Construcción de Compiladores - Laboratorio Guiado No. 1

Gabriel Brolo, Bidkar Pojoy

Julio de 2024

Adrian Fulladolsa Palma

Carnet: 21592

1. Introducción

En este laboratorio, explicaremos el proceso de construcción de un compilador utilizando herramientas como Lex y Yacc. Utilizaremos un lenguaje de programación simple que soporta expresiones aritméticas y asignaciones de variables. Lea todo el documento antes de comenzar a realizar las actividades.

6. Actividades por completar

Utilice el lenguaje definido y las herramientas Lex y Yacc para realizar las siguientes tareas:

1. Cree un programa que asigne un valor a una variable.

Código generado en archivo Actividades/actividad1.txt

```
C/C++
a = 1
```

```
root@1322a0f84bdf:/home# ./calc < Actividades/actividad1.txt
Assign a = 1
```

2. Cree un programa que realice una operación con aritmética simple.

Código generado en archivo Actividades/actividad2.txt

```
C/C++
b = 1 + 2
```

```
root@1322a0f84bdf:/home# ./calc < Actividades/actividad2.txt
Assign b = 3
```

3. Experimente con expresiones más complejas y verifique que el compilador las procese correctamente.

Código generado en archivo Actividades/actividad3.txt

```
C/C++
a = 4
b = 2
c = 14

d = ( 2 * a / b ) + (2 * c - 4)
e = 2 * c
g = b * 14
f = 2 * c / b * 14
h = e / g
```

```
root@1322a0f84bdf:/home# ./calc < Actividades/actividad3.txt
Assign a = 4
Assign b = 2
Assign c = 14
Assign d = 28
Assign e = 28
Assign g = 28
Assign f = 196
Assign h = 1
```

4. Modifique el lenguaje para incluir la asignación de variables con expresiones aritméticas.

La versión que estoy utilizando del lenguaje ya incluye la asignación de variables con expresiones aritméticas por lo que no es necesario realizar cambios, acá hay ejemplos de asignaciones de variables con expresiones aritméticas:

Código generado en archivo Actividades/actividad4.txt

```
C/C++
a = 1 + 2
b = a * 15
c = b / 15
```

```
d = b / c + a - b + 144
```

```
root@1322a0f84bdf:/home# ./calc < Actividades/actividad4.txt
Assign a = 3
Assign b = 45
Assign c = 3
Assign d = 237
```

5. Agregue manejo de errores al compilador para detectar tokens inválidos en el programa fuente.

Archivo simple_language.l modificado

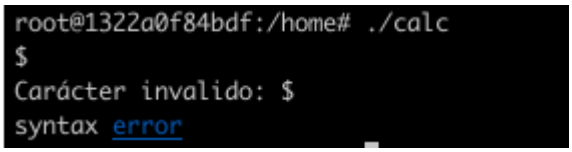
```
C/C++
   %{
   #include <cstdlib>
   #include <string>
   #include "y.tab.h"
   #include <iostream>
   %}

   %%

   [a-zA-Z][a-zA-Z0-9]*    { yylval.str = new std::string(yytext);
   return ID; }
   [0-9]+                  { yylval.num = strtol(yytext, NULL, 10);
   return NUMBER; }
   "+"                     { return '+'; }
   "-"                     { return '-'; }
   "*"                     { return '*'; }
   "/"                     { return '/'; }
   "="                     { return '='; }
   ":"                     { return ':'; }
   [ \t]                   ;  // skip whitespace
   \n                      ; /* skip newline */
   .                       { std::cout << "Carácter invalido: " <<
   yytext << std::endl; return 0; }

   %%

   int yywrap() {
       return 1;
   }
```



Se modificó la línea correspondiente a la expresión .|/n para que únicamente se ignore el newline y se agregó una línea para que al encontrar un carácter no definido previamente se retorna el mensaje “Carácter inválido: “ y se muestre el carácter que no fue aceptado.

6. Experimente con la precedencia de operadores en el lenguaje y observe cómo afecta la generación del árbol sintáctico.

Archivo simple_language.y modificado

C/C++

```
%{
#include <iostream>
#include <string>
#include <map>
static std::map<std::string, int> vars;
inline void yyerror(const char *str) { std::cout << str <<
std::endl; }
int yylex();
%}

%union { int num; std::string *str; }

%token<num> NUMBER
%token<str> ID
%type<num> expression
%type<num> assignment

%right '='
%left '*' '/'
%left '+' '-'

%%

program: statement_list
        ;

statement_list: statement
    | statement_list statement
    ;

statement: assignment
    | expression ':'          { std::cout << $1 << std::endl; }
    ;
```

```
assignment: ID '=' expression
    {
        printf("Assign %s = %d\n", $1->c_str(), $3);
        $$ = vars[*$1] = $3;
        delete $1;
    }
    ;

expression: NUMBER                  { $$ = $1; }
    | ID                            { $$ = vars[*$1];      delete
$1; }
    | expression '/' expression     { $$ = $1 / $3; }
    | expression '*' expression     { $$ = $1 * $3; }
    | expression '+' expression     { $$ = $1 + $3; }
    | expression '-' expression     { $$ = $1 - $3; }
    ;

%%

int main() {
    yyparse();
    return 0;
}
```

```
root@1322a0f84bdf:/home# ./calc < Actividades/actividad4.txt
Assign a = 3
Assign b = 45
Assign c = 3
Assign d = 0
```

Se modificó el orden en el que estaban los operadores de tal manera que se contemplan primero * y / antes que + y -. Esto retorno los mismos resultados para las primeras 3 asignaciones al ejecutar el archivo Actividades/actividad4.txt pero en la última se obtuvo 0 en vez de 237 ya que se está realizando la división de último. Esto se debe a que se está afectando la precedencia de los operadores, al colocar la línea `%left '*' '/'` antes que la línea `%left '+' '-'` se está generando un árbol sintáctico donde primero se van a considerar las operaciones de suma y resta antes de las de multiplicación y división. Esto causó la inversión del orden de operaciones llevando a resultados erroneos.

Repositorio de GitHub:

https://github.com/adrianfulla/Compilers-2024/tree/Lab1

7. Entregables

Documento PDF con capturas de pantalla de la ejecución del ambiente en Docker de Lex y Yacc, o de su propio entorno, para realizar las actividades solicitadas.

El documento debe de incluir el enlace a un repositorio privado de Github con todo el código utilizado para la realización de este laboratorio guiado. Recuerde que debe de compartir todos los repositorios utilizados en este curso con el catedrático y los auxiliares. La información de los usuarios se encuentra en Canvas.

8. Rúbrica

La siguiente rúbrica ser´a utilizada para asignar puntos a las tareas anteriores:

Tarea	Puntos
Crear un programa de asignaci´on	10
Crear un programa de operación	15
Agregar manejo de errores	15
Experimentar con la precedencia	15
Completar todas las tareas	10
Respuestas comprensibles y claras	10
Seguimiento de instrucciones	10
Organización y presentación adecuada	10