

Ingeniería en sistemas de información.

Sintaxis y semántica de los lenguajes.

TRABAJO PRÁCTICO TEÓRICO.

ÁREA TEMÁTICA: Autómatas y Pilas.

GRUPO N° 22

Apellido y Nombre	N° de Legajo
Gaetan, María Luz	163627-3
Laino, Ramiro Angel	175883-4
Lamothe, Genty Clarke	
Mendiolar Colombo, Nahuel Nehuen	169188-0
Rolando, Sebastian	176.587-5

CURSO: K2055

DOCENTE A CARGO: Roxana Leituz

FECHA DE VENCIMIENTO: 1/10/2023

FECHA DE PRESENTACIÓN: 1/10/2023

FECHA DE DEVOLUCIÓN: __/__/__

CALIFICACIÓN: _____

FIRMA DOCENTE: _____

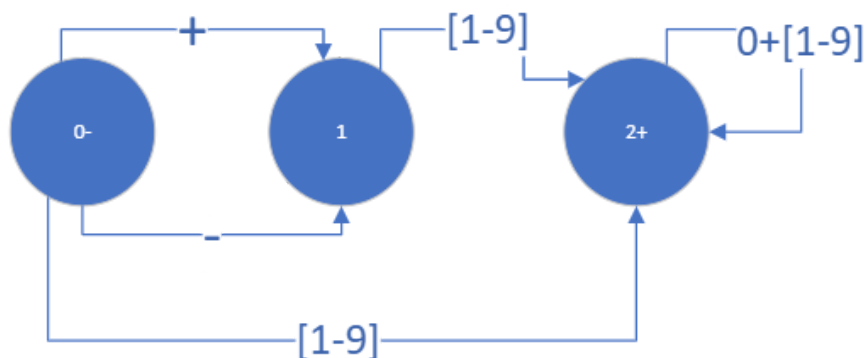
Índice

Ejercicio 1	2
Ejercicio 2	8
Ejercicio 3	10
Conclusión	17

1) Dada una cadena que contenga varios números que pueden ser decimales, octales o hexadecimales, con o sin signo para el caso de los decimales, separados por el carácter '\$' , reconocer los tres grupos de constantes enteras, indicando si hubo un error léxico , en caso de ser correcto contar la cantidad de cada grupo. Debe diagramar y entregar el o los autómatas utilizados y las matrices de transición. La cadena debe ingresar por línea de comando o por archivo.

Dibujamos los autómatas correspondientes para cada caso

Decimal: consideramos como primer carácter los signos +,-, y que el número comience con 1, no con 0 para no confundir con un número octal.



$$M = (Q, E, T, q_0, F)$$

$$Q = \{0, 1, 2\}$$

$$E = \{+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$q_0 = 0$$

$$F = \{2\}$$

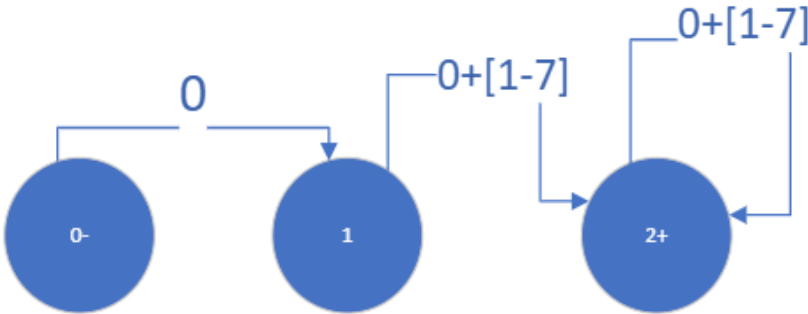
Estado	0	+	-	[1-9]
0	-	1	1	2
1	-	-	-	2
2+	2	-	-	2

Tabla de transición

Estado	0	+	-	[1-9]	
0	3	1	1	2	
1	3	3	3	2	
2+	2	3	3	2	
3	3	3	3	3	Estado de rechazo

Tabla de transiciones completa

Octales: Los números octales comienzan con 0 y tienen caracteres de 0 al 7.



$M = (Q,E,T,q_0,F)$
 $Q = \{0,1,2\}$
 $E = \{0,1,2,3,4,5,6,7\}$
 $q_0 = 0$
 $F = \{2\}$

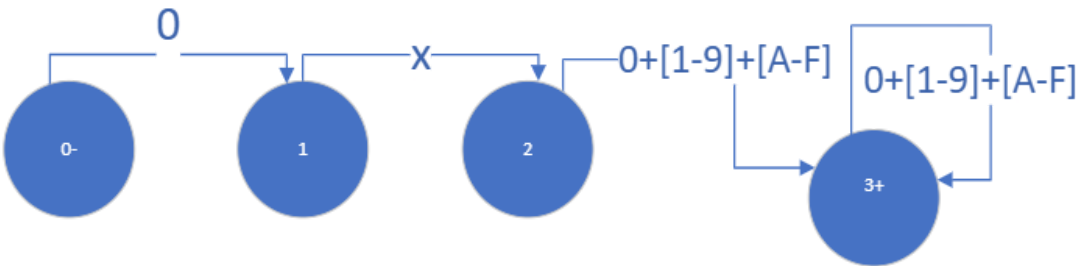
Estado	0	[1-7]
0	1	-
1	2	2
2+	2	2

Tabla de transiciones

Estado	0	[1-7]	
0	1	3	
1	2	2	
2+	2	2	
3	3	3	(Estado de rechazo)

Tabla de transiciones completa

Hexadecimales: comienzan con 0x y tiene caracteres del 0 al 9 y del [a-f]



$M = (Q,E,T,q_0,F)$
 $Q = \{0,1,2,3\}$
 $E = \{x,0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$
 $q_0 = 0$
 $F = \{3\}$

Estado	0	x	[1-9]+[A-F]
0	1	-	-
1	-	2	-
2	3	-	3
3+	3	-	3

Tabla de transiciones

Estado	0	x	[1-9]+[A-F]	
0	1	4	4	
1	4	2	4	
2	3	4	3	
3+	3	4	3	
4	4	4	4	(Estado de rechazo)

Tabla de transiciones completa

Para resolver este ejercicio utilizamos en total nueve (9) funciones:

- *VerificaDecimal*, *VerificaOctal*, *VerificaHexadecimal*: nos permite verificar si el carácter pertenece al alfabeto del autómata.
- *ColumnaDecimal*, *ColumnaOctal*, *ColumnaHexadecimal*: retorna el número de columna, de la tabla de transición.
- *EsDecimal*, *EsOctal*, *EsHexadecimal*: verifica si se llega a un estado de transición final del autómata.

```

16
17  int VerificaDecimal (char *s) {
18      unsigned i;
19      for (i=0; s[i]; i++)
20          if (! (isdigit(s[i]) || s[i] == '+' || s[i] == '-')) return 0;
21      return 1;
22  } /* fin Verifica */
23

```

Captura de la función *VerificaDecimal*

```
40
41 int ColumnaDecimal (int c) {
42     switch (c) {
43         case '0': return 0;
44         case '+': return 1;
45         case '-': return 2;
46
47         default /* es digito 1-9 */: return 3;
48     }
49 } /* fin Columna */
50
```

Captura de la función ColumnaDecimal

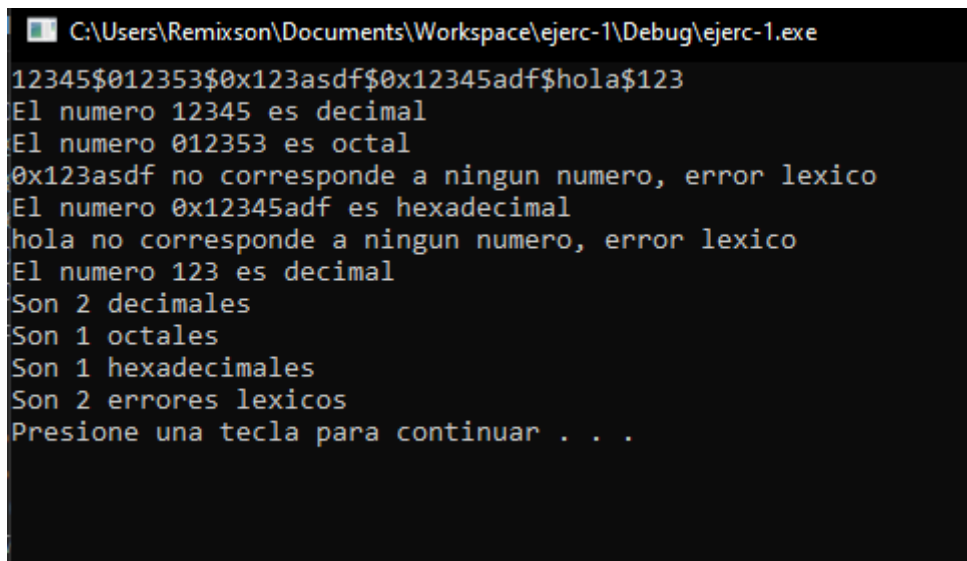
```
66
67 int EsDecimal (const char *cadena) { /* Automata 1 */
68     static int tt [4][4] = {{3,1,1,2}, /* Tabla de Transiciones */
69         {3,3,3,2},
70         {2,3,3,2},
71         {3,3,3,3}};
72     int e = 0; /* estado inicial */
73     unsigned int i = 0; /* recorre la cadena */
74     int c = cadena[0]; /* primer caracter */
75
76     while (c != '\0') {
77         e = tt[e][ColumnaDecimal(c)]; /* nuevo estado */
78         c = cadena[++i]; /* proximo caracter */
79     }
80     if (e == 2) /* estado final */ return 1;
81     return 0;
82 } /* fin EsPalabra */
83
```

Captura de la función EsDecimal

El *main()* utiliza la función *strtok(char *str, const char *delim)*, la cual toma una cadena como parámetro de entrada y la va cortando cuando encuentra el caracter delimitador, que en nuestro caso es '\$', cada cadena luego es verificada por cada autómatas, y si cumple con las condiciones de los *if*, el cual se cuenta cuantas cadenas numéricas de cada tipo hay y la cantidad de errores léxicos, que serían las palabras que no pertenecen a ninguno de los lenguajes.

```
120
121 int main () {
122     char s1[100];
123     scanf("%s", s1);
124
125     const char s[2] = "$";
126     char *token;
127
128     unsigned int w = 0; //cantidad errores lexicos
129     unsigned int x = 0; //cantidad numeros decimales
130     unsigned int y = 0; //cantidad numeros octales
131     unsigned int z = 0; //cantidad numeros hexadecimales
132
133     /* get the first token */
134     token = strtok(s1, s);
135
136     /* walk through other tokens */
137     while( token != NULL ) {
138         if( VerificaDecimal(token) && EsDecimal(token)){
139             x++;
140             printf( "El numero %s es decimal \n", token);
141         }else if( VerificaOctal(token) && EsOctal(token)){
142             y++;
143             printf( "El numero %s es octal \n", token);
144         }else if( VerificaHexadecimal(token) && EsHexadecimal(token)){
145             z++;
146             printf( "El numero %s es hexadecimal \n", token);
147         } else{
148             w++;
149             printf("%s no corresponde a ningun numero, error lexico \n", token);
150         }
151
152         token = strtok(NULL, s);
153     }
154     printf("Son %d decimales \n", x);
155     printf("Son %d octales \n", y);
156     printf("Son %d hexadecimales \n", z);
157     printf("Son %d errores lexicos \n", w);
158     return(0);
159 }
160
```

Captura de la función main



```
C:\Users\Remixson\Documents\Workspace\ejerc-1\Debug\ejerc-1.exe
12345$012353$0x123asdf$0x12345adf$hola$123
El numero 12345 es decimal
El numero 012353 es octal
0x123asdf no corresponde a ningun numero, error lexico
El numero 0x12345adf es hexadecimal
hola no corresponde a ningun numero, error lexico
El numero 123 es decimal
Son 2 decimales
Son 1 octales
Son 1 hexadecimales
Son 2 errores lexicos
Presione una tecla para continuar . . .
```

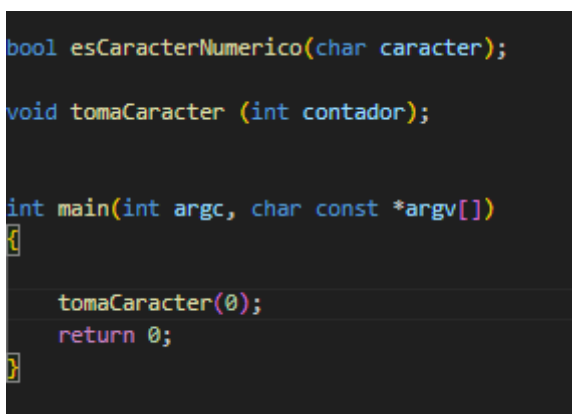
Captura del programa en ejecución

2) Debe realizar una función que reciba un carácter numérico y retorne un número entero.

Para resolver este ejercicio utilizamos dos funciones:

- *esCaracterNumerico*: nos permite determinar cuando el carácter que ingresa en el autómata es numérico.
- *tomaCaracter*: función recursiva para pedir entrada de datos por consola.

Dentro del *main* llamamos a la función recursiva para entrada de datos.



```
bool esCaracterNumerico(char caracter);

void tomaCaracter (int contador);

int main(int argc, char const *argv[])
{
    tomaCaracter(0);
    return 0;
}
```

Captura de la función main

Luego dentro de la función recursiva *tomaCaracter* validamos que el carácter ingresado es numérico.

```

void tomaCaracter (int contador){
char caracter;
bool continuar;

printf("----- | Ingrese un caracter numerico: ");
scanf("%c", &caracter);

if (esCaracterNumerico(caracter))
{
printf("----- | \n");
printf("| q%d+ | | El caracter %c es numerico y su valor entero es %d\n", contador, caracter, caracter - '0');
printf("----- | \n");
printf("----- | \n");
printf("----- | \n");
printf("----- | \n");
printf("----- | \n");
tomaCaracter(contador + 1);
}
else
{
continuar = false;
printf("----- | \n");
printf("| q%d- | | El caracter %c no es numerico\n", contador, caracter);
printf("----- | \n");
}

if (continuar)
{
tomaCaracter(contador);
}
}

```

Captura de la función tomaCaracter

Cuando se ejecuta este programa, se pide ingresar por línea de comando el ingreso de caracteres numéricos y el programa corta la ejecución al detectar un carácter no numérico.

```

C:\Users\Monster\Desktop\SSL-GRUPO\TP2\Ejerc-2.exe
----- | Ingrese un caracter numerico: 5
----- | q0+ | | El caracter 5 es numerico y su valor entero es 5
----- |
----- | v
----- | Ingrese un caracter numerico: 4
----- | q1+ | | El caracter 4 es numerico y su valor entero es 4
----- |
----- | v
----- | Ingrese un caracter numerico: 6
----- | q2+ | | El caracter 6 es numerico y su valor entero es 6
----- |
----- | v
----- | Ingrese un caracter numerico: M
----- | q3- | | El caracter M no es numerico
-----
Process exited after 6.479 seconds with return value 0
Presione una tecla para continuar . . .

```

Captura del programa en ejecución

3) Ingresar una cadena que represente una operación simple con enteros decimales y obtener su resultado, se debe operar con +, -, /, *. Ejemplo = $3+4*8/2+3-5 = 29$. Debe poder operar con cualquier número de operandos y operadores respetando la precedencia de los operadores aritméticos y sin paréntesis. La cadena ingresada debe ser validada previamente preferentemente reutilizando las funciones del ejercicio 1. Para poder realizar la operación los caracteres deben convertirse a números utilizando la función 2. La cadena debe ingresar por línea de comando o por archivo.

Para este ejercicio la idea fundamental es recibir la cadena a operar en notación infija, que es la notación que usamos para resolver las operaciones manualmente, necesitamos pasar a notación postfija o también conocida como notación polaca inversa. Luego procesamos la notación polaca inversa con una pila consumiendo número y operador.

Para procesar primero la cadena en notación infija a postfija, necesitamos usar una pila la cual nos va a ser útil para almacenar los operadores matemáticos.

Fue necesario crear funciones que manipulan la Pila:

```
6 //Codigo para generar la cadena postfija (notacion polaca inver
7
8 // creamos la pila para los operadores
9 ▼ typedef struct opNodo
10 {
11     char info;
12     struct opNodo* sig;
13 }nodoOp;
14
15 typedef nodoOp* ptrNodoOp;
16
17 void pushOp(ptrNodoOp* pila, char info)
18 {
19     ptrNodoOp nuevo=(ptrNodoOp)malloc(sizeof(nodoOp));
20     nuevo->info=info;
21     nuevo->sig=*pila;
22     *pila=nuevo;
23 }
24
25 char popOp(ptrNodoOp* pila)
26 {
27     char ret=(*pila)->info;
28     ptrNodoOp aux=(*pila);
29     *pila=aux->sig;
30     free(aux);
31
32     return ret;
33
34 }
35
36 int estaVacia(ptrNodoOp pila){
37     if(pila == NULL){
38         return 1;
39     }else{
40         return 0;
41     }
42 }
```

Captura de las funciones relacionadas a la pila

La funcion *nivelPrecedencia* reordena las prioridades de los operadores

```
44 //devuelve un int que ese el nivel de precedencia del operador
45 int nivelPrecedencia(char operador){
46     int nivel =0;
47     switch(operador){
48         case '+': nivel =1;
49             break;
50         case '-': nivel =1;
51             break;
52         case '*': nivel =2;
53             break;
54         case '/': nivel =2;
55             break;
56     }
57     return nivel;
58 }
59
```

Captura de la funcion nivelPrecedencia

La funcion *tieneMayorOIgualPrioridad* compara las prioridades entre los operadores

```
60 int tieneMayorOIgualPrioridad(char operador1, char operador2){
61     int precedenciaPrimerOperador=nivelPrecedencia(operador1);
62     int precedenciaSegundoOperador=nivelPrecedencia(operador2);
63     if(precedenciaPrimerOperador>=precedenciaSegundoOperador){
64         return 1;
65     }else{
66         return 0;
67     }
68 }
69
```

Captura de la funcion tieneMayorOIgualPrioridad

Las funciones *esOperador* y *esOperando* identifican los elementos

```
71  int esOperador(char elemento){
72      if(elemento == '+' ||
73         elemento == '-' ||
74         elemento == '*' ||
75         elemento == '/')
76      ){
77          return 1;
78      }else{
79          return 0;
80      }
81  }
82
83  //verifica que los caracteres operando son decimales
84  int esOperando(char elemento){
85      if(elemento >='0' && elemento <='9'){
86          return 1;
87      }else{
88          return 0;
89      }
90  }
91
```

Captura de las funciones esOperador y esOperando

La función *infijaToPostfija* lo que hace es usar un *while* para procesar todos los caracteres de la cadena infija. El segundo *while*, mientras reciba un carácter numérico va colocando cada uno dentro de la cadena postfija, delimitados por el carácter '\$'.

```

91
92
93 //transforma la cadena de la expresion aritmetica que se ingresa en notacion infija, a la notacion postfija (polaca inversa)
94 char* infijaToPostfija(char * infija){
95     char elemento, operador;
96     char *postfija;
97     postfija=malloc(sizeof(char)*100);
98     int j=0;
99     int i=0;
100     ptrNodoOp pilaOp = NULL;
101
102     //creamos el ciclo para procesar la cadena infija
103     int longitud= strlen(infija);
104
105     while (i<longitud){
106         elemento= infija[i];
107         i++;
108         //procesar todos los caracteres numericos de izquierda a derecha hasta encontrar un operador
109         while(esOperando(elemento)){
110             postfija[j]=elemento;
111             j++;
112
113             elemento= infija[i];
114             i++;
115         }
116         postfija[j]= '$'; //delimitador
117         j++;

```

Cuando recibe un operador, se fija si la pila está vacía y lo guarda; si la pila tiene al menos un elemento, compara el nivel de precedencia del operador de la pila con el elemento actual, si el operador de la pila es mayor o igual lo coloca en la cadena postfija, sino guarda el elemento en la pila de los operadores.

```

119         if (esOperador(elemento)){ //generamos pila de operadores
120             if(!estaVacía(pilaOp)){ //check pila diferente de vacío
121                 int seDebeContinuar;
122                 do{
123                     operador=popOp(&pilaOp);
124                     if(tieneMayorOIgualPrioridad(operador, elemento)){
125                         postfija[j]=operador;
126                         j++;
127                         postfija[j] = '$';
128                         j++;
129
130                         seDebeContinuar=1;
131                     }else{
132                         seDebeContinuar=0;
133                         pushOp(&pilaOp,operador);
134                     }
135                 }while(!estaVacía(pilaOp) && seDebeContinuar);
136             }
137             pushOp(&pilaOp,elemento);
138
139
140         }
141     }

```

Cuando ya se consumieron los caracteres de la cadena infija, lo que hace este while es popear todos los operadores restantes de la pila, y los coloca en la cadena postfija, separados por '\$'.

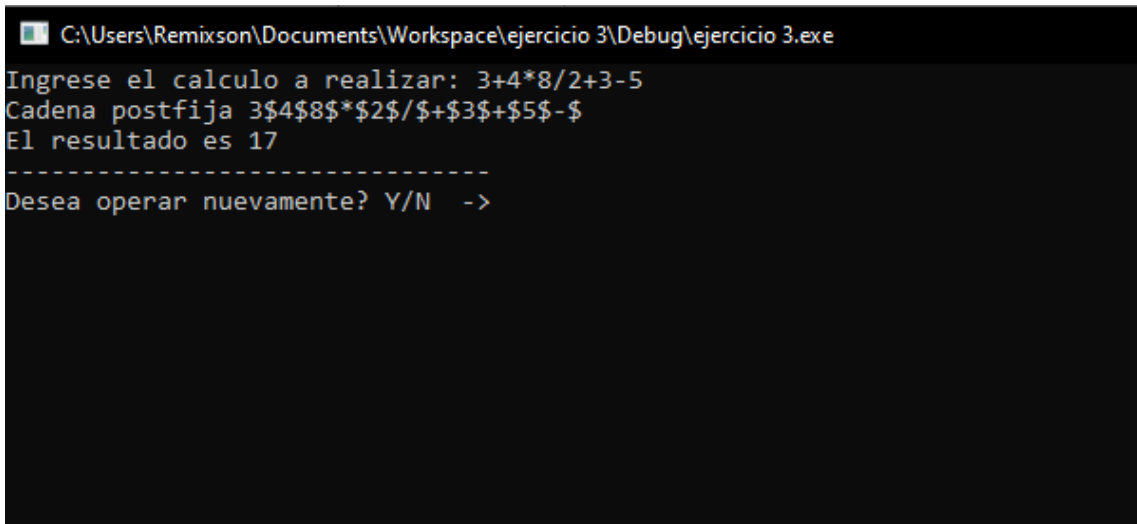
```
142         while(!estaVacia(pilaOp)) {
143             operador=popOp(&pilaOp);
144             postfija[j]=operador;
145             j++;
146             postfija[j] = '$';
147             j++;
148         }
149         postfija[j]='\0';
150     return postfija;
151 }
```

Luego en el main se procesa la cadena postfija generada por la función, para realizar los cálculos matemáticos en esta notación polaca inversa.

Cuando llega un número lo guarda en la pila de números y cuando llega un operador matemático, el switch dictamina la operación a realizar, popeando los elementos de la pila.


```
265
266
267     token = strtok(postfija, delimitador);
268
269     /* para recorrer la cadena con el $ como delimitador */
270     //mientras token diferente de vacio
271     while( token != NULL ) {
272         if(VerificaDecimal(token) && EsDecimal(token)){
273
274             push(&pila, atoi(token)); //transformamos caracter a numero
275             //printf("%s \n", token);
276         }else{
277
278             switch(token[0]){
279                 case '+':
280                     push(&pila, pop(&pila) + pop(&pila));
281                     break;
282                 case '-':
283                     op2 = pop(&pila);
284                     push(&pila, pop(&pila) - op2);
285                     break;
286                 case '/':
287                     op2 = pop(&pila);
288                     if(op2 != 0){
289                         push(&pila, pop(&pila) / op2);
290
291                     }else{
292                         printf("error division por cero \n");
293                     }
294                     break;
295                 case '*':
296                     push(&pila, pop(&pila) * pop(&pila));
297                     break;
298                 default:
299                     printf("Error desconocido");
300                     break;
301             }
302         }
303         token = strtok(NULL, delimitador);
304     }
305
306 }
```

Screenshot del programa en ejecución (se utilizó el caso de la consigna)



```
C:\Users\Remixson\Documents\Workspace\ejercicio 3\Debug\ejercicio 3.exe
Ingrese el calculo a realizar: 3+4*8/2+3-5
Cadena postfija 3$4$8$*$2$/$+$3$+$5$-$
El resultado es 17
-----
Desea operar nuevamente? Y/N ->
```

Conclusión:

En este trabajo práctico, pudimos desarrollar funciones que nos permiten implementar los autómatas, los cuales deben ser completos para que sean fácilmente programables.

Utilizamos los conceptos de pilas aprendidos, para poder procesar por una parte el orden de precedencia de los operadores para la notación polaca inversa, y luego usar otra pila adicional para poder calcular las operaciones correspondientes popeando la pila.