Objetivos generales

- Identificar las diferentes fases que necesita realizar un compilador
- Que el estudiante observe cómo los conceptos de compiladores se utilizarán en variadas aplicaciones
- Conocimiento de sintaxis básica de diferentes lenguajes y conceptos de lenguaje ensamblador

Objectivos específicos

- Demostrar en forma práctica todas las fases de un compilador
- Identificar el trabajo de la tabla de símbolos en el proceso de la compilación y generación de código de 3 direcciones.
- Diseñar un generador de código de 3 direcciones que traduzca un AST en código intermedio de tres direcciones.
- Implementar un traductor capaz de convertir el código de tres direcciones producido por el compilador en código válido tanto en lenguaje C como en ensamblador.

Descripción de la actividad

El objetivo del presente proyecto es poner en práctica todas las fases de la compilación a través de la creación de un Compilador Multilenguaje y multi-fase.

El compilador debe constar de una interfaz gráfica similar a un IDE, y, usando herramientas como JFlex y Cup se deben implementar las fases de análisis léxico, sintáctico y semántico, así como la generación de código intermedio usando cuadruplas.

Una vez generadas las cuadruplas,, se debe generar código tres direcciones con sintaxis de C. Luego se aplicaran optimizaciones sobre el código intermedio y se representará dicho código con sintaxis de C. Por último se trasladará el código intermedio optimizado a lenguaje ensamblador. De este modo se podrá observar cómo funciona un compilador en su totalidad.

IDE

La aplicación servirá como un IDE donde se pueden crear nuevos proyectos, agregar archivos y se tendrán disponibles todas las funcionalidades de un IDE, incluyendo un editor de código con coloreado para las distintas palabras del lenguaje.

Menús

Archivo

~~**Nuevo proyecto**~~
~~El IDE debe permitir la creación de proyectos donde se organizan los archivos con el código a compilar. Esta opción permite al usuario crear un proyecto nuevo que será almacenado usando una estructura xml.~~

~~**Nuevo archivo**~~
~~Permite agregar un archivo nuevo al proyecto, se debe indicar el paquete al que se agrega el archivo.~~

~~**Abrir proyecto**~~
~~Esta opción permite abrir un proyecto. La interfaz de usuario debe mostrar el proyecto usando una estructura de árbol donde se muestran los paquetes y archivos del proyecto.~~

Abrir archivo

Usando el área de despliegue de proyecto, el usuario puede seleccionar un archivo .mlg para editarlo, éste contiene el código fuente de un programa almacenado con anterioridad.

Guardar
Guarda el contenido de la pestaña activa en ese momento.

Guardar Como
Permite guardar el contenido de la pestaña activa en ese momento, con diferente nombre en algún paquete del proyecto.

~~**Cerrar proyecto**~~
~~Cierra el proyecto del IDE.~~

NOTA:
Los archivos .mlg son los archivos que contienen el código fuente del programa a compilar. Entiéndase los archivos de entrada para su compilador.
Las extensiones que se manejan para los archivos con Código en Tres direcciones y Código Optimizado serán .cpp y para el Código en Assembler tendrá que ser .nasm

Generar código

Código en Tres Direcciones
Esta opción genera el código en tres direcciones del proyecto y este resultado deberá ser mostrado en la pestaña Código Tres Direcciones.

Código Optimizado
Esta opción genera el código 3 direcciones optimizado del programa, y será mostrado en la pestaña Optimización.

Código Assembler
Esta opción genera el código Assembler del programa, y se mostrará el código generado en la pestaña Assembler

Ejecutar código

Una vez generado algún tipo de código, el usuario puede indicarle al IDE que cree un archivo ejecutable a partir del código seleccionado. De esta forma el usuario finalmente podrá ejecutar el programa.

NOTA:
El IDE en ningún momento ejecuta el código, solo genera un programa binario con ayuda del compilador de C o Assembler para crear el ejecutable que posteriormente el usuario ejecutará en una terminal.

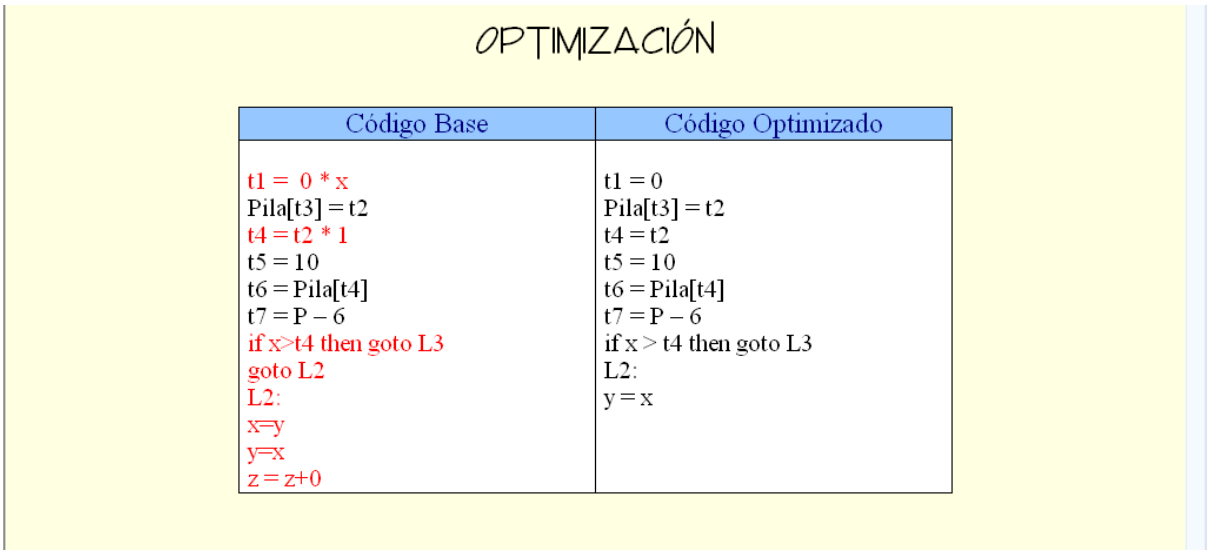
Reporte de Errores

Los errores léxicos, sintácticos y semánticos deberán ser mostrado en la sección de Errores, recuerde que estos errores solo podrán estar presente únicamente en el archivo .mlg. El formato para mostrar los errores es:

Fila - columna - tipo de error - quien lo produjo - breve descripción

Reporte de optimización

Esta opción permite verificar la optimización realizada paso a paso del código en tres direcciones de forma gráfica. Se valorará toda la información que se muestre para el fácil entendimiento de la optimización realizada.



Reporte de Optimización

NOTA:
En el reporte deberá mostrar las líneas que fueron tomadas para la optimización de un color distinto. Cada celda de la tabla será el bloque que usted tomó para la optimización del código. Respetar el formato presentado.

Estructura archivo .mlg

```
%%%PASCAL
   Funciones y Procedimientos en Pascal
%%%JAVA
   Clases en Java
%%%PROGRAMA
Sección de librerías de C     //opcionales
#include "PASCAL"             //incluye todo el código en Pascal
#include "PASCAL.nombre_suprograma"//incluye el código en Pascal
definido para un subprograma
#include "JAVA.*"       //incluye todas las clases declaradas en
                        la sección de Java
#include "JAVA.nombre"  //incluye   una   clase   en   especifico
                        //declarada en la sección de Java

// no hay orden en las instrucciones de la sección de librerias

Sección declaración de Constantes
Sección de Variables Globales

void main()
{
 // Programa principal
}
```

Los lenguajes que se utilizarán para conformar la sección de librerías son:

- Pascal
- Java

Mientras que el programa principal utilizará C, para cada sección del programa se deberá respetar la sintaxis de cada lenguaje de programación.
El área del programa principal puede o no contener código, al igual que la sección de pascal y de java.

En los lenguajes Pascal y Java se utilizarán de forma limitada, únicamente:

- Expresiones aritméticas + - * / % ^ (potencia)
- mensajes a pantalla: se usará la palabra **print** y **println** tanto para pascal como en java, y la sintaxis es **print(valores_separados_por_coma)**
- solicitud de datos desde java a usuarios: intinput, floatinput, charinput, ej. vardestino = intinput()
- ciclos (for i, while, do while), incluyendo **continue** y **break**.

- manejo de condiciones (if-elseif-else, switch)
- declaración de variables
- asignaciones variables
- Funciones
- Procedimientos
- Clases (Java) No se incluyen clases abstractas pero sí existirá la posibilidad de extender de una clase y sobreescribir métodos. Tampoco se incluyen interfaces, ni clases anidadas.
- La cadena es considerada como un tipo primitivo.
- La herencia funcionará de la misma manera que en Java (excepto que no existen clases abstractas), pero si es posible sobrescribir métodos.
- La sobrecarga de métodos funciona de la misma manera que en java.
- No se usa la palabra final ni static.
- Es posible recibir objetos mediante parámetros dentro de los métodos y constructores de una clase.
- Cada clase tendrá un constructor por defecto, sin argumentos a menos que se indique lo contrario, tal y como funciona en java, los modificadores de acceso para estos funcionan tal y como lo hacen en java.
- Por simplicidad, los subprogramas de pascal no pueden invocar a otros subprogramas pascal.
- Por simplicidad, en Pascal no se manejan definición de tipos ni records ni subrangos.
- SE DEBE soportar arreglos en todos los lenguajes.

Además los únicos tipos primitivos que se manejan en estos lenguajes son entero, real, carácter, boolean y string.

NOTA:
Para la sintaxis de Pascal, utilizar lo mismo que en el proyecto 1 pero con las limitaciones descritas anteriormente.

El programa principal que se maneja en sintaxis de C se especifica a continuación:

Especificaciones del programa principal

Solo el programa principal es capaz de invocar funciones y clases de pascal y java,

Tipos de datos

Los tipos de datos que se utilizarán son:

- int
- char
- float

- string

NOTA: La variable puede o no ser iniciada al momento de definirla.

Arreglos

Arreglos de N dimensiones de cualquier tamaño. Los arreglos pueden ser de cualquier tipo.

```
Tipo    arreglo[dim1];
Tipo    arreglo[dim1][dim2][dim3];
```

NOTA: Al momento de declarar un arreglo, las dimensiones pueden ser expresadas usando literales enteras, constantes o expresiones aritméticas sobre literales y constantes.
Al momento de usar un arreglo, dentro de las dimensiones podrán venir expresiones aritméticas, arreglos, funciones, constantes, etc. Ejemplo:

```
int arreglo[25+4][CONSTANTE_ENTERA];
x = vector[5*4/7+8*9(4+2)][matriz[1][2*7]];
```

Constantes

Definición de constantes:
const tipo nombre_constante = valor;

```
const float pi = 3.14159;
const char c = 'X';       // X es una constante tipo char
const int X = 10;   // X es un tipo int
```

Operadores

Todos los operadores aritméticos, números enteros y reales ambos pueden ser negativos, los operadores aritméticos pueden ser: +, -, *, /, % (modulo ó residuo) ^, = asignación y paréntesis.
Ejemplo

```
var = ( 1 + 2 – 3 * 4 / 5 ) % (5 * -3);
```

Comentarios

Los comentarios de línea o de bloque que se hagan deberán de ser pasados a código tres direcciones y al código ensamblador.

```
//Comentario de línea
/*Comentario
de bloque*/
```

Sentencia if-else

En la instrucción If, dentro de la condición pueden venir expresiones aritméticas y relacionales (<, >, = =, !=), esto implica operadores && (and), || (or), !(not); **con o sin** Else.

```
if ( condición )
{
	SENTENCIAS;
}
else
{
	SENTENCIAS;
}
```

Donde **condición** puede ser cualquier condición que involucra operadores aritméticos, relacionales y lógicos, y cualquier tipo de variable. Si el valor resultante de la condición es mayor que 0, la condición es VERDADERA, si el resultado de la condición es menor o igual a 0 la condición es FALSA. Ejemplo:

```
 (VarAr[1] + 1 > var2 * 2 && var3 != var4 )
```

Sentencia switch

La instrucción Switch, con *n* case y **con o sin** Default.

```
switch  (variable)
{
	case valor1:
			SENTENCIAS;
			break;
	case valor2:
			SENTENCIAS;
			break;
	….		….
	case valorn:
			SENTENCIAS;
			break;
	default:
			SENTENCIAS;
			break;
}
```

Ciclo for

for (variable = **valor_inicial**; **condición**; incremento|disminucion/accion posterior)

```
{
	SENTENCIAS;
}
```

Valor_inicial será un número entero o una variable tipo entero, y **accionposterior** es la accion que se ejecuta al finalizar una iteracion, **condición** puede ser cualquier condición, refiérase a la parte **condición** de la instrucción if.
La variable de asignación puede ser declarada antes o inicializada directamente en la misma asignación.

Ciclo while

```
while ( condición)
{
        SENTENCIAS;
}
```

Condición puede ser cualquier condición, refiérase a la parte condición de la instrucción **if**

Ciclo do while

```
do
{
        SENTENCIAS;
}
while ( condición );
```

Condición puede ser cualquier condición, refiérase a la parte condición de la instrucción **if.**

Instrucción continue y break

Los ciclos pueden incluir la palabra reservada **continue** cuya función es obviar las instrucciones siguientes e iniciar la siguiente iteración.
La palabra reservada **break** es opcional e indica que se finaliza cualquier ciclo o sentencia de control sin ejecutar el código que se encuentre por debajo de esta palabra.

Las instrucciones **continue** y **break** dependen del contexto por lo que están sujetas a errores semánticos.

Llamadas a funciones y procedimientos

En Pascal puede existir paso de parámetros por valor y paso de parámetros por referencia. En el caso de nuestro código Java, el paso de parámetros se manejan igual que en el lenguaje oficial.

Invocación (activación):

```
Variable  = PASCAL.nombre_funcion(parámetro);
PASCAL.nombre_procedimiento(parámetros);
```

~~Si la firma de una función/procedimiento invocado coincide con varias funciones/procedimientos, entonces se activa la que se encuentra más abajo en la sección de declaraciones de librerías.~~

scanf

Esta instrucción permitirá asignar valores a las variables.

scanf(“ *mascara* “, &variable);

Mascara se refiere a texto y al indicador de que tipo de dato leerá, ejemplo:

```
scanf(“%d”, &var);      //Leerá del teclado un valor para asignar a variable tipo int
scanf(“%c”, &var);      //Leerá del teclado un valor para asignar a variable tipo char
scanf(“%f”, &var);      //Leerá del teclado un valor para asignar a variable tipo float
scanf(“%s”, &var);      //Leerá del teclado un valor para asignar a variable tipo string
```

printf

Esta instrucción permitirá desplegar mensajes y los valores de las variables

printf(“ texto “); ó printf(“El valor es *mascara* “, var);

En la primera instrucción solo se mostrará *texto*, en la segunda se mostrará el texto y el valor que contenga la variable *var*, puede haber más de una variable por mostrar.

Mascara se refiere a texto y al indicador de que tipo de dato desplegará, ejemplo:

```
printf(“Valor %d”, var);//Desplegará el valor de una variable tipo int
printf (“Valor %c”, var);       //Desplegará el valor de una variable tipo char
printf (“Valor %f”, var);//Desplegará el valor de una variable tipo float
```

clrscr

limpiar la pantalla, ejemplo:

clrscr();

getch

Esta instrucción leerá una tecla del teclado para poder seguir la ejecución del programa. El valor que regresa **puede o no** ser asignado a una variable tipo **char o int,** si es asignado a una variable tipo char la variable tendrá el carácter que se presionó; si es asignado a una variable tipo int la variable tendrá el valor ASCII del carácter que se presionó.

x = getch();

Clases

Todas las clases serán públicas, por lo que deben incluir el modificador de visibilidad ‘public’, sin embargo, los miembros pueden incluir alguno de los modificadores ‘public’ ‘protected’ y ‘private’.
Las clases deben soportar herencia simple.

El uso de las clases se maneja de la siguiente manera

Declaración

```
JAVA.Nombre_Clase    Nombre_Var1(parametros), Nombre_Var2(parametros);
JAVA.Nombre_Clase    Nombre_Var1(parámetros); // llamando a constructor
```

Llamada a métodos

```
Variable = JAVA.nombre_objeto.método(parámetros);
JAVA.nombre_objeto.método(parámetros);
```

~~Si el nombre de una clase coincide con varias clases, entonces se usa la que se encuentra mas abajo en la sección de declaraciones de librerías~~

Documentación

La documentación que se deberá entregar será la siguiente:

- Manual de Usuario
- Manual Técnico indicando el funcionamiento de su programa así como las herramientas usadas para el desarrollo del mismo y gramáticas creadas.

Importante

- Uso de herramientas JFlex/cup para análisis léxicos, sintácticos y semántico del lado del backend.
- Las copias obtendrán nota de cero y se notificará a coordinación.
- Si se va a utilizar código de internet, entender la funcionalidad para que se tome como válido.
- Por cada modificación que se le haga al archivo de entrada tendrán una penalización de 2 puntos sobre la nota final del proyecto.
- Al momento de seleccionar la ejecución de un código (3D, optimizado, etc), la aplicación deberá ser capaz de realizar la llamada al compilador y ejecutar el código

(a través de una terminal). No se permitirá realizar las llamadas al compilador de forma manual para compilar el código.
- Los resultados al momento de ejecutar cada uno de los códigos generados deberán ser idénticos al programa con extensión .mlg.
- Mínimo requerido para calificación: Generacion de codigo 3 direcciones y ejecución del mismo para el código Java y C

Entrega

La fecha de entrega es el día 30 de octubre, a las 15:00. Los componentes a entregar por medio de github son:

- Código fuente
- Ejecutable (jar)
- Manual de Usuario
- Manual Técnico indicando el funcionamiento de su programa así como las herramientas usadas para el desarrollo del mismo y gramáticas creadas.

Calificación

Pendiente de definir.