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 Diciembre de 2006
3er Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: _____

Calificación: _____

TEORÍA

1.- Sea la gramática:

S → ε | P f C n S | C h P n S
P → n f P | f
C → g C | h
%token f n h g

cuya tabla de reconocimiento LL(1) es:

N \ T	**f**	**n**	**h**	**g**	**$**
S	S → P f C n S	S → P f C n S	S → C h P n S	S → C h P n S	S → ε
P	P → f	P → n f P			
C			C → h	C → g C	

se pide:

a) reconocer o rechazar la cadena:
n f n f f f g g h n f f h n $
b) Construir un único diagrama de Conway para reconocer el no terminal S.

2.- A partir del diagrama de Conway construído en el paso anterior, codificar en C o en pseudocódigo, una función que reconozca el no terminal S.

3.- Dada la expresión regular PCLex:

[^a-k]{3, 5}[qr]"77"

indicar cuáles de las siguientes cadenas se reconocen y cuáles se rechazan (responder equivocadamente una opción resta puntos; no responder una opción ni suma ni resta puntos):

a) rrrrq77
b) qqqqpqr77
c) kkkqr77
d) qrrq77
e) qqqqrq77

4.- La gramática:

S → S m A | m | m m
A → m a

¿tiene algún problema a la hora de ser reconocida por PCYacc? Y si es así, ¿es posible solucionarlo?

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 Diciembre de 2006
3er Curso de I.T. Informática de Sistemas.

Apellidos, Nombre: _____

Calificación: _____

Se pretende construir el lenguaje Imago cuyo objetivo es manipular imágenes en dos dimensiones almacenadas de tal manera que cada pixel se representa mediante un *byte* que representa un tono de gris que va del blanco (representado por el 0) al negro (representado por el 255). Para simplificar el trabajo, consideraremos que las imágenes, en lugar de ser en dos dimensiones, constarán de una única dimensión (array simple de *bytes*).

Las operaciones que pueden hacerse con una imagen son las siguientes:
- Crear una imagen a partir de una secuencia de pixeles. Cada valor de un píxel va del 0 al 255. Ej.:

```
RAW 123,45,65,65,45,80
```

- Cargar una imagen (una secuencia de *bytes*) en una variable. Ej.:

```
LOAD img1= RAW 123,45,65,65,45,80;
```

- Visualizar el contenido de una imagen (ya venga dada por un identificador o por una imagen en forma de expresión):

```
VIEW img1;
```

produciría:

```
123-45-65-65-45-80-
```

- Crear una imagen con una trama, dada por otra imagen. Por ejemplo, si se tiene la trama de imagen 23,45,65 y se desea cargar una imagen de longitud 20:

```
VIEW 20 ENTRAMADO_CON 23,45,65;
```

produciría la imagen:

```
23-45-65-23-45-65-23-45-65-23-45-65-23-45-65-23-45-65-23-45-
```

- Cambiar el brillo en un procentaje **pct** real que va del -1 al +1, donde el valor 0 deja a la imagen igual. El valor final de un *byte* **b** se calcula de la forma b+b*pct. Si este valor supera 255 entonces se queda en 255. Por ejemplo:

```
VIEW img1 WITH_BRIGHT 0.3;
```

produciría:

```
159-58-84-84-58-104-
```

- Fusionar dos imágenes. Si las imágenes tienen distinta longitud, la más corta se escala para que sea de igual longitud a la más grande. La fusión puede ser mediante suma o mediante resta. Por ejemplo:

```
LOAD imgxx= RAW 12,23,123,34,5;
LOAD imgyy= RAW 12,4;
VIEW imgyy + imgxx;
```

produciría (si un valor supera 255 se pone 255):

```
24-35-135-38-9-
```

ya que imgyy posee sólo dos elementos y se expandiría hasta llegar a tener los mismos que el

otro operador (imgxx), que tiene 5. Así 12,4 pasaría a 12,12,12,4,4.

Y la resta daría (si un valor es negativo se pone 0):

VIEW imgxx - imgyy;

0-11-111-30-1-

- Obtener el negativo de una imagen; cada *byte* **b** se sustituye por 255-**b**. Por ejemplo:

NEG img1

produciría

132-210-190-190-210-175-

Así pues, la gramática quedaría como:

```
prog    :               /* Epsilon */
                |       prog LOAD ID '=' expr ';'
                |       prog VIEW expr ';'
                |       prog error ';'
                ;
expr    :       ID
                |       RAW lista_num
                |       NUM ENTRAMADO_CON expr
                |       expr WITH_BRIGHT NUMREAL
                |       NEG expr
                |       expr '+' expr
                |       expr '-' expr
                |       '(' expr ')'
                ;
lista_num       :       num0_255
                |       lista_num ',' num0_255
                ;
num0_255        :       NUM
                ;
```

Los errores que se desean controlar son:

- Si una variable se usa sin haber sido previamente asignada, se produce un error.
- La longitud de una imagen en el operador de ENTRAMADO no puede ser 0.
- El número real que especifica el brillo debe estar entre -1 y +1.
- El valor que compone cada píxel debe estar entre 0 y 255.
- El tamaño máximo de una imagen no puede superar los 1024 píxeles.

Se pide:

- Construir los programas Lex y Yacc que proporcionen la funcionalidades pedidas en Imago.

Para ello se proporcionan los siguientes esqueletos:

Fichero TabSimb.c

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

typedef struct{
      int longitud;
      unsigned char *datos;
} imagen;
```

```
typedef struct _simbolo {
      struct _simbolo * sig;
    char nombre[20];
    imagen valor;
} simbolo;

simbolo * crearTabla() {
  return NULL;
};

simbolo * insertarSimbolo(simbolo * * p_t, char nombre[20]) {
      simbolo * s = (simbolo *)malloc(sizeof(simbolo));
      strcpy(s->nombre, nombre);
      s->valor.longitud=0;
      s->sig = (*p_t);
      return (*p_t) = s;

}

simbolo * buscarSimbolo(simbolo * t, char nombre[20]) {
  while( (t != NULL) && (strcmp(nombre, t->nombre)) )
      t = t->sig;
  return (t);
}

void verTabla(simbolo * t){
      while(t != NULL){
            int i;
            printf("%s:\n", t->nombre);
            for (i=0; i<t->valor.longitud; i++)
                  printf("%d-", t->valor.datos[i]);
            printf("\n");
            t = t->sig;
      }
}
```

Fichero exdic06l.lex

```
%%
                                                    ]; }
```

Fichero exdic06y.yac

```
%{
#include "TabSimb.c"
#define MAX_SIZE 1024

   int longitudActual=0;
   unsigned char listaBytes[MAX_SIZE];
   simbolo * tabla;

   void normaliza(imagen *img, int longFinal){
       int i;
       float razon = (float)(*img).longitud / (float)longFinal;
       unsigned char * destino = (unsigned char *) malloc(longFinal);
       for(i=0; longFinal > i; i++){
          destino[i] = (*img).datos[(int)(((float)i)*razon)];
       }
       free((*img).datos);
       (*img).datos = destino;
       (*img).longitud = longFinal;
   }

%}

%union{
```

```
}
```

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

```
                }
      |  prog VIEW expr ';'       {
```

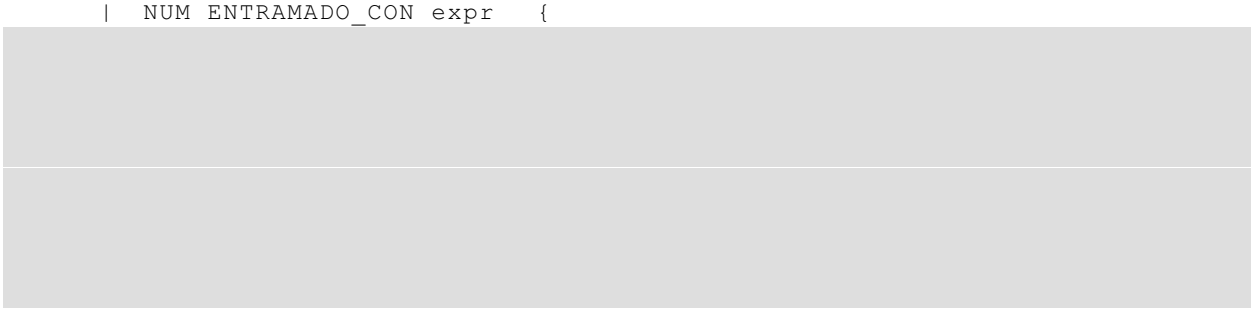
```
              }
      |  prog error ';' { yyerrok; }
      ;
expr  :     ID        {
```



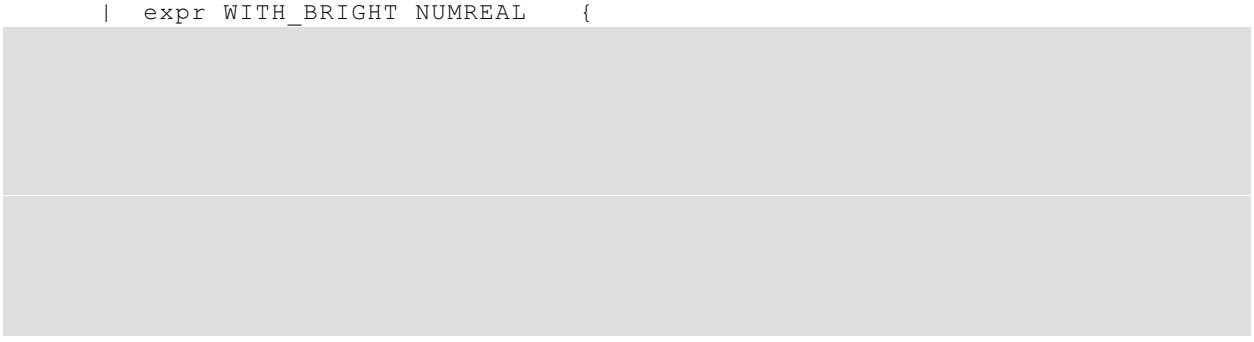
```
                      }
      |  RAW lista_num   {
```



```
                      }
      |  NUM ENTRAMADO_CON expr   {
```



```
                                 }
      |  expr WITH_BRIGHT NUMREAL    {
```



```
                      }
      |  NEG expr        {
```

```
                        }
      |  expr '+' expr   {
                        }
      |  expr '-' expr   {
                        }
      |  '(' expr ')'       {
                        }
      ;
lista_num   :  num0_255     {
                        }
            |  lista_num ',' num0_255  {
                        }
            ;
num0_255    :  NUM    {
                    }
            ;
%%
#include "errorlib.c"
#include "exdic06l.c"
void main(){
   yylineno=1;
   tabla = crearTabla();
   yyparse();
```

```
}
```