ERC

SOLUTIONS

Soluciones de Software

Informe de proyecto final

A continuación, se presenta todo lo relacionado con el proyecto final de la materia de compiladores y lenguajes.

ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

COMPILADORES Y LENGUAJES

PROYECTO 2do Bim

CALCULADORA PARA VARIOS SISTEMAS DE NUMERACIÓN

INTEGRANTES:

ERIKA ANRRANGO

RICHARD MIRANDA

CHRISTIAN SATAMA

Contenido

Acerca de

El presente proyecto pretende mostrar los resultados de todos los conocimientos adquiridos durante el curso Compiladores y Lenguajes del semestre 2020-B de la Escuela Politécnica Nacional a manera de una aplicación de consola usando Lex, Yacc.

El propósito de dicha aplicación es resolver operaciones con números en uno o varios de los siguientes sistemas de numeración: decimal, binario, romano, octal, hexadecimal.

Objetivos

1. Diseñar e implementar expresiones regulares capaces de reconocer cadenas de números en diferentes bases numéricas.
2. Realizar operaciones entre números provenientes de varios sistemas de numeración.
3. Comprender la fase de compilación: análisis léxico, con ayuda de la herramienta LEX.
4. Ejecutar el código fuente en el analizador léxico LEX y que no presente errores, para posteriormente usar el compilador del ambiente Linux.

Sistemas de numeración

Dentro de las ciencias de la computación, uno de los aspectos más importantes radica en la representación de la información. Dentro de la representación de los números, existen varios sistemas de numeración vigentes, de los cuales se utilizan en el presente proyecto los siguientes:

Decimal

Es el sistema de numeración que se usa diariamente entre las personas y en las ciencias.
Como características principales, contiene diez elementos o dígitos y es posicional.

Dígitos = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Binario

Es el sistema de numeración por excelencia dentro de las ciencias de la computación, tiene reglas similares al sistema decimal.
Como característica principal, contiene dos elementos o dígitos.

Dígitos = {0, 1}

Romano

Es el sistema de numeración más antiguo de entre los utilizados en el proyecto y de los más conocidos.
Está basado en el sistema de numeración etrusco, comenzó siendo aditivo, pero llegó a incluir la resta y la multiplicación.
Como característica principal, contiene siete elementos o dígitos que, comparándolos con los caracteres alfanuméricos usados diariamente vienen siendo letras en vez de números.

Dígitos = {I=1, V=5, X=10, L=50, C=100, D=500, M=1000}

Octal

Es un sistema de numeración utilizado también dentro de las ciencias de la computación, es posicional que comparte muchas características con el sistema decimal.
Como característica principal, contiene siete elementos o dígitos.

Dígitos = {0, 1, 2, 3, 4, 5, 6, 7}

Hexadecimal

Es un sistema de numeración igual de importante que el sistema octal.
Como características principales, es un sistema posicional y contiene dieciséis elementos o dígitos, los cuales son los diez que provee el sistema decimal y se complementa con las seis primeras letras del alfabeto español/inglés.

Dígitos = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

Analizadores

Aunque no se trata directamente de analizadores, más bien generadores de analizadores, se utilizó lex, yacc para el desarrollo del proyecto.

Lex

Es un programa para generar analizadores léxicos, originialmente escrito por Eric Schmidt y Mike Lesk. Es el analizador standard en sistemas Unix, tomando como entrada una especificación de analizador léxico y devolviendo el código fuente.

Su estructura es similar a la de un archivo yacc, del cual se tratará en el siguiente punto. Sus principales secciones son las siguientes, separadas entre sí mediante dos símbolos “%%”.

```
Sección de declaraciones
%%
Sección de reglas
%%
Sección de código en C
```

Sección de declaraciones: es el lugar para definir macros y para importar los archivos de cabecera escritos en C.

Sección de reglas, es la más importante; asocia patrones o expresiones regulares a sentencias de C.

Sección de código en C, contiene sentencias en C y funciones que serán copiadas en el archivo fuente generado.

Yacc

Es un programa para generar analizadores sintácticos, desarrollado por Stephen C. Johnson de AT&T para Unix, sus siglas vienen de las iniciales que significan “Otro generador de compiladores más”.
Comprueba que la estructura del código fuente se ajuste a la especificación sintáctica del lenguaje, se basa en una gramática analítica escrita en notación similar a BNF.

Herramientas y conocimientos técnicos aplicados

Para el correcto desarrollo del proyecto, y gracias al curso, se alcanzó gran cantidad del conocimiento, mismo que se detallará a continuación y se verá plasmado en el proyecto en su versión final.

Sistema operativo Unix-Linux

Unix-Linux es un sistema operativo de software libre, está diseñado para aprovechar al máximo las capacidades de cualquier ordenador y que cuenta con características muy interesantes como el multiprocesamiento, multitarea y multiusuario.

Su origen se remonta hasta los años 80, cuando Richard Stallman, inició el Proyecto GNU con el propósito de crear un sistema operativo similar y compatible con UNIX. En el año 1985, se creó la Fundación del Software Libre y se desarrolló la Licencia pública general de GNU para tener un marco legal que permitiera difundir libremente este software.

Posteriormente, en 1991, fue Linus Torvalds, un estudiante de informática de 23 años de la Universidad de Helsinki, quien propone hacer un sistema operativo que se comporte como UNIX pero que, además, funcione sobre cualquier ordenador.

Para el correcto desenvolvimiento del curso y desarrollo del proyecto, ya que se usa principalmente lex y yacc, se requirió un sistema operativo basado en debian, debido principalmente a su intérprete de órdenes (bash) y su compilador de C.

Fig 1. Sistema operativo Unix-Linux, distribución debian

Github

Para el desarrollo del proyecto, trabajar en grupo y la dificultad para realizar reuniones presencialmente, fue necesario utilizar un gestor de versionamiento, optando por GitHub.

Esto facilitó en gran medida el trabajo colaborativo, ya que, al momento de realizar algún cambio, se actualizaba la información y estaba disponible para los demás miembros, además de notificar el usuario y el cambio realizado a detalle dentro de la página oficial.

Fig 2. GitHub, gestor de versionamiento

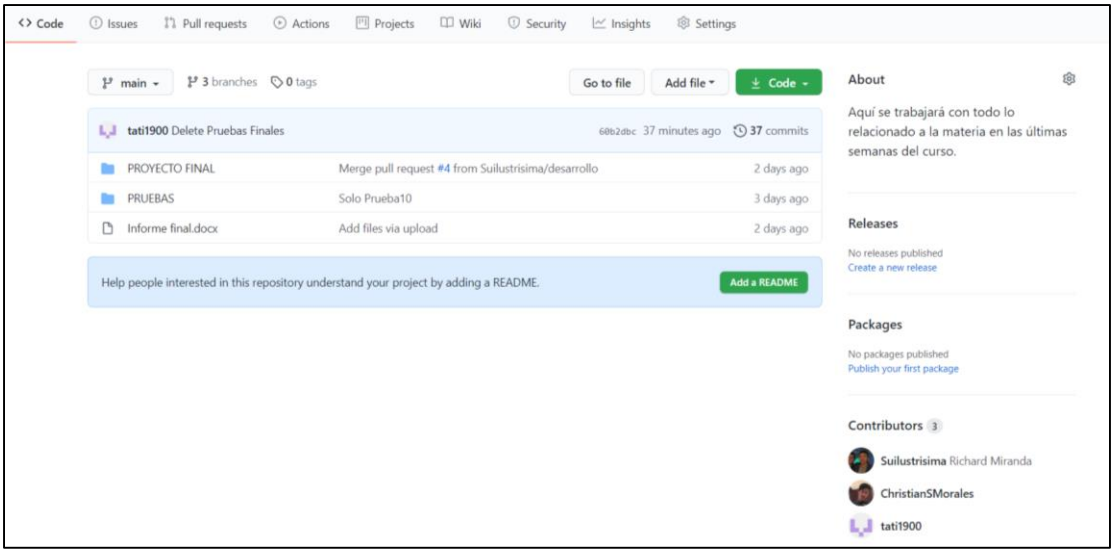

Fig 3. Trabajo colaborativo en GitHub.

Lenguajes

Lenguaje compilado

De manera general, es un lenguaje de programación cuya implementación es normalmente un compilador y no un intérprete.

Lenguaje de máquina

Es un lenguaje que trabaja sin símbolos y hace referencia a las direcciones reales de memoria y los códigos binarios de las instrucciones. Este lenguaje interactúa directamente con el hardware y constituye el nivel más bajo de programación.

Lenguaje de alto nivel

Los lenguajes de alto nivel o lenguajes de tercera generación fueron los más usados durante las décadas pasadas para el desarrollo de programas o softwares de aplicación. Ejemplos de estos lenguajes son el COBOL, PASCAL, BASIC y C entre otros. Su principal característica es que una instrucción codificada en estos lenguajes equivale a varias instrucciones en lenguaje máquina. Los programas escritos en estos lenguajes requieren ser traducidos (compilados) a lenguaje máquina.

Compilador

Es un programa cuya función es traducir o compilar un programa fuente
escrito en algún lenguaje de alto nivel a lenguaje máquina, normalmente es guardado en memoria secundaria en forma ejecutable y es cargado a memoria principal cada vez que requiera ser ejecutado.

Intérprete

Al igual que el compilador, el intérprete traduce un programa fuente escrito en algún lenguaje de alto nivel, pero con la diferencia de que cada instrucción es ejecutada inmediatamente, sin generar un programa en lenguaje de máquina.

Etapas de compilación

Una instrucción desde su escritura hasta su ejecución pasa por varias etapas dentro de la compilación, las cuales son:

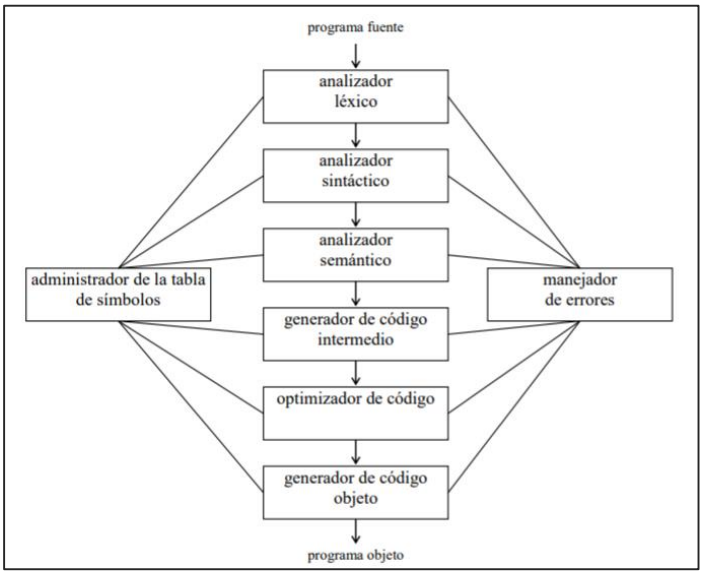

Fig 4. Etapas de compilación

Preprocesamiento:
Transformaciones al Archivo Fuente, previas a la Compilación.

Análisis Léxico:
Reconocimiento de los Elementos del Lenguaje.
Aquí, la cadena de caracteres del programa fuente es leída de carácter en carácter, con el fin de identificar y agrupar los componentes léxicos, que son caracteres del alfabeto y cadenas con significato en conjunto, también conocidos como tokens y por último también se ignoran los comentarios.
Una vez terminada esta etapa, mediante autómatas finitos se valida la correcta conformación de los tokens.

Análisis Sintáctico:
Reconocimiento de la Estructura del Lenguaje.
Las combinaciones de los tokens identificados en la etapa anterior permiten obtener secuencias del programa fuente, lo que hace necesario comprobar que dichas sentencias sean correctas sintácticamente.
Este análisis se lo logra gracias al uso de gramáticas y la construcción de un árbol de derivación.

Análisis Semántico:
Reconocimiento de la coherencia de la Entrada.
En esta etapa se revisa el programa fuente, con el fin de reunir información sobre los tipos de variables que se utilizarán posteriormente al generar código intermedio.
Además, se identifican eventuales errores semánticos, mediante la detección y comunicación de numerosos errores.

Generación de Código Intermedio:
Transformación de la Entrada en una representación de código intermedio para una máquina abstracta.

Optimización de Código:
Mejoras a la representación intermedia que resulten en un código más
rápido de ejecutar.

Generación de Código:
Transformación del código intermedio en código objeto.

Depuración:
Reconocimiento de Errores.

Administración de la Tabla de Símbolos:
Reconocimiento de los nombres de los identificadores utilizados en la entrada y sus diferentes atributos.

Derivaciones en gramáticas:

Se refiere a las maneras de realizar un análisis sintáctico de forma manual y de manera descendente. Formando un árbol.
Comenzando con una cadena con solamente el símbolo inicial y mediante la aplicación sucesiva de las reglas respectivas, es decir encontrando la parte izquierda de la regla y sustituirla por la parte derecha, se llega a una cadena solamente de terminales.

Derivación más izquierda:
Se expanden los VNt de más a la izquierda de la secuencia.

Derivación más derecha:
Se expanden los VNt de más a la erecha de la secuencia.

Derivación indistinta:
Se expanden los VNt de más a la izquierda o derecha de la secuencia.

Aplicación de lex y yacc

Gracias al uso conjunto de estos generadores, es posible crear traductores. Por una parte, lex es capaz de reconocer expresiones regulares o tokens, por otra parte, yacc usa dichos tokens, los empata con gramáticas libres de contexto.

Esto junto al uso de la terminal que ofrece Unix-Linux, permite, mediante comandos, compilar un traductor, de la siguiente manera:

A partir de texto plano con instrucciones de lo que se va a implementar, se genera un programa fuente, mismo que tiene extensión ".l", para las menciones pertinentes, se tomará como nombre del archivo "proyecto".

Mediante el comando por terminal “flex proyecto.l”. En este paso el código lex reconoce las expresiones en una cadena de entrada y la divide en cadenas que coinciden con dichas expresiones. AL final se genera el archivo con nombre “lex.yy.c”, que contiene el autómata.

Mediante el comando por terminal “gcc lex.yy.c –lfl | cc lex.yy.c -lfl” se crea el analizador léxico de nombre “a.out”, que transforma una entrada en una secuencia de tokens.

Mediante el comando por terminal “./a.out” se toma el fichero de entrada y como salida se obtienen acciones.

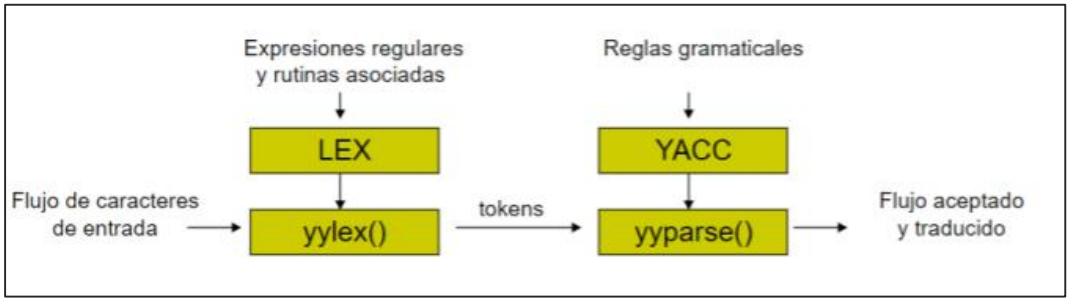

Fig 5. Aplicación de lex y yacc

Expresiones regulares

Dentro de las ciencias de la computación, una expresión regular es un patrón usado para hacer coincidir combinaciones de caracteres con cadenas ellos.
Una expresión regular está compuesta por caracteres simples y símbolos de apertura y cierre, con lo cual se crea reglas que debe cumplir un string, limitándolo. A continuación, se muestra una tabla con las reglas para la construcción de una expresión regular.

c=caracter, x,y=expresiones regulares, m,n=enteros, i=identificador	
.	Coincide con cualquier carácter, excepto nueva línea
*	Coincide con 0 o más instancias de la expresión regular anterior
+	Coincide con 1 o más instancias de la expresión regular anterior
?	Coincide con 0 o 1 de la expresión regular anterior
|	Coincide con la expresión regular anterior o siguiente
[ ]	Define una clase de caracteres
( )	Agrupa la expresión regular encerrada en una nueva expresión regular
"..."	Coincide con todo dentro de "..." literalmente
x|y	x or y
{i}	Definición de i
x/y	x, solo si va seguido de y (y no se elimina de la entrada)
x{m,n}	m a n ocurrencias de x
^x	x, pero solo al principio de la línea
x$	x, pero solo al final de la línea
"s"	Exactamente lo que está entre las comillas (excepto "\" y el siguiente carácter)

Fig 6. Ejemplos de expresión regular

Funcionamiento del proyecto

El proyecto fue desarrollado de manera que reciba como parámetro de entrada una secuencia de caracteres ingresada por teclado, por lo cual es posible que además de caracteres alfanuméricos aparezcan caracteres especiales.

Mediante el uso de expresiones regulares se definió los diferentes sistemas de numeración utilizados y excluir de una operación caracteres que no correspondan, no resolviendo dicha operación.

Tomando en cuenta la sugerencia del profesor, para reconocer un número en su sistema de numeración se debe anteponer la letra inicial del mismo, exceptuando los números romanos que se los escribe directamente.

La calculadora toma las entradas y las transforma a todas a sistema de numeración decimal, una vez de esa manera es posible operarlos.

Las operaciones que admite la calculadora son la suma, resta, multiplicación, división, potenciación, radicación y funciones trigonométricas principales.

Cuando se haya obtenido una respuesta, esta es transformada de nuevo a los diferentes sistemas para mostrar el resultado al usuario.

Código del proyecto

```
%option noyywrap
%{
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>

int obtenerValor(char numeroR);
int convertirRomanoAdecimal(char numeroR[]);

int obtenerValorH(char numeroH);
```

```
int obtenerValorB(char numeroB);

int obtenerValorO(char numeroO);


%}

SIGNOPositivo ("--"|"----"|"------")*
BINARIOPositivo (SIGNOPositivo)*(b|B)[0|1]+
OCTALPositivo (SIGNOPositivo)*(o|O)[0-5]{0,1}[0-6]{0,1}[0-7]{0,1}[0]*
DECIMALPositivo (SIGNOPositivo)*[1-9]{1}[0-9]*
ROMANOPositivo
        (SIGNOPositivo)*(M|m){0,3}((CM|cm)|(CD|cd)|(D|d)?(C|c){0,3})((XC|xc)|XL(xl)|(L|l)?(X|x){0,
3})((IX|xi)|(IV|iv)|(V|v)?(I|i){0,3})
HEXADECIMALPositivo (SIGNOPositivo)*[h|H]([0-9A-F]|[A-F0-9]|[0-9a-f]|[a-f0-9])+

BINARIONegativo "-"("+"|"-+-"|"--+"|"+--"|"----")*(b|B)[0|1]+
OCTALNegativo "-"("+"|"-+-"|"--+"|"+--"|"----")*(o|O)[0-5]{0,1}[0-6]{0,1}[0-7]{0,1}[0]*
DECIMALNegativo "-"("+"|"-+-"|"--+"|"+--"|"----")*[1-9]{1}[0-9]*
ROMANONegativo      "-"("+"|"-+-"|"--+"|"+--"|"----
")*(M|m){0,3}((CM|cm)|(CD|cd)|(D|d)?(C|c){0,3})((XC|xc)|XL(xl)|(L|l)?(X|x){0,3})((IX|xi)|(IV|iv)|(V|v)
?(I|i){0,3})
HEXADECIMALNegativo "-"("+"|"-+-"|"--+"|"+--"|"----")*[h|H]([0-9A-F]|[A-F0-9]|[0-9a-f]|[a-f0-9])+

ABRIR ("["|"("|"{"|<){0,20}
CERRAR ("]"|")"|"}"|">"){0,20}
POTENCIA (p|P)("OCTAL"|"HEXADECIMAL"|"DECIMAL"|"BINARIO"|"ROMANO"){0,1}
RAIZ    (r|R)("OCTAL"|"HEXADECIMAL"|"DECIMAL"|"BINARIO"|"ROMANO"){0,1}
SUMA  ("+")*("OCTAL"|"HEXADECIMAL"|"DECIMAL"|"BINARIO"|"ROMANO"){0,1}("+")*
RESTA("-")*("OCTAL"|"HEXADECIMAL"|"DECIMAL"|"BINARIO"|"ROMANO"){0,1}("-")*
DIVISION
        ("/")*("OCTAL"|"HEXADECIMAL"|"DECIMAL"|"BINARIO"|"ROMANO"){0,1}("/")*
PRODUCTO
        ("*")*("OCTAL"|"HEXADECIMAL"|"DECIMAL"|"BINARIO"|"ROMANO"){0,1}("*")*
```

```
OTHERS
DECIMAL(*SKIP)(*FAIL)|ROMANO(*SKIP)(*FAIL)|HEXADECIMAL(*SKIP)(*FAIL)|BINARIO(*
SKIP)(*FAIL)|OCTAL(*SKIP)(*FAIL).

%%
{DECIMALPositivo} {
        printf("\nNumero decimal positivo: %s\n",yytext);
}
{ROMANOPositivo} {
        printf("\nNumero Romano positivo: %s \n",yytext);
        printf("El numero en decimal positivo es: %d \n",convertirRomanoAdecimal(yytext));
        }

{HEXADECIMALPositivo} {
        printf("\nNumero Hexadecimal positivo: %s \n",yytext);
        }
{BINARIOPositivo} {
        printf("\nNumero Binario positivo: %s \n",yytext);
        }
{OCTALPositivo} {
        printf("\nNumero Octal positivo: %s \n",yytext);
        }
{DECIMALNegativo} {
        printf("\nNumero decimal negativo: %s\n",yytext);
        }
{ROMANONegativo} {
        printf("\nNumero Romano negativo: %s \n",yytext);
        printf("El numero en decimal negativo es: %d \n",convertirRomanoAdecimal(yytext));
        }
{HEXADECIMALNegativo} {
        printf("\nNumero Hexadecimal negativo: %s \n",yytext);
        }
{BINARIONegativo} {
        printf("\nNumero Binario negativo: %s \n",yytext);
        }
{OCTALNegativo} {
        printf("\nNumero Octal negativo: %s \n",yytext);
```

```
	}
{ABRIR} {
	printf("\nSímbolo de apertura: %s \n",yytext);
	}
{CERRAR} {
	printf("\nSímbolo de cierre: %s \n",yytext);
	}
{SUMA} {
	printf("\nOperando, suma: %s \n",yytext);
	}
{RESTA} {
	printf("\nOperando, resta: %s \n",yytext);
	}
{PRODUCTO} {
	printf("\nOperando, producto: %s \n",yytext);
	}
{DIVISION} {
	printf("\nOperando, división: %s \n",yytext);
	}
{RAIZ} {
	printf("\nOperando, raíz: %s \n",yytext);
	}
{POTENCIA} {
	printf("\nOperando, potencia: %s \n",yytext);
	}
.  {printf("Este simbolo no forma parte de una expresion matematica: %s\n",yytext);}
%%
int  obtenerValor(char numeroR){
	switch(numeroR){
		case 'I':
			return 1;
		break;
		case 'i':
			return 1;
		break;
		case 'v':
			return 5;
```

```
                break;
                case'V':
                        return 5;
                break;
                case'x':
                        return 10;
                break;
                case 'X':
                        return 10;
                break;
                case 'l':
                        return 50;
                break;
                case 'L':
                        return 50;
                break;
                case 'c':
                        return 100;
                break;
                case 'C':
                        return 100;
                break;
                case'd':
                        return 500;
                break;
                case 'D':
                        return 500;
                break;
                case 'm':
                        return 1000;
                break;
                case 'M':
                        return 1000;
                break;
        }
}
```

```
int  obtenerValorH(char numeroH){
if (numeroH>=0)
return numeroH;
}

int  obtenerValorB(char numeroB){
        printf("Si es binario");
}

int  obtenerValorO(char numeroO){
if (numeroO>=0)
return numeroO;
}

int convertirRomanoAdecimal(char numeroR[]){
        int i = 0, resultado = 0;
  while(numeroR[ i ])
  {

    if( (strlen(numeroR)-i) > 2)
    {
        //se retorna -1 si el primer digito romano es mayor que el segundo
      if(obtenerValor(numeroR[ i ]) < obtenerValor(numeroR[i + 2]))
      {
              return -1;
      }
    }
        //si el primero digito romano es mayor o igual, es valido
    if(obtenerValor(numeroR[ i ]) >= obtenerValor(numeroR[i + 1]))
      resultado = resultado + obtenerValor( numeroR[i] );
    else
    {
      resultado = resultado + (obtenerValor(numeroR[i + 1])-obtenerValor(numeroR[ i ]));
      i++;
    }
```

```
    i++;
  }
  return resultado;
}

int main(){
        int convertirRomanoAdecimal(char numeroR[]);
        yylex();
        return 0;
}
```

Imágenes con su funcionamiento

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
#include <math.h>
#include <string.h>
int obtenerValor(char numeroR);
int convertirRomanoAdecimal(char numeroR[]);
double convertirRomanoAdecimalFINAL(char numeroR[]);
void decimalesAromanos(int numero,char* buffer);
int nlines = 0;
char * nRomano[20];
//decimalesAromanos(TKN_NUM,nRomano);
%}
ROMANO  (M|m){0,3}((CM|cm)|(CD|cd)|(D|d)?(C|c){0,3})((XC|xc)|XL(xl)|(L|l)?(X|x){0,3})((IX|xi)|(IV|iv)|(V|v)?(I|i){0,3})
DIGITO  [0-9]
ID      [a-zA-Z][a-zA-Z0-9]*
%%

{DIGITO}+("."{DIGITO}+)?        {yylval.real=atof(yytext); return (TKN_NUM);}
{ROMANO}("."{ROMANO})?  {yylval.real=convertirRomanoAdecimalFINAL(yytext); return (TKN_NUM);}
"="                             {return (TKN_ASIGNACION);}
";"                             {return (TKN_PTOCOMA);}
"*"                             {return (TKN_MULT);}
"sqrt"                          {return (TKN_SQRT);}
"cbrt"                          {return (TKN_CBRT);}
"/"                             {return (TKN_DIV);}
"+"                             {return (TKN_MAS);}
"-"                             {return (TKN_MENOS);}
"("                             {return (TKN_PARENTESISA);}
")"                             {return (TKN_PARENTESISC);}
"cos"                           {return (TKN_COS);}
"sen"                           {return (TKN_SEN);}
"^"                             {return (TKN_POTENCIA);}
"e"                             {return (TKN_EULER);}
"ln"                            {return (TKN_LN);}
```

Fig 7. Código del proyecto, parte inicial

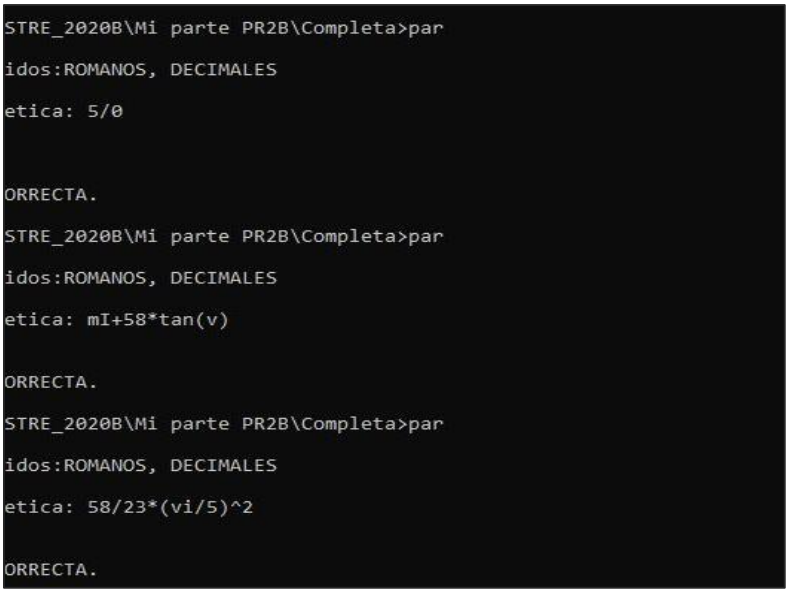

Fig 8. Compilación del proyecto

Conclusiones

- Gracias al desarrollo del presente proyecto se reforzó y puso en práctica los conocimientos impartidos durante el curso de Compiladores y lenguajes.
- El proyecto nos ha dejado muchas enseñanzas importantes sobre las cuales reflexionar.
- Durante el desarrollo nos fuimos percatando de varios puntos que no os consideramos en un principio, sin embargo y aunque nos tomó tiempo, hemos logrado sobrellevarlas.

Bibliografía y referencias:

Apuntes Introducción a la Informática. Capitulo 5, sistemas de numeración. 2010/11. Universidad de Murcia (España). Rafael Barzanallana
https://www.um.es/docencia/barzana/II/Ii05.html

Sistema de Numeración Decimal | Matemáticas
http://www.bartolomecossio.com/MATEMATICAS/sistema_de_numeracin_decimal.html

El sistema de numeración Binario
https://relopezbriega.github.io/blog/2019/03/09/el-sistema-de-numeracion-binario/

Sistema de numeración romano: reglas y ejercicios resueltos
https://www.matesfacil.com/ESO/sistemas-numeracion/sistema-romano/sistema-numeracion-romano-alfabeto-teoria-ejemplos-ejercicios-resueltos-numeros-cambio.html

Sistema de numeración octal: cambio de base 8 a base 10 y viceversa. Método y ejercicios resueltos
https://www.matesfacil.com/ESO/sistemas-numeracion/base-octal/sistema-numeracion-octal-base-ocho-ejemplos-teoria-propiedades-cambio-base-decimal-ejercicios-resueltos.html

https://www.matesfacil.com/ESO/sistemas-numeracion/base-hexadecimal/sistema-numeracion-hexadecimal-base-16-ejemplos-teoria-propiedades-cambio-base-decimal-ejercicios-resueltos.html

Lex (informática)
https://es.wikipedia.org/wiki/Lex_(informática)#:~:text=Lex%20es%20un%20programa%20para,utiliza%20para%20generar%20análisis%20sintáctico

Yacc
https://es.wikipedia.org/wiki/Yacc#:~:text=Yacc%20es%20un%20programa%20para,Otro%20generador%20de%20compiladores%20más

Todo sobre Linux, el sistema operativo de código abierto
https://www.adslzone.net/reportajes/software/que-es-linux/

Regular expressions - JavaScript | MDN
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

Compiladores y Lenguajes, notas de clase para la materia Compiladores y Lenguajes, Fis, EPN, 2020-B