Universidad de Málaga
Departamento de Lenguajes y Ciencias de la Computación
Campus de Teatinos, 29071 MÁLAGA

Examen de Traductores, Intérpretes y Compiladores.
Convocatoria extraordinaria de Septiembre de 2006
3er Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: ______________________

Calificación: ____________

TEÓRICO

1.- Construir la expresión regular en Lex equivalente a la siguiente gramática de Yacc:

```
expr    :    'x'
        |    '1'
        |    expr '+' expr
        |    expr '-' expr
        |    expr '*' expr
        |    expr '/' expr
        ;
```

donde los únicos terminales adminisbles son: T={x, 1, +, -, *, /}.

2.- Construir una gramática en Yacc equivalente a la siguiente expresión regular en Lex:
[a-c] (("_"[0-9]) | ([a-c]*))*

3.- Dada la gramática:

```
①prog´:      prog
             ;
② prog  :    ε
③       |    prog DEF ID '=' NUM
④       |    prog COD expr
             ;
⑤ expr:      TXT '>' ID
⑥       |    expr '>' ID
             ;
```

cuya tabla LALR(1)es la de la figura, se pide reconocer o rechazar la secuencia:
DEF ID = NUM COD TXT > ID $
indicando todos los pasos seguidos.

Tabla LALR(1)

Tabla Acción

	DEF	ID	NUM	COD	TXT	=	>	$
0	r 2			r 2				r 2
1	d 2			d 3				Aceptar
2		d 4						
3					d 6			
4						d 7		
5	r 4			r 4			d 8	r 4
6							d 9	
7			d 10					
8		d 11						
9		d 12						
10	r 3			r 3				r 3
11	r 6			r 6			r 6	r 6
12	r 5			r 5			r 5	r 5

Tabla Ir A

	prog	expr
0	1	
1		
2		
3		5
4		
5		
6		
7		
8		
9		
10		
11		
12		

Número de Terminales:8
Número de No Terminales:2
Numero Celdas en Conflicto en la Tabla "ACCION":0

4.- Responder a las siguientes cuestiones:
a) ¿Para qué sirve un registro de activación?
b) ¿Qué tipos de recuperación de errores se pueden aplicar en un análisis sintáctico?
c) Un análisis sintáctico descendente con retroceso, ¿puede usarse para aceptar o rechazar cadenas de una gramática con reglas épsilon?
d) ¿Para qué sirven los tipos de datos (enteros, caracteres, etc.) desde el punto de vista del constructor de compiladores?

Open Course Ware
Gálvez Rojas, S.
(2011). Traductores, Compiladores e Intérpretes
OCW- Universidad de Málaga http://ocw.uma.es
Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike
cc BY NC SA

Universidad de Málaga
Departamento de Lenguajes y Ciencias de la Computación
Campus de Teatinos, 29071 MÁLAGA

Examen de Traductores, Intérpretes y Compiladores.
Convocatoria extraordinaria de Septiembre de 2006
3er Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: ______________________
Calificación: ____________

PRÁCTICO

Se desea construir el sistema de codificación **Rodas I** que permite cifrar cada uno de los caracteres de un mensaje de entrada.

Para ello, **Rodas I** soporta la especificación por el usuario de cuantas expresiones de codificación se necesiten, a cada una de los cuales se le da un nombre. Posteriormente, un mensaje puede someterse a una secuencia cualquiera de expresiones de codificación.

Así pues, una expresión de codificación puede ser:

`DEFINE suma1 = x[0]+1;`

que quiere decir que al código ASCII de cada carácter se le va a sumar 1. El término `x[0]` hace referencia a cada uno de los caracteres que se van procesando. Así se puede codificar de la forma:

`CODIFICA "HOLA MUCHACHOS" > suma1;` que daría lugar a:

`IPMB!NVDIBDIPT`

Así, para codificar una cadena de caracteres, el algoritmo procesa cada carácter individualmente según la expresión de codificación que se indique (`suma1` en este caso). Cada vez que ésta se aplica a un carácter, a éste se le llama `x[0]`. También es posible hacer referencia a los caracteres anteriores y posteriores con `x[+1]`, `x[+2]`, etc. y `x[-1]`, `x[-2]`, etc. Así, la expresión:

`DEFINE antpos = x[0]-x[-1]+x[+1];`

codificaría el mensaje:

`CODIFICA "LOLO PROMETE" > antpos;` como:

`øOL#!éQJCLT±ç€`

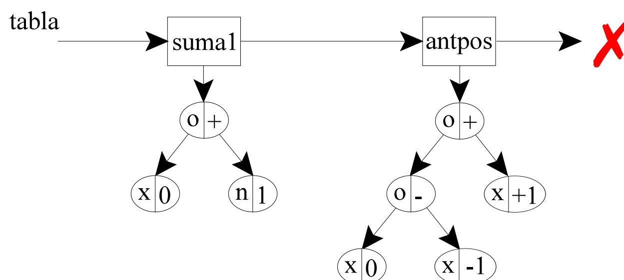
Nota: Véase como, por ejemplo, la primera “O” de “LOLO” y la segunda “L” también de “LOLO”, se quedan como estaban después de la codificación, pues ambas tienen la misma letra delante y detrás en el texto fuente a codificar.

Por último, también se permite hacer codificaciones en secuencia. Así, si se desea someter un mensaje a varias codificaciones puede hacerse de la forma:

`CODIFICA "LOLO PROMETE" > suma1 > antpos;` produciría:

`ØPM$"âRKDMU²Ð`

Realizar todo esto es, conceptualmente, sencillo. Cada vez que se crea una expresión de codificación, su árbol se almacena en una tabla de símbolos, de forma que para las dos expresiones de los ejemplos propuestos, esta tabla de símbolos quedaría:



Gálvez Rojas, S.
(2011). Traductores, Compiladores e Intérpretes
OCW- Universidad de Málaga http://ocw.uma.es
Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

Cuando se desea codificar un texto, cada uno de los caracteres de dicho texto se somete al árbol de codificación que se especifique mediante el nombre de la expresión, recordando que `x[0]` se refiere al carácter que en ese momento se está codificando, `x[-1]` es el carácter anterior, `x[+1]` es el posterior, etc. (siempre en el mensaje sin codificar). Si dichos caracteres no existen porque se esté codificando el primer, último carácter, entonces se deben asumir como caracteres nulos o ceros.

Por último, la siguiente pantalla muestra un ejemplo de funcionamiento:

```
Símbolo del sistema

K:\MisDocumentos\Sergio\Compilad\Septiembre 06>type entrada.txt
DEFINE suma1 = x[0]+1;
DEFINE mult2 = x[0]*2;
CODIFICA "HOLA MUCHACHOS" > suma1;
CODIFICA "HOLA MUCHACHOS" > suma1 > suma1;
DEFINE ant = x[0]+x[-1];
DEFINE pos = x[0]-x[-1];
CODIFICA "HOLA MUCHACHOS" > ant;
CODIFICA "HOLA MUCHACHOS" > pos;
CODIFICA "HOLA MUCHACHOS" > ant > pos;
DEFINE antpos = x[0]-x[-1]+x[+1];
CODIFICA "LOLO PROMETE" > antpos;
CODIFICA "LOLO PROMETE" > suma1 > antpos;

K:\MisDocumentos\Sergio\Compilad\Septiembre 06>exsep06y < entrada.txt
IPMB!NUDIBDIPT
JQNC"OWEJCEJQU
Hù¢ìamóÿïëäïùó
H²J▀€♣·☻♣♦
HO♦≥╘♀5÷≤▌√♀♂
¢OL#!éQJCLT±♀♂
¥PM$"âRKDMU²╤

K:\MISDOC~1\Sergio\Compilad\SEPTIE~4>_
```

Se pide:

- Codificar los programas Lex/Yacc que consiguen la funcionalidad pedida. Para ello se suministra un esqueleto y un fichero en C que contiene la tabla de símbolos. El programa resultante debe suministrar mensajes de error significativos, y no limitarse a un "*syntax error*".
- Crear la función **evaluar(...)** Que permite codificar cada carácter de manera independiente.
- Para cada función de codificación, ¿es posible siempre hacer una función de descodificación que obtenga el mensaje original?¿Por qué?

Fichero tssep06.c

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

typedef struct _nodo {
	char tipo; // 'n'umero, 'x'-posicion, 'o'perador
	union {
		int numero;
		int posicion;
		struct {
			char tipo; // +, -, *, /, %(módulo)
			struct _nodo * izq;
			struct _nodo * der;
		} operador;
	} contenido;
} nodo;

typedef struct _funcion {
	char nombre[21];
	struct _funcion * sig;
	struct _nodo * expresion;
} funcion;

funcion * tabla = NULL;
```


Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike
cc BY NC SA

```
funcion * buscarFuncion(char * nombre){
  funcion * t = tabla;
  while( (t != NULL) && (strcmp(nombre, t->nombre)) )
      t = t->sig;
  return (t);
}

void insertarFuncion(char * nombre, nodo * expresion){
      if (buscarFuncion(nombre) != NULL)
            printf("La funcion %s ya esta declarada.\n", nombre);
      else {
            funcion * aux = (funcion *)malloc(sizeof(funcion));
            strcpy(aux->nombre, nombre);
            aux->sig = tabla;
            tabla = aux;
            aux->expresion = expresion;
      }
}
```

Fichero exsep06l.lex

Fichero exsep06y.yac

```
%{
#include "tssep06.c"
#define LON_MAX 250
      char evaluar(char * texto, int posicion, nodo * ptrNodo);
      char * codificar(char * texto, char * nombreFuncion){
            funcion * ptrFuncion;
            nodo * ptrNodo;
            int longitud, i;
            char res[LON_MAX];
            res[strlen(texto)]=0;
            if ((ptrFuncion=buscarFuncion(nombreFuncion)) == NULL)
                  printf("La funcion %s no existe.\n", nombreFuncion);
            else {
                  ptrNodo = ptrFuncion->expresion;
                  longitud = strlen(texto);
                  for(i=0; i<longitud; i++)
                        res[i]=evaluar(texto, i, ptrNodo);
            }
```


Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

```
		return res;
	}
	char evaluar(char * texto, int posicion, nodo * ptrNodo){















	}
%}
%union{
	char nombre[25];
	char texto[LON_MAX];
	int numero;
	nodo * ptrNodo;
}








%%
prog	:	/* Epsilon */
		|	prog DEFINE ID '=' expr ';'	{

				}
		|	prog CODIFICA codi ';'	{

				}
		|	prog error ';' { yyerrok; }
		;
codi	:	TEXTO '>' ID		{

					}
		|	codi '>' ID		{
```


Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

```
                                    }
                ;
expr            :       NUM     {
                                        $$ = (nodo *) malloc(sizeof(nodo));
                                        $$->tipo = 'n';
                                        $$->contenido.numero = $1;
                                }
                |       POSICION        {
                                        $$ = (nodo *) malloc(sizeof(nodo));
                                        $$->tipo = 'x';
                                        $$->contenido.posicion = $1;
                                }
                |       expr '+' expr   {
                                        $$ = (nodo *) malloc(sizeof(nodo));
                                        $$->tipo = 'o';
                                        $$->contenido.operador.tipo = '+';
                                        $$->contenido.operador.izq = $1;
                                        $$->contenido.operador.der = $3;
                                }
                |       expr '-' expr   {
                                        $$ = (nodo *) malloc(sizeof(nodo));
                                        $$->tipo = 'o';
                                        $$->contenido.operador.tipo = '-';
                                        $$->contenido.operador.izq = $1;
                                        $$->contenido.operador.der = $3;
                                }
                |       expr '*' expr   {
                                        $$ = (nodo *) malloc(sizeof(nodo));
                                        $$->tipo = 'o';
                                        $$->contenido.operador.tipo = '*';
                                        $$->contenido.operador.izq = $1;
                                        $$->contenido.operador.der = $3;
                                }
                |       expr '/' expr   {
                                        $$ = (nodo *) malloc(sizeof(nodo));
                                        $$->tipo = 'o';
                                        $$->contenido.operador.tipo = '/';
                                        $$->contenido.operador.izq = $1;
                                        $$->contenido.operador.der = $3;
                                }
                |       expr '%' expr   {
                                        $$ = (nodo *) malloc(sizeof(nodo));
                                        $$->tipo = 'o';
                                        $$->contenido.operador.tipo = '%';
                                        $$->contenido.operador.izq = $1;
                                        $$->contenido.operador.der = $3;
                                }
        |       '(' expr ')'    {
                                        $$ = $2;
                                }
        ;
%%
```

Open Course Ware

Gálvez Rojas, S.
(2011). Traductores, Compiladores e Intérpretes

OCW- Universidad de Málaga http://ocw.uma.es
Bajo licencia Creative Commons Attribution-Non-Comercial-ShareAlike

cc BY NC SA