**INSTITUTO POLITÉCNICO NACIONAL**

**ESCUELA SUPERIOR DE CÓMPUTO**

PRÁCTICA 1

Calculadora básica con números racionales

**Profesor:**

• **Tecla Parra Roberto**

**Alumno:**

* **Pacheco Delgado José Jacobeth**

**Grupo:**

• **3CM8**

**Materia:**

• **Compiladores**

**Introducción**

La consiste en realizar una calculadora básica de **números racionales**, se debe realizar con lenguaje C con yacc y flex.

Recordando que yacc es un programa para generar analizadores sintácticos, y flex para generar analizadores léxicos.

La calculadora básica debe de contar con operaciones como: suma, resta, multiplicación y división entre números racionales.

**Objetivos**

* Desarrolle una calculadora para realizar operaciones con números racionales en yacc y si es requerido usar flex.

**Desarrollo**

Primero, se debe realizar la especificación de yacc para poder operar números racionales, esto consiste en modificar la gramática y las acciones para poder crear números racionales y operarlos. En el siguiente código se muestra las especificaciones de yacc.

Hoc1.y

%{

#include <stdio.h>

#include <math.h>

#include "racional\_cal.h"

*// int yylex ();*

void warning(char \*s, char \*t);

%}

%token racionalnum number fr

%left '+' '-'

%left '\*' '/'

%%

list:

| list'\n'

| list exp '\n' { imprimirR( $2 ); }

;

exp: racionalnum { $$ = $1; }

| exp '+' exp { $$ = racionalSuma( $1, $3 ); }

| exp '-' exp { $$ = racionalResta( $1, $3 ); }

| exp '\*' exp { $$ = racionalMultiplicar( $1, $3 ); }

| exp '/' exp { $$ = racionalDividir( $1, $3 ); }

| racionalnum '+' racionalnum { $$ = racionalSuma( $1, $3 ); }

| racionalnum '-' racionalnum { $$ = racionalResta( $1, $3 ); }

| racionalnum '\*' racionalnum { $$ = racionalMultiplicar( $1, $3 ); }

| racionalnum '/' racionalnum { $$ = racionalDividir( $1, $3 ); }

;

%%

#include <stdio.h>

#include <ctype.h>

char \*progname;

int lineno = 1;

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);

}

Como se observa, se agrega como símbolo terminal al elemento racionalnum y operaciones (**op**) que será el operador para realizar las operaciones especificadas.

De igual forma en el caso de expresión se modificaron las acciones gramaticales, donde en caso de obtener un número racional este se imprimirá en pantalla para facilitar el observar los resultados de las operaciones. En el caso de las operaciones, las acciones gramaticales mandan llamar a los métodos especificados en **racional\_cal.c**, dichos métodos llevan a cabo las operaciones entre los numeradores y denominadores de los números racionales a operar.

%option noyywrap

%{

#include <stdio.h>

#include <stdlib.h>

#include "racional\_cal.h"

#include "y.tab.h"

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}\*)

sub [n]

fr [+]

racionalnum {ws}\*{sub}?{ws}\*{number}{ws}\*{fr}{ws}\*{number}{ws}\*

%%

{racionalnum} {

int num, den;

RmWs(yytext);

printf("(%s)", yytext);

sscanf(yytext,"%d %d", &num, &den);

yylval=creaRacional(num, den);

return racionalnum;}

{op} |

\n {return \*yytext*;}*

{ws} { /\* Do nothing \*/ }

. { /\* Do nothing \*/ }

%%

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'*;*

}

racional\_cal.c

#include <stdio.h>

#include "racional\_cal.h"

#include <stdlib.h>

int main() { **return** yyparse(); }

int yyerror(**const** char\* s) {

printf("%s**\n**", s);

**return** 0;

}

Racional \*creaRacional(int num, int den){

Racional \*nvo;

**if**(den==0)

**return** (Racional \*)NULL;

nvo=(Racional \*)malloc(**sizeof**(Racional));

**if**(!nvo){

puts("no hay memoria para crear Racional ");

**return** (Racional \*)NULL;

}

nvo->num=num;

nvo->den=den;

**return** nvo;

}

void asignar(Racional \*r, int num, int den){

r -> num = num; r -> den = den;

}

int numerador(Racional \*r){ **return** r ->num; }

int denominador(Racional \*r){ **return** r ->den; }

Racional\* racionalSuma(Racional \*r, Racional \*s){

int nvonum = (r -> num \* s -> den) + (s -> num \* r -> den);

int nvoden = r -> den \* s -> den;

Racional \*nvo = creaRacional(nvonum, nvoden);

**return** nvo;

}

Racional\* racionalResta(Racional \*r, Racional \*s){

int nvonum = (r -> num \* s -> den) - (s -> num \* r -> den);

int nvoden = r -> den \* s -> den;

Racional \*nvo = creaRacional(nvonum, nvoden);

**return** nvo;

}

Racional\* racionalMultiplicar(Racional \*r, Racional \*s){

int nvonum = r -> num \* s -> num;

int nvoden = r -> den \* s -> den;

Racional \*nvo = creaRacional(nvonum, nvoden);

**return** nvo;

}

Racional\* racionalDividir(Racional \*r, Racional \*s){

int nvonum = r -> num \* s -> den;

int nvoden = r -> den \* s -> num;

Racional \*nvo = creaRacional(nvonum, nvoden);

**return** nvo;

}

int esIgual(Racional \*r, Racional \*s){

**return** (r -> num \* s -> den) == (r -> den \* s -> num);

}

void imprimirR(void \*r){

Racional \*p = (Racional\*)r;

printf(" = (%d / %d)**\n**",p -> num, p -> den);

}

Racional \*copiar(Racional \*r){

**return** creaRacional(r -> num, r -> den);

}

Adémas para agilizar las pruebas fue creado un archivo **Makefile** ( es un archivo (necesariamente llamado **Makefile**) que contiene una serie de **directivas** e **instrucciones** utilizadas por una **automatización** de acumulación, herramienta para generar **un objetivo / meta**)

Gram=y.tab.c y.tab.h

all: $(Gram) lex.yy.c racional\_cal.c

@gcc -o racio y.tab.c lex.yy.c racional\_cal.c

@echo Compiled

racional\_cal.c:

@echo "Si esta"

**$(**Gram**)**: racional\_cal.y

@yacc -d racional\_cal.y

lex.yy.c: racional\_cal.l

@flex racional\_cal.l

clean:

@rm -f \*.out lex.yy.c \*.tab.\* racio

@echo Clean

En la **figura 1** se muesta el programa funcionando con los objetivos requeridos.

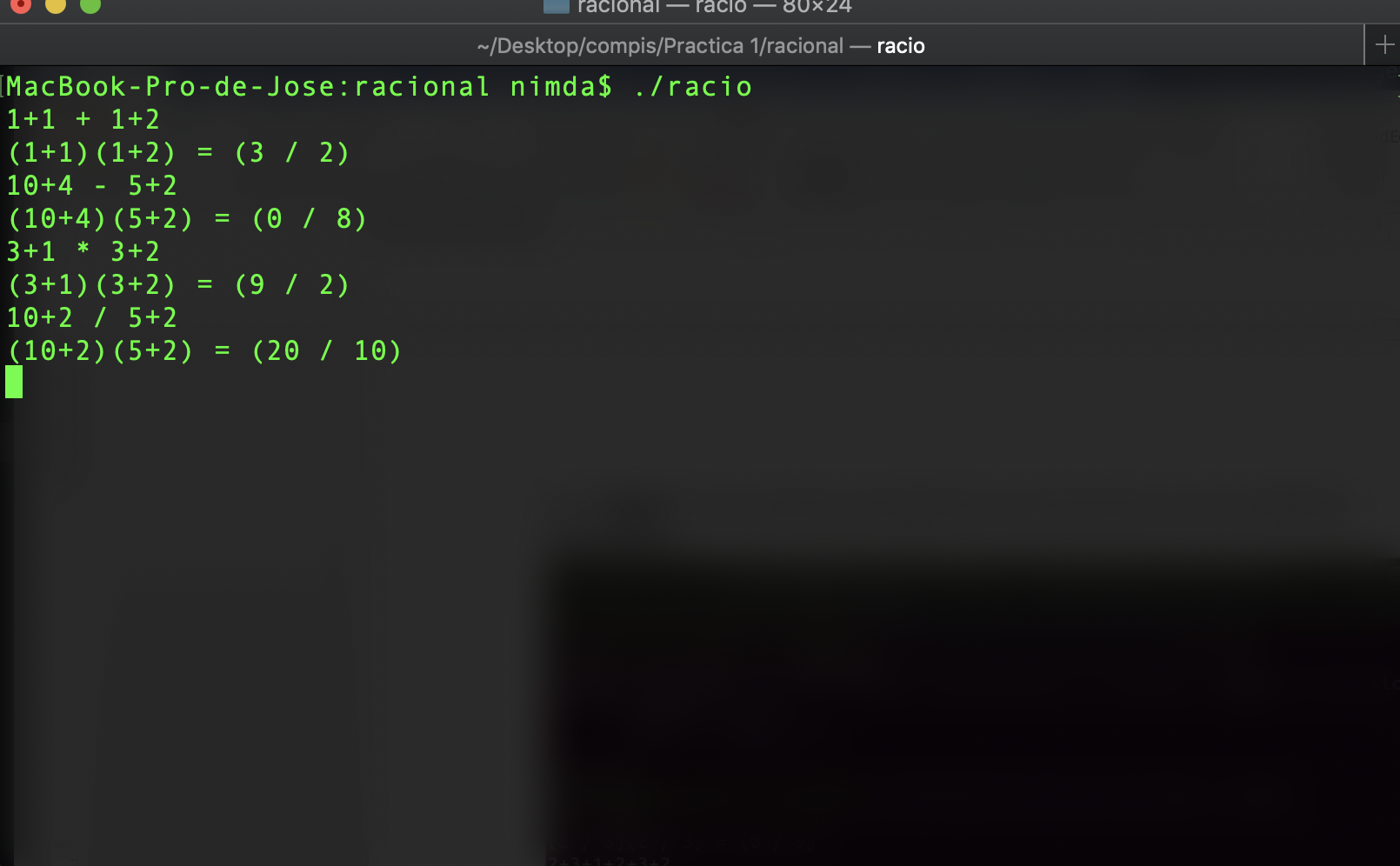


Figura 1 Ejecución del programa

**Conclusión**

Es posible crear una calculadora básica generando tu propia gramática, llevando a la practica la teoría vista en clase, los resultados obtenidos por la calculadora fueron exitosos, como es posible observar en la **figura 1.**

Se hicieron modificaciones en los archivos **racional\_calc.c**, **racional\_calc.h, hoc1.y** y **racional\_calc.l** para obtener un buen resultado en la practica. Finalmente se obtuvo una calculadora básica de números racionales con ayuda de flex y yacc.