**Q1: Write a C program to convert an infix expression to a postfix expression.**

**-> A:) Algorithm:**

**I. Create a stack using structure and make an character array to store the answer ie.,postfix**

**expression.**

**II. Implement basic functions like pop(), push(), isEmpty() to perform basic operations on stack.**

**III.Traverse the given string from left to right and follow the below steps:**

**1. If operand is found -> store it in the answer array.**

**2.else if '(' -> push it to the stack.**

**3. else if ')' -> pop from the stack and add the popped elements to the answer array until '(' is found.**

**4.else if operator -> if stack top precedence is greater than the operator,**

**pop elements from the stack and add it to the answer arrayuntil an operator with less precedence is found.**

**5.else -> print invalid infix expression and exit.**

**IV. Return the answer array and print its values in the main function.**

**`````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````**

**BELOW IS THE IMPLEMENTATION OF THE RULE( DRY RUN ):**

**Implementation of infix expression : (a – b / c) \* (a / k – l)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No.** | **Current symbol** | **stack** | **Postfix Exp.** | **Reason(see above pts.)** |
| **1.** | **(** | **(** |  | **2.** |
| **2.** | **a** | **(** | **a** | **1.** |
| **3.** | **-** | **(-** | **a** | **4.** |
| **4.** | **b** | **(-** | **ab** | **1.** |
| **5.** | **/** | **(-/** | **ab** | **4.** |
| **6.** | **c** | **(-/** | **abc** | **1.** |
| **7.** | **)** |  | **abc/-** | **3.** |
| **8.** | **\*** | **\*** | **abc/-** | **4.** |
| **9.** | **(** | **\*(** | **abc/-** | **2.** |
| **10.** | **a** | **\*(** | **abc/-a** | **1.** |
| **11.** | **/** | **\*(/** | **abc/-a** | **4.** |
| **12.** | **k** | **\*(/** | **abc/-ak** | **1.** |
| **13.** | **-** | **\*(-** | **abc/-ak/** | **4.** |
| **14.** | **l** | **\*(-** | **abc/-ak/l** | **1.** |
| **15.** | **)** |  | **abc/-ak/l-\*** | **3.** |

// Archana\_Kumari.ECE.408.Mid\_Term.CS\_DSA\_Theory.Assignment

// Q1: Write a C program to convert infix expression to postfix.

// Below is it's implementation using stack.

#include <stdio.h>

#include <ctype.h>

#define LIMIT 1000

char ansArr[LIMIT];  //global declaration.

struct myStack{

  char infixArr[LIMIT], top;

}st1;

//Fxn to push one element to the stack.

void push(char ele) {

  if (st1.top >= LIMIT -1) {

     printf("Stack underflow");

     return;

  }

  st1.infixArr[++st1.top] = ele;

}

//Fxn to pop the top element and return it's value.

char pop() {

 if(st1.top <= -1) {

  printf("Stack overflow");

  return 0;

 }

 return st1.infixArr[st1.top--];

}

//Fxn to check if the stack is empty or not.

int isEmpty() {

  return st1.top == -1;

}

//Fxn to check the mathematical precedence of any valid operator.

int operatorPrec(char ch){

    if (ch == '^')  return 3;

    else if(ch == '\*' || ch=='/')  return 2;

    else if(ch == '+' || ch=='-')  return 1;

    else  return -1;

}

//Fxn to convert an infix expression to a postfix expression.

char\* infixToPostfixExp(char exp[]) {

  int i = 0, j = 0;

  char z = 0;

  for(; exp[i] != '\0'; ++i) {

    if (isdigit(exp[i]) || isalpha(exp[i]))

      ansArr[j++] = exp[i];

    else if(exp[i] == '(')

      push(exp[i]);

    else if(exp[i] == ')') {

       while( !isEmpty() && st1.infixArr[st1.top] != '('){

           ansArr[j++] = pop();

        }

        if(!isEmpty())

          z = pop();

    }

    else if (operatorPrec(exp[i]) > 0) {

        while(operatorPrec(st1.infixArr[st1.top]) > 0   && operatorPrec(st1.infixArr[st1.top]) >= operatorPrec(exp[i]) ) {

           ansArr[j++] = pop();

        }

        push(exp[i]);

    }

    else {

      printf("Invalid INFIX expression.");

      break;

    }

  }

  while(!isEmpty() ) {

    if(st1.infixArr[st1.top] != '(')

      ansArr[j++] = pop();

      else

      z = pop();

  }

  ansArr[j] = '\0';

  return ansArr;

}

int main() {

    st1.top = -1;

   // printf("Enter the infix expression: ");

    char infixExp[100];

    gets(infixExp);

    printf("%s", infixToPostfixExp(infixExp));

    return 0;

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

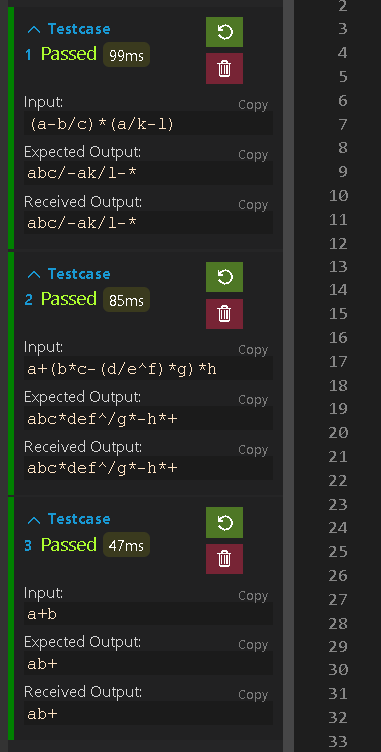
**C:) Input-Output:**

**Input\_1 : (a-b/c)\*(a/k-l) Input\_2 : a+b**

**Output\_1 : abc/-ak/l-\* Output\_2 : ab+**

**Input\_3 : (a+b)\*(c+d) Input-4 : a+(b\*c-(d/e^f)\*g)\*h**

**Output\_3 : ab+cd+\* Output\_4 : abc\*def^/g\*-h\*+**



**Q2:) Write a C program to evaluate the value of a given postfix expression.**

**A:) Algorithm:**

**I. Create a stack to store operands value.**

**II. Make 2 variables(say, op1, op2) of int datatype to store the evaluate the expression.**

**III. Traverse the given string from left to right and follow the below steps:**

**1. if the character is an operand, push it to the stack.**

**2.else if the character is an operator,**

**i) pop the elements twice from the stack.**

**ii) Store the first popped value to op2 and the next to op1.**

**iii) perform the operation on them and evaluate the result.**

**iv) push the obtained result to the stack.**

**IV. Perform the step III. until we traverse the entire string.**

**V. The element left at the end in the stack is the answer.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**DRY RUN:**

**Implementation of postfix expression : 4 6 + 2 / 5 \* 7 +**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Op\_1** | **Op\_2** | **Result** |
| **1.** | **4** | **6** | **4+6 = 10** |
| **2.** | **10** | **2** | **10/2 = 5** |
| **3.** | **5** | **5** | **5\*5 = 25** |
| **4.** | **25** | **7** | **25+7=32** |

// Archana\_Kumari.ECE.408.Mid\_Term.CS\_DSA\_Theory.Assignment

// Q2: Write a C program evaluate postfix expression.

// Below is it's implementation using stack.

#include <stdio.h>

#include <ctype.h>

#include <math.h>

#define LIMIT 1000

struct myStack{

  char postfixArr[LIMIT], top;

}st1;

//Fxn to push one element to the stack.

void push(int op) {

  if (st1.top >= LIMIT -1) {

     printf("Stack underflow");

     return;

  }

  st1.postfixArr[++st1.top] = op;

}

//Fxn to pop the top element and return it's value.

char pop() {

 if(st1.top <= -1) {

  printf("Stack overflow");

  return 0;

 }

 return st1.postfixArr[st1.top--];

}

int postfixValue(char exp[]) {

  int i = 0;

  for(; exp[i] != '\0'; ++i ) {

    if (isdigit(exp[i])) {

      push(exp[i] - '0');

    }

    else {

      int oper2 = pop();

      int oper1 = pop();

      switch(exp[i]) {

        case '+':

           push(oper1 + oper2);

           break;

        case '-':

           push(oper1 - oper2);

           break;

        case '\*':

           push(oper1 \* oper2);

           break;

        case '/':

           push(oper1 / oper2);

           break;

        case '^':

           push(pow(oper1, oper2));

           break;

        default :

           printf("Invalid operator.");

           break;

      }

    }

  }

  return st1.postfixArr[st1.top];

}

int main() {

    st1.top = -1;

   // printf("Enter the postfix expression: ");

    char postfixExp[100];

    gets(postfixExp);

    printf("%d", postfixValue(postfixExp));

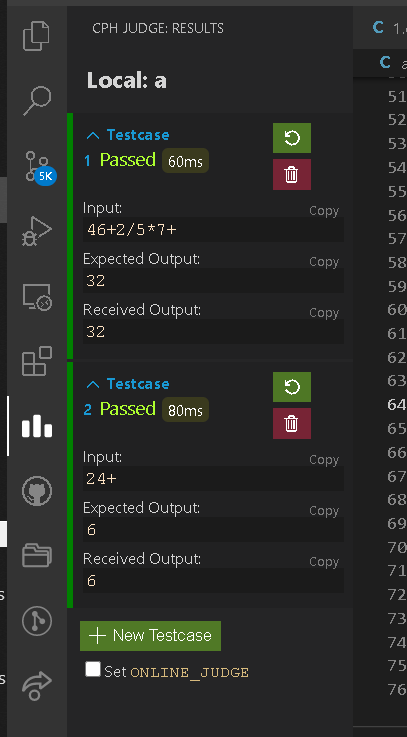
    return 0;

}

**C:) Input-Output:**

**Input\_1 : 24+ Input-2 : 46+2/5\*7+**

**Output\_1 : 6 Output\_2 : 32**



**Q3:) Give a real-life example using queue.**

**A:) Problem statement: Patient Queue (using Priority Queue) :**

* **This problem deals with the real-life implementation of Priority Queue in hospitals for managing the waiting list of patients.**
* **To assist patients in a hospitals,because some pateints may have more urgent and serious injuries than others, use priority queue to manage the waiting list of patients in a hospital.**
* **The patient with more urgent priority is seen first, regardless of the appointment no.**
* **The patient with highest priority is removed first as well.**

**B:) Algorithm: Make a structure with integer element & integer priority as structural variables.**

**I) Enqueue (Insert) -> 1. Take the data and it's priority as input.**

**2. If front == 0 & rear == size -1, then queue is full.**

**3. Else we initialise front and rear with 0.**

**4. Insert the data in Priority Queue using rear pointer.**

**II) Delete highest priority (dequeue) -> 1. Removes the element with the highest priority from the queue.**

**2. Searches the element with highest priority and stores it in a variable.**

**3. Shifts the elements to delete it and decrements the rear pointer.**

**4. Returns the deleted element.**

**III) Display -> Loop through the priority queue from front to the rear pointer and print it's data and priority. Returns the list of the patients a/c their priority.**

**IV) highestPr -> Returns the highest priority input by the user.**

**V) isFull -> Return true if the the Priority Queue is full.**

**VI) isEmpty -> Returns true if the Priority Queue is empty ie., front == -1, else false.**

// Archana\_Kumari.ECE.408.Mid\_Term.CS\_DSA\_Theory.Assignment

// Q3: Write a C program to show a real-life example using priority queue.

// Below is it's implementation.

#include <stdio.h>

#include <conio.h>

#define LIMIT 100

struct priorityQueue {

  int ele;

  int pr;

}pq[LIMIT];

int rear = -1, front = -1;

int isEmpty() {

   if(rear == -1)

     return 1;

   return 0;

}

int isFull() {

  if(rear == LIMIT -1 && front == 0)

    return 1;

  return 0;

}

void enqueue(int ele, int p) {

   if(isFull()) {

      printf("Priority Queue is full");

      return;

   }

   else{

     if(rear == -1) {

       ++front;

       ++rear;

       pq[rear].ele = ele;

       pq[rear].pr = p;

   }

   else {

      rear++;

      pq[rear].ele = ele;

      pq[rear].pr = p;

    }

  }

}

int highestPr() {

  if(isEmpty()) {

       printf("\nPriority Queue is Empty.");

       return -1;

   }

    int i = 0, p = -1;

    if(!isEmpty()) {

        for(; i<= rear; ++i) {

           if(pq[i].pr > p) {

             p = pq[i].pr;

            }

        }

     }

    return p;

}

int dequeue() {

     if(isEmpty()) {

        printf("\n Priority Queue is Empty.");

        return -1;

     }

   int i, j, p, ele;

   p = highestPr();

   for(i = 0; i <= rear; ++i) {

       if(pq[i].pr == p) {

       ele = pq[i].ele;

       break;

   }

  }

  if(i < rear) {

    for(j=i; j< rear; j++){

      pq[j].ele = pq[j++].ele;

      pq[j].pr = pq[j++].pr;

      }

   }

   rear--;

   return ele;

}

void display() {

    if(isEmpty()) {

        printf("\n Priority Queue is Empty.");

        return;

    }

   int i = front;

   printf("Priority Queue is: ");

   for( ; i<= rear; ++i) {

      printf("\nPatient's appointment no. = %d , Priority = %d", pq[i].ele, pq[i].pr);

   }

}

int main() {

   int c = 0, p = 0, ele = 0;

   do{

   printf("\n 1.Insert, 2.Peek value, 3.Delete Peek Value, 4.Display, 5.Exit");

   printf("\nEnter choice: ");

   scanf("%d", &c);

   switch(c) {

     case 1:

           printf("Enter Patient's appointment no.: ");

           scanf("%d", &ele);

           printf("Enter the priority: ");

           scanf("%d", &p);

           enqueue(ele, p);

        break;

    case 2:

           p = highestPr();

           printf("\nHighest Priority is: %d", p);

        break;

    case 3:

           ele = dequeue();

           printf("\nPateint with appointment no. %d is deleted.", ele);

       break;

    case 4:

             display();

        break;

        case 5: break;

    default: printf("\nWrong input");

           }

   }while(c != 5);

   return 0;

}

`````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````

**D:) Input-Output:**

**1.Insert, 2.Peek value, 3.Delete Peek Value, 4.Display, 5.Exit**

**Enter choice: 3**

**Priority Queue is Empty.**

**Pateint with appointment no. -1 is deleted.**

**1.Insert, 2.Peek value, 3.Delete Peek Value, 4.Display, 5.Exit**

**Enter choice: 2**

**Priority Queue is Empty.**

**Highest Priority is: -1**

**1.Insert, 2.Peek value, 3.Delete Peek Value, 4.Display, 5.Exit**

**Enter choice: 1**

**Enter Patient's appointment no.: 3**

**Enter the priority: 2**

**1.Insert, 2.Peek value, 3.Delete Peek Value, 4.Display, 5.Exit**

**Enter choice: 1**

**Enter Patient's appointment no.: 2**

**Enter the priority: 1**

**1.Insert, 2.Peek value, 3.Delete Peek Value, 4.Display, 5.Exit**

**Enter choice: 1**

**Enter Patient's appointment no.: 1**

**Enter the priority: 3**

**1.Insert, 2.Peek value, 3.Delete Peek Value, 4.Display, 5.Exit**

**Enter choice: 4**

**Priority Queue is:**

**Patient's appointment no. = 3 , Priority = 2**

**Patient's appointment no. = 2 , Priority = 1**

**Patient's appointment no. = 1 , Priority = 3**

**1.Insert, 2.Peek value, 3.Delete Peek Value, 4.Display, 5.Exit**

**Enter choice: 2**

**Highest Priority is: 3**

**1.Insert, 2.Peek value, 3.Delete Peek Value, 4.Display, 5.Exit**

**Enter choice: 3**

**Pateint with appointment no. 1 is deleted.**

**1.Insert, 2.Peek value, 3.Delete Peek Value, 4.Display, 5.Exit**

**Enter choice: 4**

**Priority Queue is:**

**Patient's appointment no. = 3 , Priority = 2**

**Patient's appointment no. = 2 , Priority = 1**

**1.Insert, 2.Peek value, 3.Delete Peek Value, 4.Display, 5.Exit**

**Enter choice: 5**

**--------------------------------**

**Process exited after 32.12 seconds with return value 0**

**Press any key to continue . . .**

