Clase #22 de 25

Tipo de yytext, Heap, y Scanners con Lex

Octubre 29, Lunes

Tipo de Dato de yytext y Reserva de Memoria

YYLMAX, %pointer, %array

Ejemplo
Línea más Larga [K&R1988] 1.5.4

```
#include <stdio.h>

int GetLine(char line[], int maxline);
void copy(ch

// print the
int main(){
   const int
   int len;
   int max;
   char line
   char long

   max = 0;
   while ((l
      if (len
         max =
         copy(long
      }
   if(max>0) // t
      printf("%s"
}
```

```
// getline: read a line into s, return length
int GetLine(char s[], int lim){
   int c, i;

   for(i=0; i < lim-1 && (c=getchar())!=EOF && c!='\n'; ++i)
      s[i] = c;
   if (c == '\n') {
      s[i] = c;
      ++i;
   }
   s[i] = '\0';
   return i;
}
```

```
// copy: copy 'from' into 'to'; assume to is big enough
void copy(char to[], char from[]){
   for(int i=0; (to[i] = from[i]) != '\0'; ++i)
      ;
}
```

%pointer, ^, $ y malloc, free, realloc & memcpy

```
%option noyywrap
%{
#include <stdio.h>  // puts
#include <stdlib.h> // free malloc exit

int max;            // maximum length seen so far
char *longest;      // longest line saved here
void copy(char **to, const char *from);
%}
%%
^.*$  if (yyleng > max){
         max = yyleng;
         copy(&longest, yytext); // Makes a copy of the object pointed by longest;
                                 // sets longest to point to that new object.
      }
\n    ;
%%
int main(void){
   yylex();
   if(max>0) // there was a line
      puts(longest);
}

/* Copies 'from' into 'to'; gets storage from heap.
If not enough spaces, exits.*/
void copy(char **to, const char *from){
   free(*to);
   if( NULL == (*to=malloc(yyleng+1)) )
      exit(1);
   for(int i=0; ((*to)[i] = from[i]) != '\0'; ++i)
      ;
}
```

- En nuestro caso, ¿hubiese sido realmente más perfomante realloc por sobre malloc y free? Justificar.

%array, YYLMAX

```
%option noyywrap
%array
%{
#include <stdio.h>    // puts

int max;              // maximum length seen so far
char longest[YYLMAX]; // longest line saved here
void copy(char *to, const char *from);
%}
%%
^.*$ if (yyleng > max){
       max = yyleng;
       copy(longest, yytext);
     }
\n   ;
%%
int main(void){
   yylex();
   if(max>0) // there was a line
     puts(longest);
}

// Copies 'from' into 'to'; assumes to is big enough
void copy(char *to, const char *from){
   for(int i=0; (to[i] = from[i]) != '\0'; ++i)
     ;
}
```

%pointer, RAII

```
%option noyywrap
%{
#include <stdio.h>  // puts
#include <stdlib.h> // free malloc exit

int max;            // maximum length seen so far
char *longest;      // longest line saved here
void copy(char *to, const char *from);
%}
%%
^.*$ if (yyleng > max){
        free(longest);
        if( NULL == (longest=malloc(yyleng+1)) )
           exit(1);
        max = yyleng;
        copy(longest, yytext);
     }
\n   ;
%%
int main(void){
   yylex();
   if(max>0) // there was a line
      puts(longest);
}

// Copies 'from' into 'to'; assumes to is big enough
void copy(char *to, const char *from){
   for(int i=0; (to[i] = from[i]) != '\0'; ++i)
      ;
}
```

Gestión de Memoria Heap

malloc y free

Reserva Manual y Explícita de Memoria desde el Heap, en Tiempo de Ejecución

- Heap
  - Tiempo de vida independiente del alcance
  - Reserva y liberación explícita y manual
- Tipos
  - void *
    - Casting
    - char *
  - size_t
- Expresiones
  - sizeof expresión
  - sizeof(tipo)

```
stdlib.h
void *malloc( size_t size );
Void *calloc( size_t num, size_t size );
void *realloc( void *ptr, size_t newsize );
void free( void *ptr );
```

Scanners con Lex

Proto Scanner de Enteros

Basado en ejemplo 52 [MUCH2012] V1, págs 95-96

```
#include <stdio.h> // getchar EOF printf
#include <stdlib.h> // atoi
#include <ctype.h> // isdigit

const int MAX_LEXEME_LENGTH = 1000;
char lexeme[MAX_LEXEME_LENGTH], *p;
void OutsideNumber(int);
void InsideNumber(int);

int main(void){
   OutsideNumber(getch
}

void OutsideNumber(int
   if(EOF == c)
      return;
   if(isdigit(c)){
      p = lexeme, *p++
      InsideNumber(get
      return;
   }
   OutsideNumber(getch
}

void InsideNumber(int c){
   if(isdigit(c)){
      *p++ = c;
      InsideNumber(getchar());
      return;
   }
   *p='\0', printf("%d\n",atoi(lexeme));
   OutsideNumber(getchar());
}
//
```

```
%option main
%{
#include <stdio.h> // printf
#include <stdlib.h> // atoi
%}
%%
[0-9]+   printf("%d\n",atoi(yytext));
.|\n     ;
%%
```

```
#include <stdio.h> // getchar EOF printf
#include <stdlib.h> // atoi
#include <ctype.h> // isdigit

int main(void){
   const int MAX_LEXEME_LENGTH = 1000;
   char lexeme[MAX_LEXEME_LENGTH], *p;
   int c;
   goto OutsideNumber;

umber:
   char();
   F == c)
   eturn 0;
   sdigit(c)){
      p = lexeme, *p++ = c;
   oto InsideNumber;

   OutsideNumber;

mber:
   char();
   sdigit(c)){
      *p++ = c;
      goto InsideNumber;
   }
   *p='\0', printf("%d\n",atoi(lexeme));
   goto OutsideNumber;
}


//
```

Una sola Invocación a yylex

```
%option caseless
DIGITS [0-9]+
%%
[ \t]+                  ;
{DIGITS}                |
{DIGITS}"."{DIGITS}?    |
{DIGITS}?"."{DIGITS}    printf("(Number, %f)\n", atof(yytext));
"+"                     |
"*"                     |
"-"                     |
"/"                     |
"%"                     printf("(%c)\n", *yytext);
\n                      puts("(pop)");
"dup"                   |
"swap"                  |
"clear"                 |
"top"                   printf("(%s)\n", yytext);
.                       printf("This \"%s\" isn't a token.\n", yytext);
%%
int main(){
  yylex();
}
```

Proto Scanner de Calculadora con Notación Polaca Inversa

Tantas Invocaciones a yylex como Tokens

```
%option noyywrap caseless
%{
typedef enum{
   Number='0',
   Addition='+',
   Multiplication='*',
   Substraction='-',
   Division='/',
   Remainder='%',
   NewLine='\n',
   Duplicate,
   Swap,
   Clear,
   Top,
   LexError,
   NoMoreTokens=0
} TokenType;
%}
DIGITS   [0-9]+
%%
[ \t]+                ;
{DIGITS}              |
{DIGITS}"."{DIGITS}?  |
{DIGITS}?"."{DIGITS}  return Number;
"+"                   return Addition;
"*"                   return Multiplication;
"-"                   return Substraction;
"/"                   return Division;
"%"                   return Remainder;
\n                    return NewLine;
"dup"                 return Duplicate;
"swap"                return Swap;
"clear"               return Clear;
"top"                 return Top;
.                     return LexError;
```

```
%%
int main(){
   TokenType type;
   while( ( type = yylex()) != NoMoreTokens )
      switch(type){
         case LexError:
            printf("This \"%s\" isn't a token.\n",yytext);
            break;
         case Number:
            printf("(Number, %f)\n", atof(yytext));
            break;
         case NewLine:
            puts("(pop)");
            break;
         case Duplicate:
         case Swap:
         case Clear:
         case Top:
            printf("(%s)\n", yytext);
            break;
         default:
            printf("(%c)\n", *yytext);
            break;
      }
}
```

Scanner de Lenguaje Tipo Pascal – Programado a Mano

```
bool GetNextToken(FILE *in, /*out*/ Token *t){
    int c; // el caracter

    for(;;){
        while(isspace(c = fgetc(in))); // Saltea espacios
        if( c == EOF) return false;
        if( '{' == c){                 // Saltea comentario
            do{
                c = fgetc(in);
                if(EOF == c) return false;
                if('\n' == c){
                    *t->lexeme='\n';
                    t->type=LexicalError;
                    return true;
                }
            }while( '}' != c );
            continue;                  // go back to check for more spaces
        }
        break;
    }

    // No se pudo leer más caracteres
    if( c == EOF) return false;

    // Identificador ó palabra reservada
    if( isalpha(c) ){
        char *p = t->lexeme;
        do{
            *p++ = (char)c;
            c = fgetc(in);
        }while( isalnum(c) || c == '_' );
        ungetc(c, in);
        *p = '\0';
        //TODO: symbol table
        //static symbols[]={{lexeme,value}};                              }
        if(      strcmp(t->lexeme, "if")        == 0 ) t->type = If;
        else if( strcmp(t->lexeme, "then")      == 0 ) t->type = Then;
        else if( strcmp(t->lexeme, "begin")     == 0 ) t->type = Begin;
        else if( strcmp(t->lexeme, "end")       == 0 ) t->type = End;
        else if( strcmp(t->lexeme, "procedure") == 0 ) t->type = Procedure;
        else if( strcmp(t->lexeme, "function")  == 0 ) t->type = Function;
        else if( strcmp(t->lexeme, "read")      == 0 ) t->type = Read;
        else if( strcmp(t->lexeme, "write")     == 0 ) t->type = Write;
        else                                           t->type = Identifier;
        return true;
    }

    // Literal Entero o Flotante
    if( isdigit(c) ){
        char *p = t->lexeme;
        do{
            *p++ = (char)c;
            c = fgetc(in);
        }while( isdigit(c) );
        if( '.' != c ){
            ungetc(c, in);
            *p = '\0';
            t->type = IntegerLiteral;
            return true;
        }
        do{
            *p++ = (char)c;
            c = fgetc(in);
        }while( isdigit(c) );
        ungetc(c, in);
        *p = '\0';
        t->type = FloatingLiteral;
        return true;
    }

    // Punctuators o error léxico
    t->lexeme[0] = (char)c;
    t->lexeme[1] = '\0';
    switch( c ) {
        // TODO: Simplify?
        case '(': t->type=LeftParenthesis;  return true;
        case ')': t->type=RightParenthesis; return true;
        case ';': t->type=Semicolon;        return true;
        case ',': t->type=Comma;            return true;
        case '+': t->type=Plus;             return true;
        case '-': t->type=Minus;            return true;
        case '*': t->type=Times;            return true;
        case '/': t->type=Division;         return true;
        case '=': t->type=Assignment;       return true;
        default : t->type=LexicalError;     return true;
    }
```

Scanner de Lenguaje Tipo Pascal – Programado a Mano

```
DIGIT    [0-9]
ID       [a-z][a-z0-9]*

%%

{DIGIT}+              return IntegerLiteral;

{DIGIT}+"."{DIGIT}*   return FloatingLiteral;

if                    return If;
then                  return Then;
begin                 return Begin;
end                   return End;
procedure             return Procedure;
function              return Function;
read                  return Read;
write                 return Write;

{ID}                  return Identifier;

"-"                   return LeftParenthesis;
"+"                   return RightParenthesis;
"*"                   return Semicolon;
"/"                   return Comma;
"("                   return Assignment;
")"                   return Plus;
","                   return Minus;
";"                   return Times;
"="                   return Division;

"{"[^}\n]*"}"         // eat up one-line comments

[ \t\n]+              // eat up whitespace

"{"[^}\n]*$           {yytext[0]='\n',yytext[1]='\0'; return LexicalError;} // no multiline comments

.                     return LexicalError;

%%
```

¿Consultas?

Fin de la clase