Universidad Naciona Autónoma de México

Facultad de Ingeniería

Estructuras de Datos y Algoritmos I

Actividad: Implementación de algoritmo para realizar operaciones contenidas en una cadena de caracteres ingresada por el usuario

Sánchez Hernández Marco Antonio

Fecha: 23/junio/2021

Código fuente

La implementación del algoritmo para realizar las operaciones contenidas en una cadena de caracteres dada por el usuario, presentado en la actividad anterior, fue implementado en el lenguaje de programación *C*. El código fuente del programa fue dividio en dos archivos (“*main.c*” y “*stack.h*”) presentados a continuación:

Código contenido en el archivo *main.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "stack.h"

//Prototipo de la función higherpreced
int higherpreced(char, char);

//Prototipo de la función chtoi
int chtoi(char);

//Función main
int main() {

    char expresion[100];
    int tamanio, i, j, x, y;

    //La definición de los tipos de dato cstack y istack se encuentran en el archivo stack.h.
    cstack temp;
    istack compute;

    temp = init_cstack(); //Inicializa la pila de caracteres temp.
    compute = init_istack(); //Inicializa la pila de números enteros compute.

    printf("Ingrese una expresi\u00F3n aritm\u00E9tica: ");
    scanf("%s", expresion);

     /*Determina la longitud de la cadena de caracteres ingresada y almacena su valor en la
       variable tamanio.*/
    tamanio = strlen(expresion);
```

```
  /*Se crea un arreglo de caracteres del mismo tamaño que la longitud de la cadena
   ingresada.*/
char postfix[tamanio];

//Se inicializan las variables de iteración.
i = 0;
j = 0;

//Este ciclo reorganiza la expresión ingresada en la notación postfix.
while (i < tamanio) {

   //Esta expresión es verdadera si el elemento a considerar es un número.
   if ((expresion[i] > 47) && (expresion[i] < 58)) {
       postfix[j] = expresion[i];
       j++;
       i++;
   }

   else {

       //Si la pila temp está vacía, automáticamente se almacena el operador.
       if (empty_cstack(temp)) {
          temp = pushc(temp, expresion[i]);
          i++;
       }

       else {

           /*Si el operador ingresado no tiene mayor jerarquía que el operador contenido
   en el tope de la pila temp, se toma el elemento tope de la pila temp y se coloca el
   operador comparado en la pila temp*/
          if (!(higherpreced(ctop_element(temp), expresion[i]))) {
              postfix[j] = ctop_element(temp);
              temp = popc(temp);
              temp = pushc(temp, expresion[i]);
              j++;
              i++;
          }


          else {
```

```
                    /*Si el operador ingresadao es de mayor jerarquía que el operador
   contenido en el tope de la pila temp, se coloca el siguiente elemento de la  cadena y
   posteriormente los operadores, primero se coloca el de mayor jerarquía.*/
                  while  (!(empty_cstack(temp))  &&  higherpreced(ctop_element(temp),
   expresion[i])) {
               postfix[j] = expresion[i+1];
               j++;
               postfix[j] = expresion[i];
               j++;
               postfix[j] = ctop_element(temp);
               temp = popc(temp);
               j++;
               i++;
            }

         }
      }
   }
}

//Se colocan los elemento que sobren en la pila en la nueva cadena.
while (!(empty_cstack(temp))) {
   postfix[j] = ctop_element(temp);
   temp = popc(temp);
   j++;
}

//Este ciclo sirve para realizar cada una de las operaciones necesarias.
for (i = 0; i < tamanio; i++) {

   /*Si los elementos de la cadena postfix son números, son convertidos a enteros y
   almacenados en la pila compute.*/
   if ((postfix[i] > 47) && (postfix[i] < 58)) {
       x = chtoi(postfix[i]);
       compute = pushi(compute, x);
   }

     /*Si un elemento de la cadena postfix  es un operador, se ejecutará la operación
   binaria correspondiente.*/
   else {
```

```
//Para la suma
if (postfix[i] == '+') {
   x = itop_element(compute);
   compute = popi(compute);
   y = itop_element(compute);
   compute = popi(compute);
   x = x + y;
   compute = pushi(compute, x);
}


else {

   //Para la resta
   if (postfix[i] == '-') {
       x = itop_element(compute);
       compute = popi(compute);
       y = itop_element(compute);
       compute = popi(compute);
       x = y - x; //El órden se invierte.
       compute = pushi(compute, x);
   }

   else {

       //Para la multiplicación
       if (postfix[i] == '*') {
          x = itop_element(compute);
          compute = popi(compute);
          y = itop_element(compute);
          compute = popi(compute);
          x = x * y;
          compute = pushi(compute, x);
       }

       else {

          //Para la división
          if (postfix[i] == '/') {
              x = itop_element(compute);
              compute = popi(compute);
```

```
                        y = itop_element(compute);
                        compute = popi(compute);
                        x = y / x; //El orden se invierte
                        compute = pushi(compute, x);
                    }
                }
            }
        }
    }

    /*Cuando se realicen todas las operaciones se muestra el resultado contenido en la pila
       compute.*/
    printf("\nResultado: %d\n", itop_element(compute));
    compute = popi(compute);

    return 0;
}


//Función que determina la jerarquía de operaciones.
int higherpreced(char op1, char op2) {

    if ((op1 == '+' || op1 == '-') && (op2 == '*' || op2 == '/'))
       return TRUE;

    return FALSE;
}


//Función que convierte un carácter numérico en un entero.
int chtoi(char operand) {

    int sum = 0;
    sum = sum*10+(operand-48);

    return sum;
}
```

Código contenido en el archivo *stack.h*

```
#include <stdio.h>
```

```
#include <stdlib.h>

//Definición de constantes
#define TAMANIO 5
#define FALSE 0
#define TRUE 1

//Definición del tipo de dato abstraco cstak, el cual representa una pila de caracteres.
typedef struct {
    char stcklen[TAMANIO];
    int top;
} cstack;


//Definición del tipo de dato abstraco istak, el cual representa una pila de números enteros.
typedef struct {
    int stcklen[TAMANIO];
    int top;
} istack;


/*Prototipos de las funciones para el tipo de dato cstack*/
cstack init_cstack();

int empty_cstack(cstack);

cstack pushc(cstack, char);

cstack popc(cstack);

char ctop_element(cstack);


/*Prototipos de las funciones para el tipo de dato istack*/
istack init_istack();

int empty_istack(istack);

istack pushi(istack, int);

istack popi(istack);
```

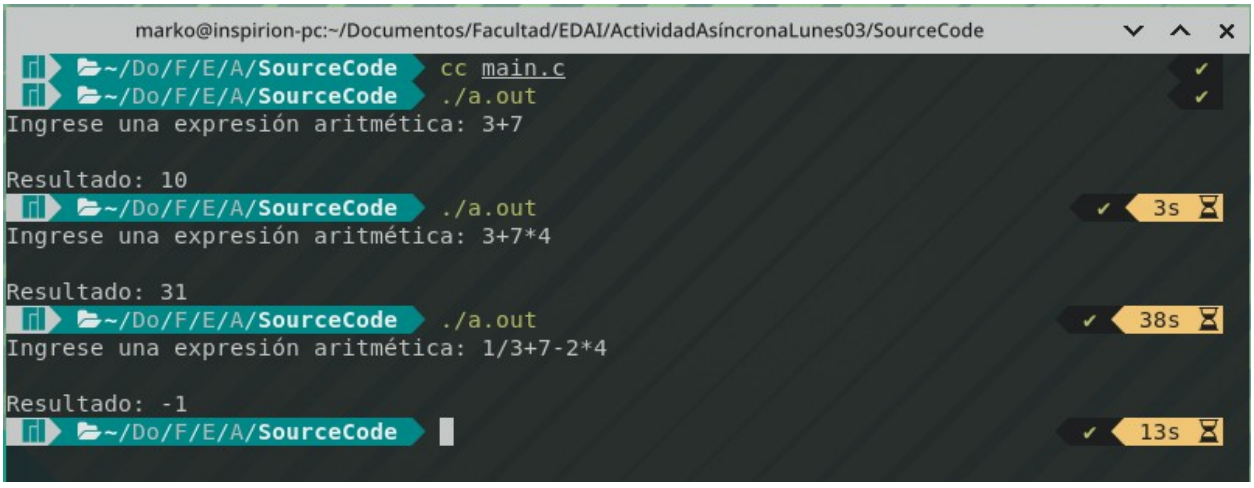
```
int itop_element(istack);


//Función que permite inicializar una pila de caracteres.
cstack init_cstack() {

    cstack temp_stack;
    temp_stack.top = -1;

    return temp_stack;
}


//Función que verifica si una pila de caracteres está vacía.
int empty_cstack(cstack input) {

    cstack temp_stack = input;

    if (temp_stack.top == -1)
       return TRUE;

    else
       return FALSE;
}


//Función que inserta un elemento a una pila de caracteres.
cstack pushc(cstack input, char element) {

    cstack temp_stack = input;

    if (temp_stack.top == TAMANIO -1) {
       printf("Stack Overflow\n");
       exit(1);
    }

    temp_stack.top++;
    temp_stack.stcklen[temp_stack.top] = element;

    return temp_stack;
```

```
}


//Función que elimina un elemento de una pila de caracteres.
cstack popc(cstack input) {

    cstack temp_stack = input;

    if (empty_cstack(temp_stack)) {
       printf("Stack Underflow\n");
       exit(1);
    }

    temp_stack.top--;

    return temp_stack;
}


//Función que regresa el valor del elemento tope de una pila de caracteres.
char ctop_element(cstack input) {

    cstack temp_stack = input;

    if (empty_cstack(temp_stack)) {
       printf("Stack Underflow\n");
       exit(1);
    }

    return temp_stack.stcklen[temp_stack.top];
}


//Función que permite inicializar una pila de números enteros.
istack init_istack() {

    istack temp_stack;
    temp_stack.top = -1;

    return temp_stack;
}
```

```
//Función que verifica si una pila de números enteros está vacía.
int empty_istack(istack input) {

    istack temp_stack = input;

    if (temp_stack.top == -1)
       return TRUE;

    else
       return FALSE;
}


//Función que inserta un elemento a una pila de números enteros.
istack pushi(istack input, int element) {

    istack temp_stack = input;

    if (temp_stack.top == TAMANIO -1) {
       printf("Stack Overflow\n");
       exit(1);
    }

    temp_stack.top++;
    temp_stack.stcklen[temp_stack.top] = element;

    return temp_stack;
}

//Función que elimina un elemento de una pila de números enteros.
istack popi(istack input) {

    istack temp_stack = input;

    if (empty_istack(temp_stack)) {
       printf("Stack Underflow\n");
       exit(1);
    }
```

```
    temp_stack.top--;

    return temp_stack;
}


//Función que regresa el valor del elemento tope de una pila de números enteros.
int itop_element(istack input) {

    istack temp_stack = input;

    if (empty_istack(temp_stack)) {
       printf("Stack Underflow\n");
       exit(1);
    }

    return temp_stack.stcklen[temp_stack.top];
}
```

Ejecución del código



Para la primera ejecución (*3+7*) el resultado es correcto, ya que 3+7=10.

Para la segunda ejecución (*3+7*4*) el resultado es correcto, ya que 7*4=28 y 28+3=31.

En la tercera ejecución el resultado no es correcto, sin embargo, el programa respeta la jerarquía de operaciones. El problema radica en que la división ejecutada es entre enteros, lo que ocasiona que se trunque la parte decimal, dando el resultado originado.

1/3 = 0.666.., sin embargo, al ser una división entre datos enteros, el resultado es 0. Luego 0+7=7, 2*4=8 y 7-8=-1.