

Instituto Tecnológico y de Estudios
Superiores de Occidente – ITESO



ITESO

Universidad Jesuita
de Guadalajara

Materia: Programación con Memoria Dinámica

Profesor: Luis Gatica

PROYECTO FINAL:

ANALIZADOR SINTÁCTICO DE FÓRMULAS BIEN
FORMADAS

Fecha: 12 de mayo de 2018

Autor(es): Avalos Guzmán Luis Joaquín Avalos

Pérez- Vargas Pinson Carolina

Proyecto Final

Analizador sintáctico de fórmulas bien formadas

Descripción del problema a resolver

Este proyecto tiene como objetivo a partir de una entrada del usuario la cual será una secuencia de caracteres, mejor dicho, una expresión lógica de la cual se tendrá que validar si es una fórmula bien formada, ¿a qué nos referimos con una fórmula bien formada?, para decir que una expresión lógica es una fórmula bien formada se tienen que hacer 2 análisis, uno léxico en el cual se tiene que comprobar que los caracteres en la expresión lógica ingresada por el usuario pertenezcan al lenguaje lógico, por ejemplo que sean letras de la a a la z pero minúsculas, letras mayúsculas no se consideran, si se ingresa una A o B o C ya no sería una fórmula bien formada, otro ejemplo es que la expresión tenga operadores válidos como los siguientes: \mid , $\&$, \sim , \rightarrow , \leftrightarrow y no operadores como $+$, $-$, $*$, $/$ pues el ingreso de uno de ellos la descalificarían como fórmula bien formada. El otro análisis es sintáctico en el cual se deben cumplir por así decirlo la gramática de una fórmula bien formada, por ejemplo, si es un operador binario como \mid o $\&$ a su izquierda y derecha debe estar una letra o una proposición válida, ejemplos: $a \mid b$ o $c \& d$ o $a \& (c \mid b)$; si es un operador unario como \sim sólo debe tener una letra a lado derecho, ejemplo $\sim a$ o $\sim b$, fórmulas mal formadas serían $a \sim$, $\mid b \& a$, $a \& \& \& \& b$.

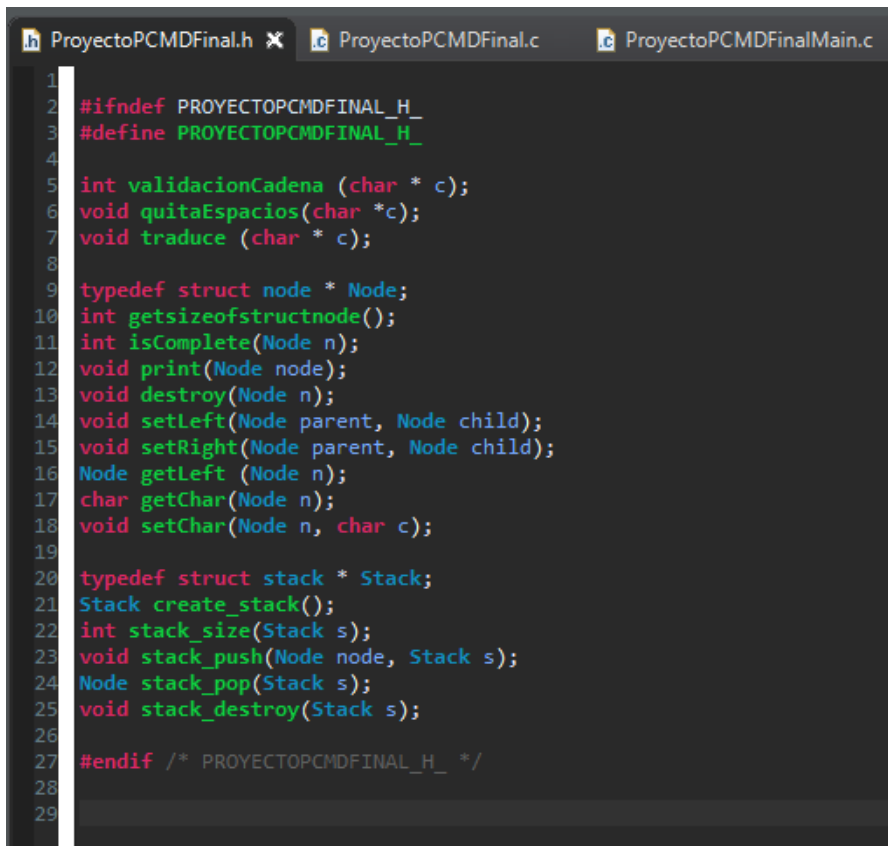
Otra validación de la cadena consiste el ingreso de paréntesis correctos, es decir que los paréntesis ingresados para especificar las preferencias en las expresiones lógicas tengan sentido como $((\sim a) \& b)$ o $((a \& b) \mid (c))$, un requisito expuesto por nosotros en este aspecto es que siempre al ingresarse una expresión lógica deben incluirse los paréntesis más

externos, sino por default se pedirá otra cadena de caracteres, expresión lógica bien formada: (a), no bien formada: a.

Solución

Código fuente:

Interfaz(.h)



```
1
2 #ifndef PROYECTOPCMDFINAL_H_
3 #define PROYECTOPCMDFINAL_H_
4
5 int validacionCadena (char * c);
6 void quitaEspacios(char *c);
7 void traduce (char * c);
8
9 typedef struct node * Node;
10 int getsizeofstructnode();
11 int isComplete(Node n);
12 void print(Node node);
13 void destroy(Node n);
14 void setLeft(Node parent, Node child);
15 void setRight(Node parent, Node child);
16 Node getLeft (Node n);
17 char getChar(Node n);
18 void setChar(Node n, char c);
19
20 typedef struct stack * Stack;
21 Stack create_stack();
22 int stack_size(Stack s);
23 void stack_push(Node node, Stack s);
24 Node stack_pop(Stack s);
25 void stack_destroy(Stack s);
26
27 #endif /* PROYECTOPCMDFINAL_H_ */
28
29
```

Implementación(.c)

```
ProyectoPCMDFinal.h ProyectoPCMDFinal.c ProyectoPCMDFinalMain.c
1 #include "ProyectoPCMDFinal.h"
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5
6 static const char NEG = '~';
7
8 //-----FUNCIONES CADENA-----
9 int validacionCadena (char * c){
10     int l = strlen(c);
11     int numParentesis= 0;
12
13     for(int i = 0; c[i] != '\0'; i++){
14         if(c[i] == '(' || c[i] == ')')
15             numParentesis++;
16     }
17
18     if(numParentesis%2 != 0)
19         return 0;
20
21     if (c[l-1] != ')' || c[0] != '(')
22         return 0;
23
24     for (int i = 0; c[i+1] != '\0'; i++)
25         if (c[i] == '-' && c[i+1] != '>')
26             return 0;
27
28     else if (c[i] == '<' && c[i+1] != '-')
29         return 0;
30 }
```

```
31     for (int i = 0; c[i] != '\0'; i++)
32         if (c[i] >= 'A' && c[i] <= 'Z') {
33             return 0;
34         }
35
36     return 1; //Si llego a este punto, es porque paso todos los check points
37 }
38
39 void quitaEspacios(char *c)
40 {
41     int count = 0; //Esto es el largo de la cadena sin espacios (cuenta cada caracter no-espacio que vas procesando)
42     for (int i = 0; c[i] != '\0'; i++) //Recorre toda la cadena
43         if (c[i] != ' ')
44             c[count++] = c[i]; //Si el caracter actual no es espacio, lo pone en index count y luego incrementa count (post-incremento)
45     c[count] = '\0'; //Agrega fin de cadena
46 }
47
48 void traduce(char * c){
49     int count = 0; //Esto es el largo de la cadena traducida (cuenta cada caracter traducible que vas procesando)
50     for (int i = 0; c[i] != '\0'; i++)//Recorre toda la cadena
51     {
52         if (c[i] == '-')
53         {
54             c[count++] = c[i]; //Si el caracter actual es -, lo pone en index count y luego incrementa count (post-incremento)
55             i = i + 1; //Se traga el >
56 }
```

```

57     else if (c[i] == '<')
58     {
59         c[count++] = c[i]; //Si el caracter actual es <, lo pone en index count y luego incrementa count (post-incremento)
60         i = i + 2; //Se traga el ->
61     }
62     else c[count++] = c[i];
63 }
64 c[count] = '\0'; //Agrega fin de cadena
65 }
66
67
68 //-----FUNCIONES TREE-----
69
70 typedef struct node{
71     char c;
72     struct node * left, *right;
73 }* Node;
74
75 int getsizeofstructnode(){
76     return (sizeof(struct node));
77 }
78

```

```

79 int isComplete(Node n){ //Un nodo es completo si se cumple cualquiera de las condiciones
80     if (n->c <= 'z' && n->c >= 'a')
81         return 1; //Corresponde a una proposición atómica
82     else if ((n->c == '&' || n->c == '|' || n->c == '<' || n->c == '-') && n->left != NULL && n->right != NULL)
83         return 1; //Es binario y tiene ambos hijos
84     else if ((n->c == NEG) && (n->left != NULL && n->right == NULL))
85         return 1; //Es unario y tiene un y sólo un hijo
86     else
87         return 0;
88 }
89
90 void print(Node node){ //RECURSIVA... es llamada con la raíz, imprime la expresión en forma polaca
91     if (node != NULL) //Si el nodo node es nulo, no se hace nada.
92     {
93         print(node->left);
94         print(node->right);
95         if (node->c == '<') printf("<->"); //el data de node
96         else if (node->c == '-') printf("->");
97         else printf("%c", node->c);
98     }
99 }
100
101 void destroy(Node n){ //RECURSIVA... es llamada con la raíz
102     if(n == NULL) return;
103     if(n->left != NULL) destroy(n->left);
104     if(n->right != NULL) destroy(n->right);
105     free(n);
106 }
107

```

```

108 void setLeft(Node parent, Node child){
109     if (parent != NULL)
110         parent->left = child;
111 }
112
113 void setRight(Node parent, Node child){
114     if (parent != NULL)
115         parent->right = child;
116 }
117

```

```

118 Node getLeft(Node n){
119     return n->left;
120 }
121
122 char getChar(Node n){
123     return n->c;
124 }
125
126 void setChar(Node n, char c){
127     n->c = c;
128 }
129
130
131 //-----FUNCIONES STACK-----
132
133 typedef struct stackNode{
134     Node node;
135     struct stackNode * prior;
136 }* StackNode;
137
138 typedef struct stack{
139     StackNode top;
140     int size;
141 }* Stack;
142
143 Stack create_stack(){
144     Stack s = (Stack) malloc (sizeof(struct stack));
145     s->size = 0;
146     s->top = NULL;
147     return s;
148 }
149

```

```

150 int stack_size(Stack s)
151 {
152     if(s == NULL)
153         return -1;
154     return s->size;
155 }
156
157 void stack_push(Node node, Stack s){
158     StackNode n = (StackNode) malloc(sizeof(struct stackNode));
159     if(n == NULL)
160         return;
161     n->node = node;
162     n->prior = s->top;
163     s->top = n;
164     s->size++;
165 }
166
167 Node stack_pop(Stack s)
168 {
169     if(s->size == 0) return NULL;
170     Node top = s->top->node;
171     StackNode temp = s->top;
172     s->top = s->top->prior;
173     s->size--;
174     free(temp);
175     return top;
176 }
177

```

```

178 void stack_destroy(Stack s)
179 {
180     StackNode aux1, aux2;
181     aux1 = s->top;
182
183     while(aux1 != NULL)
184     {
185         aux2 = aux1->prior;
186         free(aux1);
187         aux1 = aux2;
188     }
189     free(s);
190 }
191

```

Main(.c)

```

ProyectoPCMDFinal.h ProyectoPCMDFinal.c ProyectoPCMDFinalMain.c
1 #include<stdio.h>
2 #include<string.h>
3 #include<stdlib.h>
4 #include"ProyectoPCMDFinal.h"
5
6 static const char NEG = '~';
7
8 int main(){
9     setbuf(stdout, NULL);
10    int cadenaValida = 0; //bandera
11    char cadena [100];
12    puts("Bienvenido, por favor escribe una fórmula lógica y te diré si es una fórmula bien formada.");
13    while(cadenaValida==0)
14    {
15        gets(cadena);
16        quitaEspacios(cadena);
17        if(validacionCadena(cadena))
18            cadenaValida = 1;
19        else
20            puts("Fórmula inválida, escribe otra por favor");
21    }
22    traduce(cadena);
23
24    Stack s1 = create_stack();
25    Stack s2 = create_stack();
26
27    int isFbf = 1; //bandera: es fórmula bien formada
28    Node root = NULL;
29    int cursor = 0;
30

```

```

31 do{//MOOD PUSH: leer cadena e ir convirtiendo sus caracteres en nodos que se insertarán en stack1, hasta alcanzar un ')' o el fin de cadena
32
33     for (int i = cursor; cadena[i]!='\0'; i++)
34     {
35         Node n = (Node) malloc(sizeof(structnode)); //Tokenizar: ir convirtiendo la cadena en nodos
36         setChar(n, cadena[i]);
37         setLeft(n, NULL);
38         setRight(n, NULL);
39
40         stack_push(n, s1); //Que se insertarán en la pila A
41         if(cadena[i] == ')')
42         {
43             cursor = i + 1;
44             break;
45         }
46     }
47

```

```

47
48 //MODO POP: hará pop del stack1 y se procesará cada nodo según su tipo y según si está completo o no
49 do{ //Controla modo pop
50     Node N = stack_pop(s1);
51     Node aux0, aux1, aux2;
52
53     if (getChar(N) == ')') || (getChar(N) <= 'z' && getChar(N) >= 'a') || isComplete(N)) //Es un CIERRE DE PARÉNTESIS/PROP ATÓMICA
54         stack_push(N, s2); //Insertar nodo en la pila B.
55
56     else if (getChar(N) == NEG) //Es UNARIO
57     {
58         Node aux1 = stack_pop(s2); //Extraer un nodo de la pila B, que deberá ser un nodo completo.
59         if (!isComplete(aux1))
60         {
61             destroy(aux1);
62             free(N);
63             isFbf = 0;
64         }
65         else
66         {
67             setLeft(N, aux1); //Y convertirlo en el único hijo de N.
68             stack_push(N, s2); //Ahora que N es un nodo completo, insertarlo en la pila B.
69         }
70     }
71

```

```

72     else if (getChar(N) == '&' || getChar(N) == '|' || getChar(N) == '-' || getChar(N) == '<') //Es BINARIO
73     {
74         aux1 = stack_pop(s1); //Extraer un nodo de la pila A
75
76
77         if (!isComplete(aux1) && (getChar(aux1) == '(' || getChar(aux1) == ')') || getChar(aux1) == NEG || getChar(aux1) == NEG ))
78         {
79             destroy(aux1);
80             free(N);
81             isFbf = 0;
82         }
83         else{
84             setLeft(N, aux1); //Y convertirlo en el hijo izquierdo de N
85             aux2 = stack_pop(s2); //Extraer un nodo de la pila B
86
87             if (!isComplete(aux2) && (getChar(aux2) == '(' || getChar(aux2) == ')') || getChar(aux1) == NEG || getChar(aux1) == NEG))
88             {
89                 destroy(aux2);
90                 free(N);
91                 isFbf = 0;
92             }
93             else{
94                 setRight(N, aux2); //Y convertirlo en el hijo derecho de N.
95                 //N está completo y es la expresion binaria
96                 aux2 = stack_pop(s2); //Sacar un nodo de la pila B
97                 if (getChar(aux2) != ')')
98                     isFbf = 0;
99

```



```

100     aux1 = stack_pop(s1); //Sacar otro nodo de la pila A
101     if (getChar(aux1) != '(' && getChar(aux1) != NEG)
102     {
103         destroy(aux2);
104         destroy(aux1);
105         destroy(N);
106         isFbf = 0;
107     }
108     else if (getChar(aux1) == NEG) //Caso especial
109     {
110         setLeft (aux1, getLeft(N));
111         setLeft(N, aux1);
112         aux0 = stack_pop(s1); //Sacar otro nodo de la pila B que sera el paréntesis abierto
113         free(aux0); //Elimina (apertura),
114         free (aux2); //Elimina (cierre)
115         stack_push(N, s1);
116     }
117     else //Caso normal
118     {
119         free(aux1); //Eliminar apertura
120         free (aux2); //Eliminar cierre
121         stack_push(N, s1);
122     }
123 }
124 }
125
126 } //Termina caso binario
127

```

```

128
129     else //es una APERTURA DE PARÉNTESIS
130     {
131         aux1 = stack_pop(s2); //sacar un nodo de la pila B
132         if (!isComplete(aux1))
133         {
134             destroy(aux1);
135             free(N);
136             isFbf = 0;
137         }
138         aux2 = stack_pop(s2); //Sacar otro nodo de la pila B
139         if (getChar(aux2) != ')')
140             isFbf = 0;
141         free(N); //Eliminar (apertura),
142         free (aux2); //Eliminar (cierre)
143         stack_push(aux1, s1);
144     }
145
146     if(stack_size(s1) == 0) //Si en el modo POP la pila A se queda vacía
147     {
148         if (stack_size(s2) != 1)
149             isFbf = 0; //Si no es sólo 1 nodo el que quedará en B, entonces la fórmula no es una fbf
150         else
151         {
152             N = stack_pop(s2); //La pila B debe tener un solo nodo (completo), que se extraerá de ahí.
153             if (!isComplete(N))
154             {
155                 destroy(N);
156                 isFbf = 0;
157             }
158         }
159     }
160

```

```

158         else
159             root = N; //Ya ambas pilas quedan vacías y ese último nodo de la pila B es la raíz del árbol
160     }
161 }
162 }while (stack_size (s2) != 0 && isFbf); //Cambiamos al modo PUSH cuando la pila B vuelva a estar vacía.
163 }while (stack_size (s1) != 0 && isFbf); //Saldremos de ambos modos cuando queden vacías ambas pilas
164
165 //-----
166
167 if (isFbf)
168 {
169     puts("Felicitades, es una es fórmula bien formada.");
170     puts("Esta es su representación en notación polaca inversa:");
171     print(root);
172 }
173
174 else puts("Lo siento, NO es una es fórmula bien formada.");
175
176 stack_destroy(s1);
177 stack_destroy(s2);
178 destroy(root); //Destruye el árbol
179 return 0;
180 }
181

```

Ejecución:

```

Tasks Properties Problems Console X
<terminated> (exit value: 0) ProyectoPCMDFinal.exe [C/C++ Application] C:\Users\Joaquin\eclipse-workspace\ProyectoPCMD
Bienvenido, por favor escribe una fórmula lógica y te diré si es una fórmula bien formada.
~( ( p <-> p ) <-> p )
Felicitades, es una es fórmula bien formada.
Esta es su representación en notación polaca inversa:
pp<->p<->~

```

```

Tasks Properties Problems Console X
<terminated> (exit value: 0) ProyectoPCMDFinal.exe [C/C++ Application] C:\Users\Joaquin\eclipse-workspace\ProyectoPCMD
Bienvenido, por favor escribe una fórmula lógica y te diré si es una fórmula bien formada.
a
Fórmula inválida, escribe otra por favor
a&b
Fórmula inválida, escribe otra por favor
(a&->b)
Lo siento, NO es una es fórmula bien formada.

```

```

Tasks Properties Problems Console X
<terminated> (exit value: 0) ProyectoPCMDFinal.exe [C/C++ Application] C:\Users\Joaquin\eclipse-workspace\ProyectoPCMD
Bienvenido, por favor escribe una fórmula lógica y te diré si es una fórmula bien formada.
(((p | q) & (r | s)) -> ~p) )
Felicitades, es una es fórmula bien formada.
Esta es su representación en notación polaca inversa:
pq|rs|&p~->

```

```
Tasks Properties Problems Console X
<terminated> (exit value: 0) ProyectoPCMDFinal.exe [C/C++ Application] C:\Users\Joaquin\eclipse-workspace\ProyectoPCM
Bienvenido, por favor escribe una fórmula lógica y te diré si es una fórmula bien formada.
(a->)
Lo siento, NO es una es fórmula bien formada.
```

```
Tasks Properties Problems Console X
<terminated> (exit value: 0) ProyectoPCMDFinal.exe [C/C++ Application] C:\Users\Joaquin\eclipse-workspace\ProyectoPCM
Bienvenido, por favor escribe una fórmula lógica y te diré si es una fórmula bien formada.
(((p -> (q & r)) -> (~(~p) & q)) )
Felicidades, es una es fórmula bien formada.
Esta es su representación en notación polaca inversa:
pqr&->p~~q&->
```

```
Tasks Properties Problems Console X
<terminated> (exit value: 0) ProyectoPCMDFinal.exe [C/C++ Application] C:\Users\Joaquin\eclipse-workspace\ProyectoPCM
Bienvenido, por favor escribe una fórmula lógica y te diré si es una fórmula bien formada.
a->A
Fórmula inválida, escribe otra por favor
a
Fórmula inválida, escribe otra por favor
(((p -> q) | r) -> (p & ~p))
Felicidades, es una es fórmula bien formada.
Esta es su representación en notación polaca inversa:
pq->r|pp~&->
```

Conclusión

El proyecto presentó un reto, fue una buena manera para darnos cuenta del avance que hemos dado en cuestiones de programación desde que entramos a la carrera, por qué lo definimos como un reto y la manera en que hemos visto nuestros avances, eso se debe a que entender el algoritmo del analizador sintáctico de fórmulas bien formadas fue hasta cierto punto sencillo gracias a las corridas de escritorio entendimos mejor como funciona y la manera de codificarlo se facilitó, el reto se presentó en excepciones que el algoritmo no mencionaba como cuando en un operador binario como | o & en una de sus preposiciones ya sea la de la derecha o la de la izquierda había una negación, eso se volvió un sub caso especial a considerar cuando se hacían los cambios de modos y la extracción de los nodos en las pilas.

Otra dificultad fue el operador de negación \neg que debido a su código ASCII negativo causaba problemas de codificación, lo resolvimos utilizando el símbolo \sim como negación y la última dificultad que tuvimos fue que aunque se ingresarán caracteres válidos para las fórmulas lógicas como letras de la a la z minúsculas, la cantidad necesaria de paréntesis y operadores $\rightarrow, \leftrightarrow, |, \&$ y \sim evaluar que fueran correctas tomando en cuenta otros subcasos específicos de los casos generales que nos exponía el algoritmo del documento porque si sólo nos fiábamos en seguir al pie de la letra el algoritmo sin pensar en casos diferentes, el programa tomaba como válida expresiones como $\sim|z$, todo ello lo resolvimos a través de profundas reflexiones, corridas de escritorio y asesorándonos para encontrar los errores en el código que no lográbamos detectar.

Lo más importante que aprendimos al terminar este proyecto además del darnos cuenta de los grandes avances y habilidades que hemos obtenido gracias a este curso fue de herramientas para buscar errores en los códigos y depurarlos como los breakpoints.

