to C

Move disk from A to C

Move disk from A to B

Move disk from B to C

Move disk from A to C

Move disk from A to B

Move disk from B to C

Move disk from A to C

Move disk from A to B

Move disk from B to C

Move disk from A to C

Move disk from A to B

Move disk from B to C

Move disk from A to C

Move disk from A to B

Move disk from B to C

Move disk from A to C

Move disk from A to B

Move disk from B to C

Move disk from A to C

Q4:

Consider a scenario of a bank. When the customer visits the counter, a request entry is made and the customer is given a request number. After receiving request numbers, a customer has to wait for some time. The customer requests need to be queued into the system and processed on the basis of their arrival. You need to implement an appropriate data storage mechanism to store these requests in the system and display after completing the customer task on the request.

*#include* <stdio.h>

*#include* <stdlib.h>

*typedef* *struct* Request

{

*int* *requestNumber*;

*struct* Request *\*next*;

} Request;

*typedef* *struct* Queue

{

    Request *\*front*;

    Request *\*rear*;

} Queue;

Queue *\*createQueue*();

*void* *enqueue*(Queue *\**q, *int* requestNumber);

*void* *enqueueMultiple*(Queue *\**q);

*int* *dequeue*(Queue *\**q);

*void* *displayQueue*(Queue *\**q);

*void* *processRequest*(Queue *\**q);

*int* *main*()

{

    Queue *\*q* *=* *createQueue*();

*int* *choice*;

*while* (1)

    {

*printf*("\nBank Counter Management System\n");

*printf*("1. Add Customer Request\n");

*printf*("2. Process Customer Request\n");

*printf*("3. Display All Requests\n");

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

*printf*("Enter your choice: ");

*scanf*("%d", *&choice*);

*switch* (*choice*)

        {

*case* 1:

*enqueueMultiple*(*q*);

*break*;

*case* 2:

*processRequest*(*q*);

*break*;

*case* 3:

*displayQueue*(*q*);

*break*;

*case* 4:

*printf*("Exiting the system.\n");

*free*(*q*);

*exit*(0);

*default*:

*printf*("Invalid choice! Please try again.\n");

        }

    }

*return* 0;

}

Queue *\*createQueue*()

{

    Queue *\*q* *=* (Queue *\**)*malloc*(*sizeof*(Queue));

*q*->*front* *=* *NULL*;

*q*->*rear* *=* *NULL*;

*return* *q*;

}

*void* *enqueue*(Queue *\**q, *int* requestNumber)

{

    Request *\*newRequest* *=* (Request *\**)*malloc*(*sizeof*(Request));

*newRequest*->*requestNumber* *=* requestNumber;

*newRequest*->*next* *=* *NULL*;

*if* (q->*rear* *==* *NULL*)

    {

        q->*front* *=* *newRequest*;

        q->*rear* *=* *newRequest*;

    }

*else*

    {

        q->*rear*->*next* *=* *newRequest*;

        q->*rear* *=* *newRequest*;

    }

}

*void* *enqueueMultiple*(Queue *\**q)

{

*int* *requestNumber*;

*printf*("Enter request numbers (separated by spaces, end with -1): ");

*while* (1)

    {

*scanf*("%d", *&requestNumber*);

*if* (*requestNumber* *==* *-*1)

        {

*break*;

        }

*enqueue*(q, *requestNumber*);

*printf*("Request %d added to the queue.\n", *requestNumber*);

    }

}

*void* *processRequest*(Queue *\**q)

{

*if* (q->*front* *==* *NULL*)

    {

*printf*("No requests to process.\n");

*return*;

    }

*int* *processedRequest* *=* *dequeue*(q);

*printf*("Processed request number: %d\n", *processedRequest*);

}

*int* *dequeue*(Queue *\**q)

{

*if* (q->*front* *==* *NULL*)

    {

*return* *-*1;

    }

    Request *\*temp* *=* q->*front*;

*int* *requestNumber* *=* *temp*->*requestNumber*;

    q->*front* *=* q->*front*->*next*;

*if* (q->*front* *==* *NULL*)

    {

        q->*rear* *=* *NULL*;

    }

*free*(*temp*);

*return* *requestNumber*;

}

*void* *displayQueue*(Queue *\**q)

{

*if* (q->*front* *==* *NULL*)

    {

*printf*("No requests in the queue.\n");

*return*;

    }

    Request *\*current* *=* q->*front*;

*printf*("Current requests in the queue: ");

*while* (*current* *!=* *NULL*)

    {

*printf*("%d ", *current*->*requestNumber*);

*current* *=* *current*->*next*;

    }

*printf*("\n");

}

OUTPUT:

Bank Counter Management System

1. Add Customer Request

2. Process Customer Request

3. Display All Requests

4. Exit

Enter your choice: 1

Enter request numbers (separated by spaces, end with -1): 1 2 3 4 5 -1

Request 1 added to the queue.

Request 2 added to the queue.

Request 3 added to the queue.

Request 4 added to the queue.

Request 5 added to the queue.

Bank Counter Management System

1. Add Customer Request

2. Process Customer Request

3. Display All Requests

4. Exit

Enter your choice: 2

Processed request number: 1

Bank Counter Management System

1. Add Customer Request

2. Process Customer Request

3. Display All Requests

4. Exit

Enter your choice: 2

Processed request number: 2

Bank Counter Management System

1. Add Customer Request

2. Process Customer Request

3. Display All Requests

4. Exit

Enter your choice: 3

Current requests in the queue: 3 4 5

Bank Counter Management System

1. Add Customer Request

2. Process Customer Request

3. Display All Requests

4. Exit

Enter your choice: 4

Exiting the system.