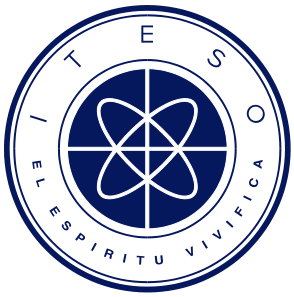
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

PROGRAMACIÓN CON MEMORIA DINÁMICA



EVALUACIÓN II – EVALUADOR DE EXPRESIONES MATEMÁTICAS.

Presentan

Enoch Joaquín Álvarez Goñi 739917

Jorge Ramón Figueroa Maya 739924

José Emiliano Figueroa Hernández 738336

Profesor: Francisco Cervantes

Fecha: <03/11/2022>

Contenido.

Introducción.

Las matemáticas siempre han sido y siempre serán las bases de nuestro mundo, incluso diría que de nuestro universo y mientras más pasa el tiempo más operaciones son inventadas, más números se descubren, más algoritmos de solución se crean, métodos a seguir y reglas que obedecer, por esto mismo en su tiempo se crearon las jerarquías de operaciones.

Las jerarquías de operaciones dicen cuál es el orden correcto en que se interpretan expresiones aritméticas que contienen varias operaciones. Esta nos dicta cuáles deben hacerse primero, de modo que el resultado sea el correcto. Este orden evita que haya más de una interpretación en las operaciones, ya que las matemáticas sólo existen una respuesta correcta.

Al mismo tiempo que las matemáticas avanzan la tecnología moderna es creada bajo las reglas de estas mismas y esta tecnología se aprovechó para resolver expresiones matemáticas. Inicio con sumas, restas, multiplicaciones y divisiones sin embargo se requería que tuviesen una jerarquía de operaciones. Esto fue resuelto, sin embargo, con el paso del tiempo se creaban nuevas tecnologías y nuevos métodos para resolver diferentes problemas.

Existen diversos algoritmos que nos permiten resolver expresiones matemáticas los cuales logran llegar al mismo resultado con diferentes métodos algunos son las notaciones infijas, prefijas y postfijas. Estas nos ayudan a realizar una exitosa evaluación con diferentes tecnologías.

En la actualidad existen estructuras de datos que nos ayudan a ordenar expresiones y a reducir tiempos de ejecución, costo de memoria y operaciones. Debido a esto se implementan nuevos métodos de evaluaciones de expresiones matemáticas con las nuevas tecnologías y algoritmos.

Análisis del problema.

Se sabe que al ingresar una expresión matemática esta lleva un orden en la que se resuelve para que exista una única solución correcta. Una computadora no tiene la remota idea de en qué orden debe evaluar la operación, esta lee de izquierda a derecha sin importar la jerarquía.

Esto nos presenta una problemática donde el programador debe de dar las instrucciones para que esta pueda seguir con un orden de ejecución de la entrada matemática, por ello tenemos los siguientes objetivos.

Objetivo General

Diseñar un algoritmo que evalúe de manera correcta la expresión matemática.

Objetivos Específicos

- Evaluar la expresión de manera correcta e imprimir al usuario.
- Presentar al usuario la expresión matemática en notación Posfija.
- Informar al usuario si a su expresión le hacen falta paréntesis.

Propuesta de solución.

```
Proceso stackPop_B(si top de stack es igual a –1 regresar –1
                     Si no regresar un espacio antes de top)
Proceso ordenJerarquico(si la variable x es igual al operador ‘)’ regresar 0
                       Si la variable x es igual a ‘+’ o a ‘-’ regresar 1
                       Si la variable x es igual a ‘*’ o a ‘/’ regresar 2)
Proceso stack_create(tamaño en bytes
                         stk = malloc(sizeof(struct stack));
                   Stk que apunta a bytes=bytes;
                   Stk que apunta a size=0;
                   Stk que apunta a top=NULL;
```

```
                    Regresar stk)
Proceso satck_push(stk en data
                        Si stk es diferente a NULL
                    Nd=proceso newNode(en data para stk que apunta a bytes)
                    Stk que apunta a size + 1
                    nd que apunta a prior=stk que apunta a top
                    Stk que apunta a top=nd
                        Sino
              Escribir “error stack is NULL”)
Proceso newNode( data y tamaño en bytes
                     new= malloc(tamaño de(struct NodeStack);
                     New que apunta a data=malloc(tamaño de bytes);
                     Memcpy(de new que apunta a data para data en tamaño de bytes)
                     New que apunta a prior =NULL;
                     Regresar new)

Proceso stack_pop(en stk
                       Si stk es diferente a NULL
               si stk que apunta a top es diferente a NULL
                                  *dataTemp=stk que apunta a top que apunta a data;
                                  new_top=stk que apunta a top que apunta a prior;
                                  liberar(stk que apunta a top);
                                  Stk que apunta a top=new_top;
                                   Stkque apunta a size-1;
                                   regesar dataTemp;
                      si no
                          regresar NULL;
                     Si no
                      Regresar NULL;)
Inicio del procedimiento
input[100;
*ptr, x, num;
Escribir “expresion a evaluar”
```

```
Leer input
Ptr igual a input
Pos=0;
Stk=proceso stack_create(tamaño de(double));
Mientras que *ptr sea diferente a ‘\0’
n, x10
si *ptr es un numero
escribir los numeros
entero=*ptr menos ‘0’;
numd=entero;
proceso stack_push(&numd en estk)
x10=double proceso stack_pop(en stk)
si no si *ptr es igual a ‘(’
proceso stack_pushB(en *ptr)
sino si *ptr es igual a ‘)’
mientras x que es igual al proceso stackpop_B sea diferente a ‘(‘
op;
escribir x
*pop1=NULL;
*pop2=NULL;
res=0;
inicio switch
caso ‘+’
escribir "--SUMA--"
                                                pop1=stack_pop(stk);
                                                pop2=stack_pop(stk);
                                                 escribir pop1
                                                escribir pop2
                                                 res = *pop1+*pop2;
                                                  escribir res
                                                  proceso   stack_push(de   &res   en
stk);
                                                 printf("%f\n", stack_top(stk));
```

```
                                                parar
caso ‘-’
escribir "--RESTA--"
                                                parar
caso ‘*’
escribir "--MULTI--"
                                                parar
casi ‘/’
escribir "--DIV--"
                                                parar
fin switch
escribir “operador : x”
si no
mientras proceso ordenJerarquico(en stack[top)sea menor a proceso
                                                ordenJerarquico(en *ptr)
n=proceso stackpop_B
escribir n
fin mientras
proceso stackPush_B(en *ptr)
si pos es menor a 1
pos +1
si no
 *pop1 = NULL;
 *pop2 = NULL;
  res = 0;
inicio switch en *ptr
case '+':
                                        escribir "--SUMA--"
                                                    pop1=proeso      stack_pop(en
stk);
                                                    pop2=proceso     stack_pop(en
stk);
                                                    escribir"valor de", *pop1
                                                    escribir "valor de", *pop2
```

```
                                        res = *pop1+*pop2;
                                        escribir "valor de", res
                                        proceso stack_push(de &res en
stk);
                                        escribir "valor de", proceso
stack_top(en stk));
                                        parar
                                    case '-':
                                        escribir "--RESTA--"
                                        parar
                                    case '*':
                                        escribir "--MULTI--"
                                        parar
                                    case '/':
                                        escribir "--DIV--"
                                        parar
                            Fin switch
escribir “valor de operador”, x
fin sino
ptr + 1
Fin mientras
Mientras top sea diferente a –1
n=proceso stackPop_B();
escribir “valor de” n
Fin
```

Pruebas.

```
Expresion a evaluar : (5+5)/6+(6*9)-2

5 5 +
10.000000
6 /
1.666667
6 9 *
54.000000
+
55.666667
2 -
53.666667

Resultado: 53.666657
```

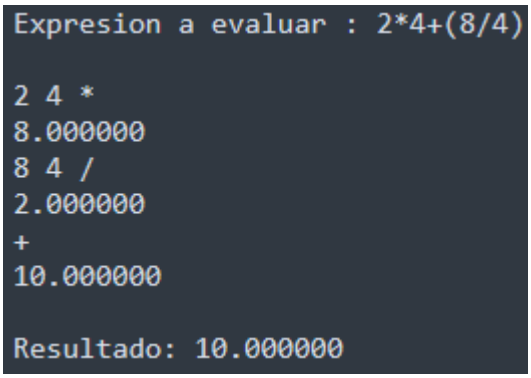
En este ejemplo se muestran los resultados parciales cada que se topa con un operador resolviéndolos con sus respectivos operandos.

```
Expresion a evaluar : 2+3*5/2-1

2 3 5 *
15.000000
2 /
7.500000
+
9.500000
1 -
8.500000

Resultado: 8.500000
```

Un ejemplo con el mismo funcionamiento sin paréntesis.



Misma funcionalidad con un ejemplo más pequeño.

Conclusiones.

Enoch Joaquín Álvarez Goñi

Este proyecto honestamente estuvo complicado, lo más complicado fue el poder extraer los datos para poder evaluarlos y acomodarlos y además intentar resolver la ecuación, en general fue un reto.

Jorge Ramón Figueroa Maya

Gracias a esta labor pude reforzar mis conocimientos sobre los contenedores Stack. Aprendí a identificar cual era la conveniencia de cada uno de estos contenedores en los problemas de aplicación profesional, así como su propósito.

José Emiliano Figueroa Hernández

Al mismo tiempo que elaboraba este proyecto me di cuenta de que las estructuras auto referenciadas son demasiado importantes en el campo laboral ya que no solo nos ayuda a mantener un orden en el programa como desarrolladores, si no que estos nos ayudan a que el compilador pueda manejar estos datos de manera más sencilla ahorrando memoria y tiempo.

Fuentes de información.

Este proyecto lo desarrollamos solos, tratamos de no utilizar tutoriales, ejemplos de proyectos, entre otro tipo de apoyo, ya sea audiovisual o de texto, como códigos y ejercicios, todos los problemas los solucionamos en archivos externos con nombres como “prueba1.c”, los cuales, en nuestro repositorio de GitHub está repleto de estos archivos.

Sin embargo, con el poco avance que teníamos a través de los días, pudimos concluir con el proyecto de manera efectiva y concisa.