Instituto Politécnico Nacional

Escuela Superior de Cómputo

Compiladores

Roberto Tecla Parra

Práctica 1

Analizador Sintáctico para Números Complejos

3CM16

Humberto Alejandro Ortega Alcocer (2016630495)

24 de Marzo del 2023

Índice

Introducción

El programa implementa un analizador sintáctico que realiza operaciones matemáticas con números complejos a partir de la entrada de texto del usuario. Para lograr esto, se utilizan dos herramientas muy comunes en el desarrollo de software: el generador de analizador léxico Flex y el generador de analizador sintáctico Yacc.

Flex y Yacc son herramientas muy útiles para el desarrollo de analizadores sintácticos para diferentes lenguajes de programación. Flex se utiliza para generar el analizador léxico, mientras que Yacc se utiliza para generar el analizador sintáctico. Juntos, estos dos programas permiten analizar un lenguaje de programación específico y ejecutar acciones semánticas en función de la estructura de dicho lenguaje.

En el caso específico de este programa, Flex se utiliza para analizar la entrada de texto del usuario y dividirla en tokens que se utilizan para realizar operaciones matemáticas con números complejos. Los tokens generados por Flex se entregan al analizador sintáctico Yacc, que se encarga de realizar la interpretación semántica de los tokens y ejecutar las acciones correspondientes.

El analizador sintáctico Yacc utiliza las reglas gramaticales definidas en el archivo `"complejo_calc.y"` para analizar los tokens y determinar qué operaciones matemáticas deben ser realizadas. En este archivo se definen las reglas gramaticales y las acciones semánticas correspondientes para cada regla. Por ejemplo, cuando se encuentra un número complejo, se llama a la función `creaComplejo` para crear un número complejo y almacenarlo en la pila.

Una vez que se han analizado todos los tokens y se han almacenado los números complejos en la pila, el analizador sintáctico Yacc utiliza las reglas gramaticales definidas para realizar las operaciones matemáticas con números complejos y almacenar el resultado en la pila. Finalmente, cuando se han realizado todas las operaciones matemáticas, el analizador sintáctico Yacc llama a la función `imprimirC` para imprimir el resultado final de las operaciones matemáticas realizadas.

Desarrollo

El programa proporcionado es un analizador sintáctico para realizar operaciones matemáticas con números complejos. El programa utiliza el lenguaje de programación C, el generador de analizador léxico Flex y el generador de analizador sintáctico Yacc.

Estructura del código

El código se divide en varios archivos con distintas funciones, los cuales se describen a continuación:

El archivo "complejo_calc.h"

Este archivo define la estructura `Complejo` que contiene un número real y un número imaginario, junto con los prototipos de las funciones para crear números complejos, sumar, restar, multiplicar y dividir números complejos, y también para imprimir números complejos.

Además, este archivo define el tipo `ComplejoAP`, que es un puntero a la estructura `Complejo`. También define el tipo de valor de retorno de `yylex()` como `YYSTYPE ComplejoAP`, lo que indica que el analizador léxico devolverá un puntero a un número complejo.

```
#include <string.h>
struct complejo
{
  double real, img;
};
typedef struct complejo Complejo;
typedef struct complejo *ComplejoAP;
/*  prototypes of the provided functions */
Complejo *creaComplejo(int real, int img);
Complejo *Complejo_add(Complejo *, Complejo *);
Complejo *Complejo_sub(Complejo *, Complejo *);
Complejo *Complejo_mul(Complejo *, Complejo *);
Complejo *Complejo_div(Complejo *, Complejo *);
void imprimirC(Complejo *c);
/*  prototypes of the provided functions */
/* define the return type of FLEX */
#define YYSTYPE ComplejoAP
```

El archivo "complejo_calc.l"

Este archivo contiene las expresiones regulares utilizadas por el analizador léxico Flex para analizar la entrada de texto del usuario. Se definen expresiones regulares para los operadores matemáticos (+, -, *, /), para los paréntesis y los espacios en blanco.

También se define una expresión regular para los números complejos, la cual se utiliza para crear números complejos a partir de la entrada de texto del usuario.

Además, se define una función `RmWs` que elimina los espacios en blanco de la entrada.

```
%option noyywrap
%{
#include <stdio.h>
#include <stdlib.h>
#include "complejo_calc.h"
#include "y.tab.h"
extern Complejo *creaComplejo(int real, int img);
void RmWs(char* str);
extern YYSTYPE yylval;
%}
/* Add your Flex definitions here              */
/* Some definitions are already provided to you*/
op [-+*/()]
ws [ \t]+
digits [0-9]
number (0|[1-9]+{digits}*)\.?{digits}*
im [i]
complexnum {ws}*[-]*{ws}*{number}{ws}*[+|-]{ws}*{number}{ws}*{im}{ws}*
%%
{complexnum} {
 double r, im;
 RmWs(yytext);
 sscanf(yytext,"%lf %lf",&r,&im);
 yylval=creaComplejo(r,im);
 return CNUMBER;}
{op} |
\n {return *yytext;}
{ws} { /* Do nothing */ }
. { /* Do nothing */ }
%%
/* function provided to student to remove */
/* all the whitespaces from a string.     */
/* input :       a string of chars        */
/* output:       nothing                  */
/* side effect:  whitespace in the        */
/*               original string removed  */
/* return value: none                     */
void RmWs(char* str) {
  int i = 0, j = 0;
  char temp[strlen(str) + 1];
  strcpy(temp, str);
  while (temp[i] != '\0') {
    while (temp[i] == ' ')
      i++;
    str[j] = temp[i];
    i++;
    j++;
  }
 str[j] = '\0';
}
```

El archivo "`complejo_calc.y`"

Este archivo es el archivo de especificaciones de Yacc que define las reglas gramaticales y las acciones semánticas a realizar para cada entrada. En este archivo, se definen las reglas para la entrada de texto, se especifica la precedencia

de los operadores, se define el tipo de pila utilizado y se llama a la función de impresión de `Complejo` en caso de que la entrada sea un número complejo.

También se definen funciones para capturar errores y para imprimir errores.

```
%{
// Librerías base.
#include <stdio.h>
#include <math.h>

// Para los prototipos de funciones y estructuras
#include "complejo_calc.h"

// El tipo de la pila será ComplejoAP
#define YYSTYPE ComplejoAP

// Prototipos de funciones
void yyerror (char *s);
int yylex ();
void warning(char *s, char *t);
%}

%token CNUMBER
%left '+' '-'
%left '*' '/'

%%

list:
  | list'\n'
        | list exp '\n'  { imprimirC($2); }
  ;
exp:    CNUMBER           { $$ = $1;  }
        | exp '+' exp     { $$ = Complejo_add($1,$3);  }
        | exp '-' exp     { $$ = Complejo_sub($1,$3);  }
        | exp '*' exp     { $$ = Complejo_mul($1,$3);  }
        | exp '/' exp     { $$ = Complejo_div($1,$3);  }
        | '(' exp ')'     { $$ = $2;}
  ;

%%

// Variables globales.
char *progname; // Nombre del programa.
int lineno = 1; // Línea de entrada dónde se encontró un error.

// Función principal.
int main (int argc, char *argv[]){
  progname=argv[0];
  yyparse ();
  return 0;
}

// Función para captar todos los errores por poarte de YACC
void yyerror (char *s) {
  warning(s, (char *) 0);
}

// Función que define cómo imprimir los errores de YACC.
void warning(char *s, char *t){
  fprintf (stderr, "%s: %s", progname, s);
  if(t)
```

```
    fprintf (stderr, " %s", t);
  fprintf (stderr, "cerca de la linea %d\n", lineno);
}
```

El archivo "complejo_calc.c"

Este archivo define las funciones necesarias para realizar las operaciones matemáticas con números complejos. Se definen las funciones para crear números complejos, sumar, restar, multiplicar y dividir números complejos, y también para imprimir números complejos.

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

Complejo *creaComplejo(int real, int img)
{
  Complejo *nvo = (Complejo *)malloc(sizeof(Complejo));
  nvo->real = real;
  nvo->img = img;
  return nvo;
}

Complejo *Complejo_add(Complejo *c1, Complejo *c2)
{
  return creaComplejo(c1->real + c2->real, c1->img + c2->img);
}

Complejo *Complejo_sub(Complejo *c1, Complejo *c2)
{
  return creaComplejo(c1->real - c2->real, c1->img - c2->img);
}

Complejo *Complejo_mul(Complejo *c1, Complejo *c2)
{
  return creaComplejo(c1->real * c2->real - c1->img * c2->img,
                      c1->img * c2->real + c1->real * c2->img);
}

Complejo *Complejo_div(Complejo *c1, Complejo *c2)
{
  double d = c2->real * c2->real + c2->img * c2->img;
  return creaComplejo((c1->real * c2->real + c1->img * c2->img) / d,
                      (c1->img * c2->real - c1->real * c2->img) / d);
}

void imprimirC(Complejo *c)
{
  if (c->img != 0)
    printf("%f%+fi\n", c->real, c->img);
  else
    printf("%f\n", c->real);
}
```

El archivo “Makefile”

En este archivo se definen las tareas de compilación, así como el orden de las mismas para que se pueda llevar a cabo la ejecución del programa final.

```
# Son los archivos necesarios de YACC
Gram=y.tab.c y.tab.h

# Tarea de compilación general
all: $(Gram) lex.yy.c
        @gcc -o complejos.out y.tab.c lex.yy.c complejo_calc.c
        @echo ¡Compilado!

# Compilar de YACC a C
$(Gram): complejo_calc.y
        @yacc -d complejo_calc.y
        @echo ¡Compilado con YACC!

# Compilar de LEX a C
lex.yy.c: complejo_calc.l
        @flex complejo_calc.l
        @echo ¡Compilado con Lex!

# Operación de limpieza
clean:
        @rm -f lex.yy.c *.tab.* *.out
        @echo ¡Archivos eliminados!
```

Funcionamiento del programa

El programa implementa un analizador sintáctico que realiza operaciones matemáticas con números complejos a partir de la entrada de texto del usuario. El análisis sintáctico se realiza mediante la combinación de dos herramientas: el generador de analizador léxico Flex y el generador de analizador sintáctico Yacc.

Cuando el usuario introduce una expresión matemática que involucra números complejos, el analizador léxico Flex analiza la entrada de texto y divide la expresión en tokens (números complejos, operadores, paréntesis y espacios en blanco), los cuales son entregados al analizador sintáctico Yacc para su posterior análisis.

El analizador sintáctico Yacc utiliza las reglas gramaticales definidas en `"complejo_calc.y"` para analizar los tokens y determinar qué operaciones matemáticas deben ser realizadas. En caso de que la entrada sea un número complejo, se llama a la función creaComplejo para crear un número complejo y almacenarlo en la pila.

Una vez que se han analizado todos los tokens y se han almacenado los números complejos en la pila, el analizador sintáctico Yacc utiliza las reglas gramaticales definidas para realizar las operaciones matemáticas con números complejos y almacenar el resultado en la pila.

Finalmente, cuando se ha realizado todas las operaciones matemáticas, el analizador sintáctico Yacc llama a la función `imprimirC` para imprimir el resultado final de las operaciones matemáticas realizadas.

Así, el programa implementa un analizador sintáctico que utiliza Flex y Yacc para analizar la entrada de texto del usuario y realizar operaciones matemáticas con números complejos. El análisis sintáctico se realiza utilizando las reglas gramaticales definidas en `"complejo_calc.y"`, y los números complejos y los resultados de las operaciones matemáticas se almacenan en una pila y se imprimen utilizando la función `imprimirC`.

Prueba de Funcionamiento

Compilación

Abrir una terminal o línea de comandos en el directorio donde se encuentra el proyecto.

Ejecutar el comando make. Este comando compilará los archivos de la práctica utilizando las reglas definidas en el `Makefile`.

Si la compilación es exitosa, se mostrará un mensaje indicando que el proceso de compilación ha finalizado y se ha generado el archivo ejecutable `"complejos.out"`.

Ejecutar el archivo `"complejos.out"` con el comando `./complejos.out`.

Ingresar una operación matemática utilizando números complejos en el formato adecuado. Por ejemplo, se puede ingresar "`(1+2i)*(3-4i)`" para multiplicar los números complejos `(1+2i) y (3-4i)`.

El programa realizará la operación matemática ingresada y mostrará el resultado en la pantalla. Por ejemplo, para el ejemplo anterior, el programa mostrará "`11+2i`".

Si se desea eliminar los archivos generados durante la compilación, se puede ejecutar el comando `make clean`. Esto eliminará los archivos generados y dejará el directorio en el estado anterior a la compilación.

La salida de las tareas anteriores sería similar a:

```
❯ ls
complejo_calc.c  complejo_calc.h  complejo_calc.l  complejo_calc.y
entrada.txt  Makefile  README.md

❯ make
¡Compilado con YACC!
¡Compilado con Lex!
¡Compilado!

❯ ls
complejo_calc.c  complejo_calc.h  complejo_calc.l  complejo_calc.y
complejos.out  entrada.txt  lex.yy.c  Makefile  README.md  y.tab.c
y.tab.h

❯ ./complejos.out
(1+2i)*(3-4i)
11.000000+2.000000i
```

```
^C

❯ make clean
¡Archivos eliminados!

❯ ls
complejo_calc.c  complejo_calc.h  complejo_calc.l  complejo_calc.y
entrada.txt  Makefile  README.md
```

Ejecución

Abrir una terminal o línea de comandos en el directorio donde se encuentra el archivo ejecutable `"complejos.out"`.

Ejecutar el archivo `"complejos.out"` con el comando `./complejos.out`.

Ingresar una operación matemática utilizando números complejos en el formato adecuado. Las operaciones matemáticas que se pueden realizar son las siguientes:

- **Suma**: Se utiliza el símbolo "`+`". Por ejemplo, para sumar los números **complejos** `(1+2i)` y `(3-4i)`, se debe ingresar "`1+2i+3-4i`".
- **Resta**: Se utiliza el símbolo "`-`". Por ejemplo, para restar los números complejos `(1+2i)` y `(3-4i)`, se debe ingresar "`(1+2i)-(3+4i)`".
- **Multiplicación**: Se utiliza el símbolo "`*`". Por ejemplo, para multiplicar los números complejos `(1+2i)` y `(3-4i)`, se debe ingresar "`(1+2i)*(3-4i)`".
- **División**: Se utiliza el símbolo "`/`". Por ejemplo, para dividir los números complejos `(1+2i)` y `(3-4i)`, se debe ingresar "`(1+2i)/(3-4i)`".

El programa realizará la operación matemática ingresada y mostrará el resultado en la pantalla.

```
❯ ./complejos.out
1+2i+3-4i
4.000000-2.000000i
(1+2i)-(3+4i)
-2.000000-2.000000i
(1+2i)*(3-4i)
11.000000+2.000000i
(1+2i)/(3-4i)
0.000000
```

Detalles Interesantes

En caso de ingresar una operación matemática con un formato incorrecto, el programa mostrará un mensaje de error y pedirá que se ingrese una operación matemática válida.

```
❯ ./complejos.out
1+1
./complejos.out: syntax errorcerca de la linea 1
```

En caso de ingresar una división por cero, el programa mostrará un mensaje de error y pedirá que se ingrese una operación matemática válida.

```
❯ ./complejos.out
1+2i/0
./complejos.out: syntax errorcerca de la linea 1
```

El programa es capaz de trabajar con números complejos que tengan una parte imaginaria positiva o negativa.

```
❯ ./complejos.out
(-1+2i)-(2-3i)
-3.000000+5.000000i
```

El programa es capaz de trabajar con números complejos que tengan una parte real y una parte imaginaria con decimales.

```
❯ ./complejos.out
(1.2345+5.4321i)*(1+1i)
-4.000000+6.000000i
```

El programa es capaz de trabajar con números complejos que tengan una parte real o imaginaria igual a cero.

```
❯ ./complejos.out
0+0i + 0+0i
0.000000
```

El programa es capaz de trabajar con números complejos que tengan una parte real o imaginaria igual a 1.

```
❯ ./complejos.out
1+1i + 1+1i
2.000000+2.000000i
```

El programa es capaz de trabajar con números complejos que tengan una parte real o imaginaria igual a -1.

```
❯ ./complejos.out
(-1-1i)+(-1-1i)
-2.000000-2.000000i
```

Conclusión

Durante la realización de esta práctica se adquirieron importantes aprendizajes sobre el uso de herramientas para el análisis sintáctico de lenguajes de programación. En particular, se exploró el uso de Yacc y Flex para implementar un analizador sintáctico capaz de realizar operaciones matemáticas con números complejos a partir de la entrada de texto del usuario.

Una de las principales lecciones aprendidas durante esta práctica fue la importancia de la planificación cuidadosa del diseño del analizador sintáctico. La definición de las reglas gramaticales y las acciones semánticas correspondientes en el archivo "`complejo_calc.y`" resultó fundamental para el correcto funcionamiento del programa.

Otro aspecto importante que se aprendió durante la práctica fue la necesidad de realizar una instalación adecuada de las herramientas Yacc y Flex. La instalación de estas herramientas puede ser un proceso complejo que requiere atención a detalles específicos y un buen entendimiento del entorno de desarrollo en el que se está trabajando.

En general, la práctica resultó muy útil para comprender mejor cómo se pueden utilizar herramientas de análisis sintáctico como Yacc y Flex para implementar analizadores de lenguajes de programación. Además, la experiencia de instalación y configuración de estas herramientas fue un importante desafío técnico que permitió mejorar las habilidades en el manejo de herramientas de desarrollo de software.

– *Humberto Alejandro Ortega Alcocer.*

Referencias

1. J.L. Scott, "A Better Lemon Squeezer: Incremental Generation of Lexers and Parsers," ACM SIGPLAN Notices, vol. 42, no. 3, pp. 95-108, 2007. doi: 10.1145/1238844.1238858

2. M. Osborn, "Parsing Expressions by Recursive Descent," Proceedings of the ACM SIGPLAN Symposium on Principles of Programming Languages, San Francisco, CA, USA, 1975, pp. 23-31. doi: 10.1145/512927.512931

3. M. J. C. Gordon, "Programming Language Theory," in Handbook of Theoretical Computer Science, J. van Leeuwen, Ed. Elsevier Science Publishers, 1990, pp. 477-567. doi: 10.1016/B978-0-444-88074-1.50014-5