UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Ingeniería en sistemas de Información

Sintaxis y semántica de los lenguajes – 2023

TURNO: Noche CURSO: K2055

DOCENTE A CARGO: Ing. Roxana Leituz

AYUDANTE:

TRABAJO PRÁCTICO TEORICO N°2

"Autómatas"

ÁREA TEMÁTICA: Autómatas

GRUPO N° 30

Rodríguez Lucas Ariel

Golato Barcia Ivan Nahuel

Sayago Pablo

Rabahia Maron Leonel

Schinca Mauro

FECHA DE VENCIMIENTO: 1/10/2023

FECHA DE PRESENTACIÓN:1/10/2023

FECHA DE DEVOLUCIÓN: __/__/__

CALIFICACIÓN: _____ FIRMA PROFESOR: _____

Índice:

Punto 1

Autómata:

Para reconocer los números decimales, octales y hexadecimales, se decidió hacer un único autómata que reconozca los tres y diferenciarlos dependiendo del estado final en el que estos terminen.

Diagrama:

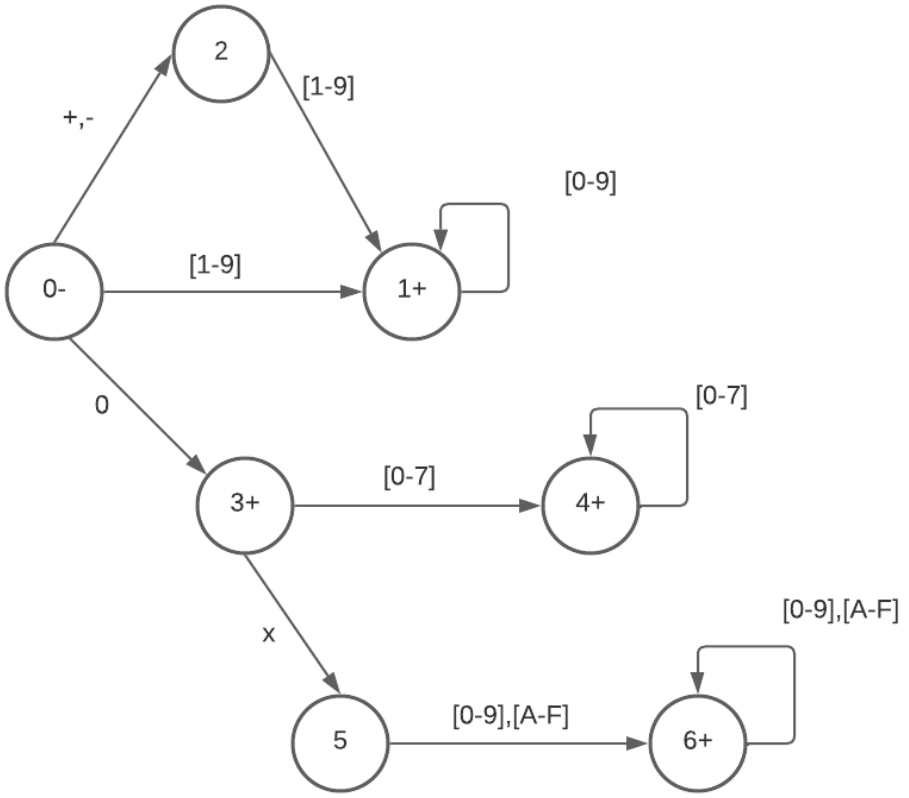


Tabla de transiciones:

	0	"1-7"	"8-9"	A-F	x	"-,+"
0-	3	1	1	7	7	2
1+	1	1	1	7	7	7
2	7	1	1	7	7	7
3+	4	4	7	7	5	7
4+	4	4	7	7	7	7
5	6	6	6	6	7	7
6+	6	6	6	6	7	7
7	7	7	7	7	7	7

Consideraciones:

Así:

- Si el estado final es 1 o 3: es decimal
- Si el estado final es 4: es octal
- Si el estado final es 6: es hexadecimal

Se tomó en cuenta que:

- La cadena “0” pertenece a los números decimales.
- El cero decimal no puede tener signo, por lo que las cadenas “+0” o “-0” son rechazadas.
- Para los hexadecimales, la “x” debe estar en minúscula y los caracteres de A a la F deben estar en mayúscula. Ej: “0xAF12C2” es válido, “0X123” no es válido y “0xab123c” tampoco es válido.

Especificaciones del código:

El programa recibe una cadena por consola y la evalúa, se van imprimiendo los números y diciendo si hubo algún error o si fueron reconocidos como parte del lenguaje.

Se tiene en cuenta que:

- La cadena empieza con un número y termina con un número. Ej: “123$0123$0x123”.
- La cantidad máxima de caracteres que cada número puede tener es de 1000.
- Se lee una única cadena y se evalúa la misma.
- Si hay un error (el número es mayor que 1000 caracteres, se detectaron caracteres que no pertenecen al alfabeto o la palabra no fue reconocida como parte del lenguaje) se deja de evaluar la cadena y se imprimen por pantalla cuantos decimales, octales y hexadecimales se contaron hasta el momento.

Así, el autómata en el código está dividido en el archivo “automataMain.c”, donde se encuentra el main() y donde se encuentra la parte de la lógica referida a leer una cadena y contar los tipos de números, el archivo “automata.c” (y su respectivo header), donde se encuentra la mayor parte de las funciones del autómata, y el archivo “columnas.c” (y su respectivo header) donde está la función que dado un carácter, devuelve a que columna de la tabla de transiciones corresponde.

También, se utilizaron las bibliotecas “ctype.h”, usada por la función isdigit(char) que devuelve si el carácter dado es un número o no, la biblioteca “stdlib.h”, usada por la función system(“pause”) para que espere que se presione una tecla antes de cerrar la consola de comandos una vez finalizada la evaluación de la cadena, y la biblioteca “stdio.h” por sus funciones de entrada y salida por la consola.

El ejecutable “automata.exe” es compilado con el Makefile que se encuentra junto con los otros archivos.

Ejemplos de funcionamiento:

Ejemplo práctico:

```
Ingrese su expresion de numeros intercalados con $ (Maximo 1000 caracteres por numero)
123$01237$0xFBC982$0x2$+12$-43
 1  2  3     Cadena leida
La palabra ingresada es correcta
 0  1  2  3  7     Cadena leida
La palabra ingresada es correcta
 0  x  F  B  C  9  8  2     Cadena leida
La palabra ingresada es correcta
 0  x  2     Cadena leida
La palabra ingresada es correcta
 +  1  2     Cadena leida
La palabra ingresada es correcta
 -  4  3     Cadena leida
La palabra ingresada es correcta
Cantidad de octales 1
Cantidad de decimales 3
Cantidad de hexadecimales 2
Press any key to continue . . .
```

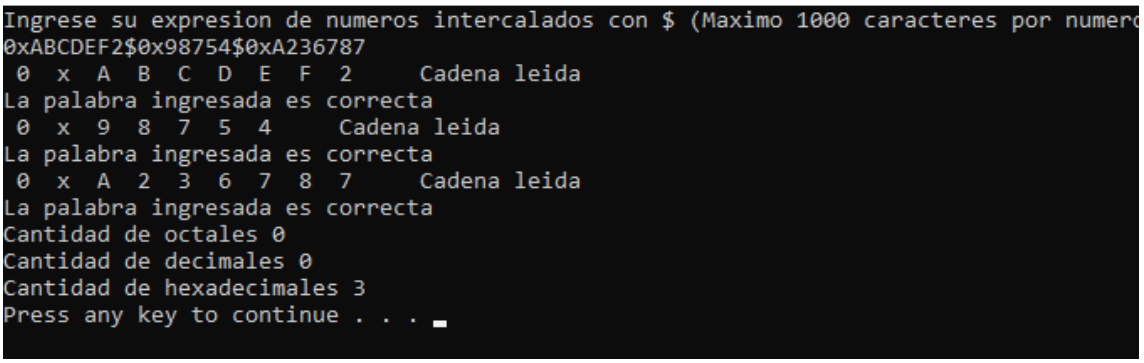
Números decimales:

```
Ingrese su expresion de numeros intercalados con $ (Maximo 1000 caracteres por numero)
123$+123$-123
 1  2  3     Cadena leida
La palabra ingresada es correcta
 +  1  2  3     Cadena leida
La palabra ingresada es correcta
 -  1  2  3     Cadena leida
La palabra ingresada es correcta
Cantidad de octales 0
Cantidad de decimales 3
Cantidad de hexadecimales 0
Press any key to continue . . .
```

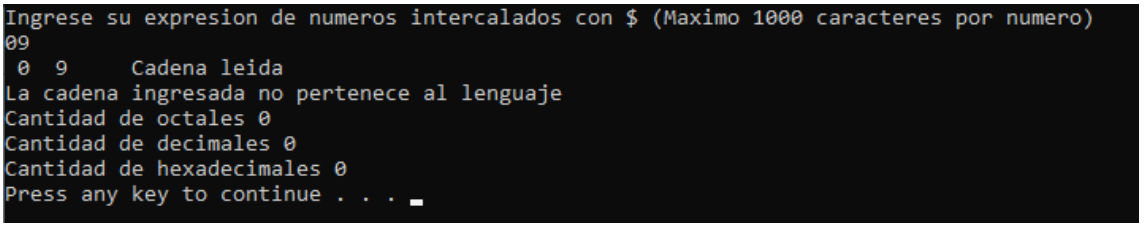
Números octales:

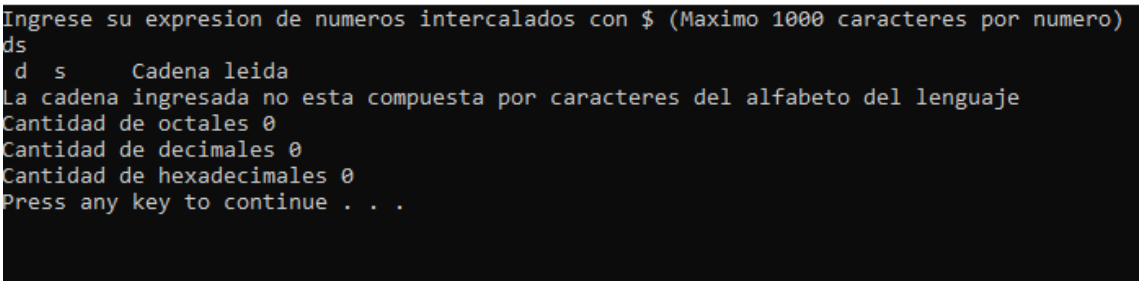
```
Ingrese su expresion de numeros intercalados con $ (Maximo 1000 caracteres por numero)
0123$0765
 0  1  2  3     Cadena leida
La palabra ingresada es correcta
 0  7  6  5     Cadena leida
La palabra ingresada es correcta
Cantidad de octales 2
Cantidad de decimales 0
Cantidad de hexadecimales 0
Press any key to continue . . .
```

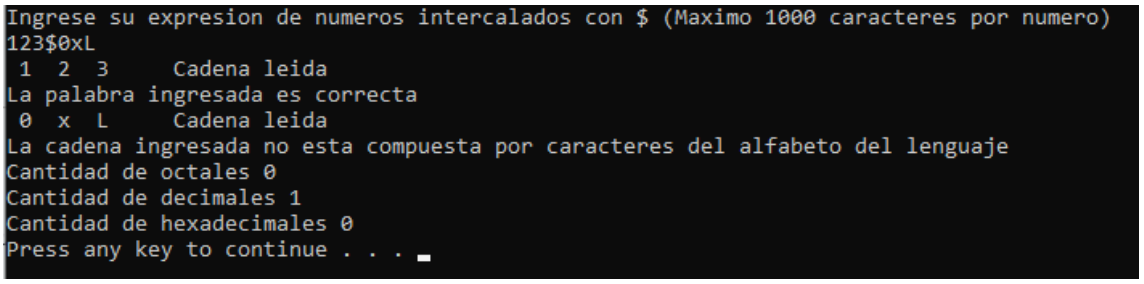
Números hexadecimales:



Errores:







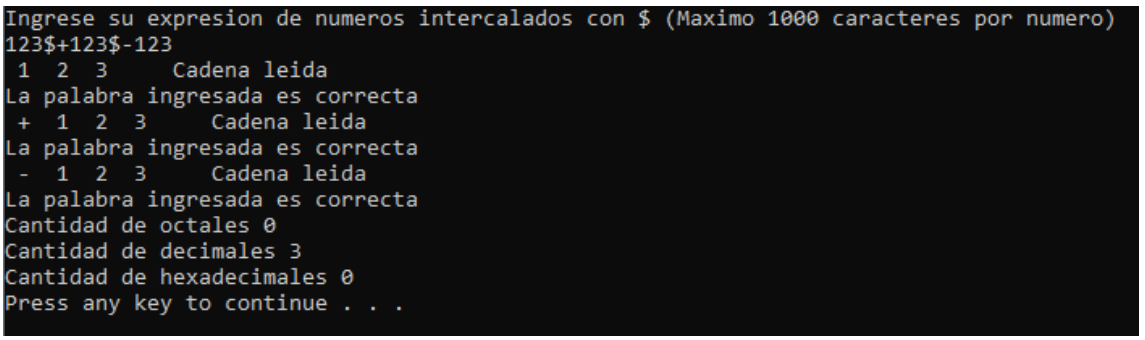
Modo de uso:

Primero debemos compilar nuestros archivos **.c** con los archivos **.h**, para eso utilizamos el **Makefile**.

Luego de tener nuestro archivo automata.exe ya creado procedemos a ejecutarlo.

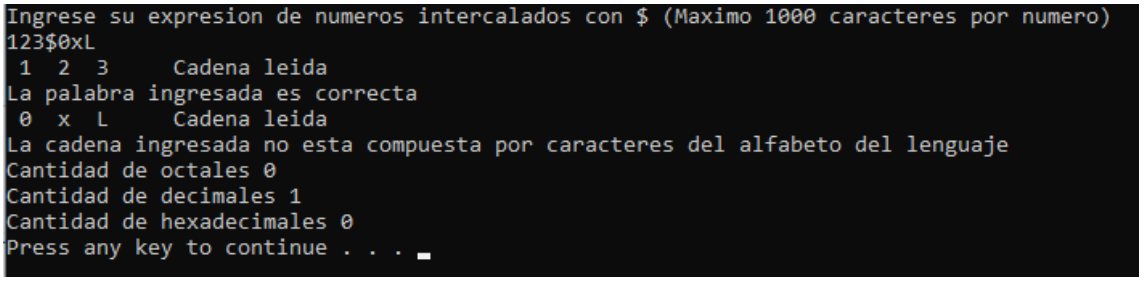
Esto nos llevaría a ejecutar nuestro programa donde debemos introducir una expresión de números tanto decimales con o sin signo, como octales y hexadecimales. Todos ellos intercalados por el símbolo **"$"** que lo utilizaremos como un carácter para saber el final de un número y el comienzo de otro.

Importante: la expresión debe comenzar directamente con un número y terminar también con un número (sin el carácter '$').



En caso de que todos los números ingresados sean correctos el programa nos brinda la cantidad de cada uno, como se muestra en la imagen superior.

En caso contrario, cuando se encuentre con un número que no es parte del lenguaje, parará la ejecución y mostrará la cantidad de números reconocidos hasta el momento



Como vemos también nos muestra cuál fue el caso erróneo.

Funciones y estructura del código:

automataMain.c:

```c
#include "automata.h"
#include "columnas.h"
#include <ctype.h>
#include<stdio.h>
#include<stdlib.h> //system()

#define MAX_LENGTH 1000
```

Se incluyen las bibliotecas a utilizar y se define una macro para el largo máximo de un número. El header columnas.h contiene la función columna() y isAToF(), que las utilizamos para determinar la columna de la tabla de transición y si un carácter pertenece al conjunto [A-F] respectivamente.

```c
int main()
{
    char palabra[(MAX_LENGTH + 1)];
    int c=0,i=0,octal=0,decimal=0,hexa=0;
    printf("Ingrese su expresion de numeros intercalados con $ (Maximo %d caracteres por numero) \n", MAX_LENGTH);

    while(c!='\n')
    {
        i=0;

        //Lee una cadena
        c=getchar();
        while(c!='$'&& c!='\n')
        {
            palabra[i]=c;
            printf(" %c ",palabra[i]);
            i++;
            c=getchar();
        }

        printf("    Cadena leida \n");

        if(i>MAX_LENGTH){      You, ayer • Cambios en automata y verificacion calculadora
            printf("La cadena leida tiene más caracteres de lo permitido \n");
            break;
        }


        // Agrega el caracter de fin de cadena al final de la cadena leida
        palabra[i]='\0';

        if(cadenaPerteneceConstantesEnteras(palabra))
        {
            if(esPalabraConstantesEnteras(palabra,&octal,&decimal,&hexa))
            {
                printf("La palabra ingresada es correcta \n");
            }
            else
            {
                printf("La cadena ingresada no pertenece al lenguaje \n");
                break;
            }
        }
        else
        {
            printf("La cadena ingresada no esta compuesta por caracteres del alfabeto del lenguaje \n");
            break;
        }
    }
    printf("Cantidad de octales %d \n",octal);
    printf("Cantidad de decimales %d \n",decimal);
    printf("Cantidad de hexadecimales %d \n",hexa);

    system("pause");
    return 0;
}
```

El main consiste principalmente en ir tomando cada carácter de la entrada estándar y guardarlos en un array, que se sobrescribe constantemente por la aparición del carácter **“$”.**

Y al encontrar el carácter “/n” (salto de línea) sin haber tenido ningún error muestra la cantidad de números de cada conjunto encontrado. En caso contrario nos muestra el error pertinente y deja de reconocer números.

```c
// Devuelve si es una palabra reconocida por el automata de constantes enteras y aumenta el contador acorde
int esPalabraConstantesEnteras(char*palabra,int *octal,int *decimal,int *hexa)
{
    int estadoFinal = automataConstantesEnteras(palabra);
    if(esDecimal(estadoFinal)){
        *decimal+=1;
        return 1;
    }
    if(esOctal(estadoFinal)){
        *octal+=1;
        return 1;
    }
    if(esHexa(estadoFinal)){
        *hexa+=1;
        return 1;
    }
    return 0;
}
```

La función esPalabraConstantesEnteras() recibe los punteros a enteros con el fin de usarlos como contadores de los tipos de números que vamos recibiendo.

automata.c:

```c
// Devuelve si la cadena esta compuesta por caracteres del alfabeto de constantes enteras
int cadenaPerteneceConstantesEnteras(char *  palabra)
{
    int i=0;
    while(palabra[i]!='\0')
    {
        if(isdigit(palabra[i]) ||
                palabra[i]=='+'||
                palabra[i]=='-'||
                palabra[i]=='A'||
                palabra[i]=='x'||
                palabra[i]=='B'||
                palabra[i]=='C'||
                palabra[i]=='D'||
                palabra[i]=='E'||
                palabra[i]=='F'||
                palabra[i]=='$')
        {
            i++;
        }
        else
        {
            return 0;
        }
    }
    return 1;
}
```

La función cadenaPerteneceConstantesEnteras() recibe lo que podría ser un número y determina si todos sus miembros pertenecen a nuestro alfabeto de constantes enteras.

```
// Devuelve si la cadena esta compuesta por caracteres del alfabeto de operadores aritmeticos
int cadenaPerteneceCalculadora(char * palabra){
    int i=0;
    while(palabra[i]!='\0' && palabra[i]!='\n')
    {
        printf("%c", palabra[i]);
        if(isdigit(palabra[i]) ||
            palabra[i]=='+'||
            palabra[i]=='-'||
            palabra[i]=='*'||
            palabra[i]=='/')
        {
            i++;
        }
        else
        {
            return 0;
        }
    }
    return 1;
}
```

La función cadenaPerteneceCalculadora() recibe lo que podría ser un número y determina si todos sus miembros pertenecen a nuestro alfabeto de caracteres posibles que reconoce la calculadora (esto va a ser usado en el punto 3).

```
// Devuelve si una cadena pertenece al lenguaje de los decimales
int esPalabraDecimal(char*palabra){
    return esDecimal(automataConstantesEnteras(palabra));
}
```

La función esPalabraDecimal() devuelve su una determinada cadena pertenece al grupo de los números decimales (esto va a ser usado en el punto 3).

```
//Devuelve si el estado final corresponde a un numero decimal
int esDecimal(int estadoFinal){
    return estadoFinal == 1 || estadoFinal == 3;
}

//Devuelve si el estado final corresponde a un numero hexadecimal
int esHexa(int estadoFinal){
    return estadoFinal == 6;
}

//Devuelve si el estado final corresponde a un numero octal
int esOctal(int estadoFinal){
    return estadoFinal == 4;
}
```

Las funciones esDecimal(), esOctal(), y esHexa() dado un estado final devuelve si corresponde a un estado de aceptación de los respectivos grupos de constantes enteras.

```c
//Devuelve el estado final del automata de constantes enteras
int automataConstantesEnteras(char*palabra){
    int tt[8][6]={{3,1,1,7,7,2},
                  {1,1,1,7,7,7},
                  {7,1,1,7,7,7},
                  {4,4,7,7,5,7},
                  {4,4,7,7,7,7},
                  {6,6,6,6,7,7},
                  {6,6,6,6,7,7},
                  {7,7,7,7,7,7}};
    int estado = 0;
    int i=0 ;
    while(palabra[i]!='\0'&& estado!=7) {
        estado=tt[estado][columna(palabra[i])];
        i++;
    }
    return estado;
}
```

La función automataConstantesEnteras() que recibe la cadena (número) a analizar y con la tabla de transición va cambiando el estado utilizando la función columna que se aloja en **columna.h** y por último devuelve el estado final del autómata.

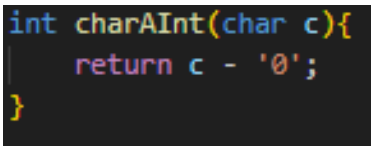
```c
#include "columnas.h"      LucasArielRodriguez20, la semana pasada • first commit …
#include<stdio.h>

// Devuelve si c esta en el rango [A-F]
int isAToF(int c){
    if(c>='A'&&c<='F')
        return 1;
    else
        return 0;
}

// Dado un caracter devuelve cual es la columna que le corresponde en la tabla de transiciones del automata de constantes enteras
int columna(int c)
{
    if(c=='0')
        return 0;
    if(c>='1' && c<='7')
        return 1;
    if(c=='8' || c=='9')
        return 2;
    if(isAToF(c))
        return 3;
    if(c=='x')
        return 4;
    if(c=='-' || c=='+')
        return 5;
}
```

Por ultimo dentro de **columna.c** podemos encontrar estas dos funciones, isAToF() determina si c pertenece al conjunto [A-F] y columna nos devuelve el valor que coincide con la columna de la tabla de transición según el carácter que sea c.

Punto 2

Carácter numérico a int

Para obtener el valor entero de un carácter numérico dado simplemente se debe restar a este carácter ‘0’ ya que los caracteres son solo un subconjunto de los enteros en base a su código ASCII y sus valores en ASCII están en orden ascendente empezando por 0.



```
int charAInt(char c){
    return c - '0';
}
```

Esta función que implementa lo descripto anteriormente se encuentra en “calculadora.c”.

Punto 3

Calculadora

La calculadora consiste en dos partes fundamentalmente, la primera parte consiste en leer la expresión aritmética y analizar qué está correctamente armada, tanto la expresión en su totalidad como cada uno de sus miembros.

La lógica para esta validación se encuentra en "***verificacion.c***":

```c
#include "./automata/automata.h"
#include <stdio.h>

int verificacion(){
    int c,i=0,estado=0,top=0;
    int ultimoFueNumero = 0;
    char expresionAritmetica[1000];
    char numero[10];
    while((c=getchar())!='\n')
    {
        if(c!=32||c!=9)//ignoramos esapcios y tabs de la expresion
        {
            expresionAritmetica[i]=c;
            i++;
        }

    }
    //obtengo la expresion en su totalidad
    ungetc(c,stdin);
    expresionAritmetica[i]='\n';
    top=i-1;
    if(!cadenaPerteneceCalculadora(expresionAritmetica))//verificamos la expresion leida
    {
        printf("hay un caracter incorrecto en la expresion \n");
        return 0;
    }
```

```
    //si esta todo correcto seguimos analizando
    i=0;
    while(expresionAritmetica[i]!='\n')//mientras no se acabe la expresion
    {
        //encontramos un operador o un numero en la expresion
        if(isdigit(expresionAritmetica[i]) && !ultimoFueNumero)///es un numero
        {
            int j = 0;
            while(isdigit(expresionAritmetica[i]))//recolecto los numeros de la expresion
            {
                numero[j]=expresionAritmetica[i];
                j+=1;
                i+=1;
            }
            if(esPalabraDecimal(numero))//verifico que ese numero es realmente un decimal
            {
                ultimoFueNumero = 1;// Dejo la marca que el ultimo fue un numero, por lo que el siguiente debe ser un operador
            }
            else
            {       LucasArielRodriguez20, anteayer • agregaron cambios en la calculadora
                printf("ERROR ingreso de numero invalido\n");
                return 0;
            }
        }
        else if(ultimoFueNumero)//no es un numero entonces es un operador
        {
            ultimoFueNumero = 0;
            i++;
        }
        else{
            printf("ERROR expresion invalida\n");
            return 0;
        }
    }
    if(ultimoFueNumero == 1)
    {
        while (top!=-1)//devulevo la entrada en el mismo sentido que la ingrese para no afectar la precedencia
        {
           ungetc(expresionAritmetica[top],stdin);
           top--;
        }
        return 1;
    }
    else
        return 0;
}
```

Así esta función implementa algunas funciones del autómata del punto 1 y verifica que no haya caracteres que no deberían (esto siendo un análisis léxico).

Además, se usa la variable *ultimoFueNumero* para verificar que la forma en la cual esta formada la expresión sea correcta, teniendo que empezar siempre con un número y terminando con un número (esto siendo un análisis sintáctico)

Ya teniendo la expresión verificada podemos continuar con la siguiente parte que consiste en pasar de una expresión aritmética infija a postfija, con el fin de resolver el problema de la precedencia en las operaciones. Para hacer esto seguimos el siguiente algoritmo que utiliza dos pilas, una final y otra de operadores.

“stack.c” y “stack.h”:

```h
#ifndef STACK_H_INCLUDED
#define STACK_H_INCLUDED
#include <stdlib.h>// free() malloc()

LucasArielRodriguez20, la semana pasada | 1 author (LucasArielRodriguez20)
typedef struct Nodo{
    int valor;
    struct Nodo * sgte;
}nodo;

LucasArielRodriguez20, la semana pasada | 1 author (LucasArielRodriguez20)
typedef struct NodoN{
    double valor;
    struct NodoN * sgte;
}nodoN;

typedef nodo* ptrNodo;//el equivalente a *& que seria **
typedef nodoN* ptrNodoN;

void push(ptrNodo * ,int );
int pop(ptrNodo* );
void pushN(ptrNodoN * ,double );
double popN(ptrNodoN* );
#endif
```

```c
#include "stack.h"
void push(ptrNodo * stack,int dato)
{
    ptrNodo p=(ptrNodo)malloc(sizeof(nodo));//new nodo
    p->valor=dato;
    p->sgte=*stack;//desreferencio para acceder a la pila
    *stack=p;
}

int pop(ptrNodo* stack)
{
    int x=(*stack)->valor;
    ptrNodo aux=(*stack);
    *stack=aux->sgte;
    free(aux);
    return x;
}

void pushN(ptrNodoN* stack,double dato)
{
    ptrNodoN p=(ptrNodoN)malloc(sizeof(nodoN));//new nodo
    p->valor=dato;
    p->sgte=*stack;//desreferencio para acceder a la pila
    *stack=p;
}

double popN(ptrNodoN* stack)
{
    double x=(*stack)->valor;
    ptrNodoN aux=(*stack);
    *stack=aux->sgte;
    free(aux);
    return x;
}
```

Primero, en un archivo se define las funciones para usar pilas, una pila en la cual se guardan int y otra en la cual se guardan double.

calculadora.c:

```c
int main()
{
    struct NodoN * stack=NULL;
    struct Nodo * pilaPolaca=NULL;
    int c=0;
    double top=0;
    printf("\n Ingrese una expresion aritmetica del modo '1*2+5/5' \nActualmente la calculadora solo admite * / - + como operandos \n");
    if(!verificacion())
    {
        printf("ERROR expresion aritmetica erronea\n");
        return 0;
    }

    printf("    La expresion es valida \n");
    infijaApostfija(&pilaPolaca);
    ///comienzo calculadora
     while((c=numeroADecimal(&stack,&pilaPolaca))!=EOF)
    {
        switch(c)
        {
            case '+':
                pushN(&stack,(popN(&stack)+popN(&stack)));
                break;
            case '-':
                top=popN(&stack)*-1;
                pushN(&stack,(popN(&stack)+top));
                break;
            case '*':
                pushN(&stack,(popN(&stack)*popN(&stack)));
                break;
            case '/':
                top=popN(&stack);
                if(top!=0)
                {
                    pushN(&stack,(popN(&stack)/top));
                }
                else{
                    printf("Error division por 0");
                    return 0;
                }
                break;
            case '0':
                break;
            case ' ':
                break;
            case '\t':
                break;
            case '$':
                break;
            case '!':
                break;
            case '\n':
                printf("El resultado es %.2f \n",popN(&stack));
                system("pause");
                break;
            default:
                printf("Es un dato no valido %c \n",c);
                return 0;
                break;
        }
    }
    return 0;
}
```

La calculadora polaca funciona leyendo datos desde la pilaPolaca que obtenemos de la función infijaApostfija(), por cómo se transformó la expresión ahora el switch va a ir encontrando sucesivamente dos numero y un operador, lo cual va a permitirle realizar las operaciones que vaya leyendo con los números que hay guardados en la pila y finalmente guardando los resultados en la pila para seguir operando sucesivamente.

Y para obtener los números utilizamos la función de numeroAdecimal() que consiste en ir transformando los número de char a int y sumarlos en una sumatoria de doubles teniendo en cuenta la posición de los caracteres, por lo tanto el número obtenido va a una nueva pila pero en este caso una de doubles que será nuestra pila final.

```c
void infijaApostfija(ptrNodo * pilaPolaca){
    int c=0,aux;
    struct Nodo * stack=NULL;
    struct Nodo * operadores=NULL;
    struct Nodo * auxPila=NULL;
     ///infija a postfija
    while((c=getchar())!='\n')//continuo mientras c no sea fin de archivo y sea diferente a salto de linea
    {
    switch(c)
    {
        case '+':
            push(&stack,' ');
            if(operadores == NULL)//si la pila esta vacia agrego el operador
                push(&operadores,c);
            else
            {
                if(operadores->valor =='*'|| operadores->valor =='/')
                {
                    //sino esta vacia y en la cima de la pila hay un * o una /
                    while(operadores)//vacio la pila de operadores
                        push(&stack,pop(&operadores));// los operadores van a la pila principal
                    push(&operadores,c);//por ultimo guardo el + en la pila secundaria
                }
                else
                {
                    //tengo en mi pila secundaria un + o -
                    push(&stack,pop(&operadores));//extraigo el operador de mi pila secundaria y lo guardo en la principal
                    push(&operadores,c);//por ultimo guardo el + en la pila secundaria
                }
            }
            break;
        case '-':
            push(&stack,' ');
        //mismo procedimiento para el -
            if(operadores == NULL)
                push(&operadores,c);
            else
            {
                if(operadores->valor =='*'|| operadores->valor =='/')
                {
                    while(operadores)
                    {
                        push(&stack,pop(&operadores));
                    }
                    push(&operadores,c);
                }
                else
                {
                    push(&stack,pop(&operadores));
                    push(&operadores,c);
                }
            }
            break;
```

```c
        case '*':
            push(&stack,' ');
            if(operadores == NULL)//si la pila esta vacia agrego el operador
                push(&operadores,c);
            else
            {
                if(operadores->valor =='/'||operadores->valor =='*')
                {
                    //tengo en mi pila secundaria un * o /
                    push(&stack,pop(&operadores));//extraigo el de la pila secundaria y lo coloco en la principal
                    push(&operadores,c);//guardo el * en la pila secundaria
                }
                else
                    //tengo en mi pila secundaria un + o -
                    push(&operadores,c);//guardo el * en la pila secundaria
            }
            break;
        case '/':
            push(&stack,' ');//separar numeros con operadores
            if(operadores == NULL)
                push(&operadores,c);
            else
            {
                if(operadores->valor =='*'||operadores->valor =='/')
                {
                    push(&stack,pop(&operadores));
                    push(&operadores,c);
                }
                else
                    push(&operadores,c);
            }
            break;
        case'\n'://ignoro saltos de linea
            break;
        case'\t'://ignoro tabulaciones
            break;
        case' '://ignoro espacios
            break;
        default:
            push(&stack,c);//guardo un numero en la pila principal
            break;
        }
    }
    while(operadores)// si queda algun operador en mi pila secundaria lo extraigo
        push(&stack,pop(&operadores));
    ///pila cargada con la expresion
    push(&stack,'\n');
    while(stack)
        push(pilaPolaca,pop(&stack));//devuelvo la exprecion en polaca inversa a la entrada
}
```

Así, la función infijaApostfija() convierte la expresión ingresada por consola en notación infija a una pila ordenada de acuerdo a la notación postfija.

```
int numeroADecimal(ptrNodoN * stack,ptrNodo *pilaPolaca)
{
    int c=0,contador=0,contadorFraccionario=0;
    double sum=0.0;
    double numero[1000],fraccion[1000];//el numero en su parte entera y en su parte fraccionaria
    if (isdigit(c=pop(pilaPolaca)))
    {
            //decimal
            //mismo procedimiento para el decimal solo que cambia el uso de pow , en este caso se utiliza para formar la parte fraccionaria
                while (isdigit(c))
                {
                    sum*=10;
                    sum+=(c-'0');
                    c=pop(pilaPolaca);
                }
                contador--;
                for(int i=0;contador>=0;i++)
                {
                    sum*=10;
                    sum+= numero[i];
                    contador--;
                }
                pushN(stack,sum);
                sum=0;
                contador=0;
                contadorFraccionario=0;
                push(pilaPolaca,c);
                return '0';
    }
    else
    return c;//devuelvo el operador o espacio o tabulacion
}
```

La función numeroAdecimal() realiza el push por cada número obtenido y si lee algo que no es un número lo devuelve para que el switch de la calculadora polaca realice la operación , lo ignore , devuelva un error o por último nos dé el resultado que va estar en el tope de la pila final.

Ejemplos de funcionamiento:

```
 Ingrese una expresion aritmetica del modo '1*2+5/5'
Actualmente la calculadora solo admite * / - + como operandos
2*3+5*2
2*3+5*2    La expresion es valida
El resultado es 16.00
Press any key to continue . . .
```

```
 Ingrese una expresion aritmetica del modo '1*2+5/5'
Actualmente la calculadora solo admite * / - + como operandos
2/3
2/3    La expresion es valida
El resultado es 0.67
Press any key to continue . . .
```

```
 Ingrese una expresion aritmetica del modo '1*2+5/5'
Actualmente la calculadora solo admite * / - + como operandos
1/0
1/0    La expresion es valida
Error division por 0
Press any key to continue . . .
```

```
 Ingrese una expresion aritmetica del modo '1*2+5/5'
Actualmente la calculadora solo admite * / - + como operandos
+323+4
+323+4    ERROR Expresion invalida
Press any key to continue . . .
```

```
 Ingrese una expresion aritmetica del modo '1*2+5/5'
Actualmente la calculadora solo admite * / - + como operandos
asd
a    ERROR Hay un caracter incorrecto en la expresion
Press any key to continue . . .
```

Modo de uso:

Primero ejecutamos el comando **make** en nuestra consola y obtendremos el **calculadora.exe**

Al ejecutarlo nos va mostrar la siguiente pantalla donde ingresamos la expresión **sin espacios**

```
 Ingrese una expresion aritmetica del modo '1*2+5/5'
Actualmente la calculadora solo admite * / - + como operandos
```

Al ingresar la expresión oprimiendo el botón **enter** obtenemos el resultado.

Se muestran hasta 2 posiciones después de la coma, redondeando de ser necesario.

Referencias:

Enlace al material que nos con la parte de pasar de notación infija a una pila polaca:

https://www.youtube.com/watch?v=kpLlufrQiTk&t=2s

La calculadora principalmente se inspiró en muchas ideas del libro de Brian Kernighan y Dennis Ritchie, específicamente el capítulo cuatro.