Documento Técnico

Proyecto Compiladores II

Número de Cuenta: 11811146

Nombre: Tiffanny Alexa Varela Banegas

Catedrático: Román Arturo Pineda Soto

Clase: Compiladores II

Sección: 269

Fecha de Entrega: 18/12/2025

Contenido

Introducción

El presente documento describe el diseño e implementación del compilador MiniC desarrollado como proyecto para la asignatura de Compiladores II. El objetivo principal es aplicar de manera practicas los fundamentos que hemos aprendido sobre el proceso de compilación, abarcando desde el análisis léxico y sintáctico hasta la generación de código ensamblador ejecutable.

MiniC es un subconjunto del lenguaje C que conserva las características esenciales del lenguaje original (variables, expresiones aritméticas y lógicas, estructuras de control, funciones y arreglos), pero con una sintaxis simplificada que facilitan su implementación.

En este compilador se ha hecho uso de **ANTLR4** para la construcción del analizador léxico y sintáctico, del cual luego se genera el **AST**. Sobre este AST se realiza el análisis semántico. Luego de la validación del programa fuente, procede a la traducción del AST hacia su **REPRESENTACION INTERMEDIA (IR).**

La fase final consiste en la generación de **código ensamblador MIPS,** junto con el runtime que proporciona las rutinas necesarias para la ejecución del programa. Esto permite obtener programas ejecutables que pueden ser simulados en entornos como **MARS**.

Contenido

Definición formal del lenguaje

Tipos de datos soportados

MiniC define los siguientes tipos:

- Int: números enteros
- Char: caracteres
- Bool: valores booleanos
- String: cadenas de caracteres
- Void: retorno de funciones sin valor
- Arreglos unidimensionales
- Arreglos bidimensionales

Variables y arreglos

- Las variables deben declararse antes de su uso
- No se permite redeclaracion en el mismo ámbito
- Los arreglos pueden ser de una o dos dimensiones
- Los índices deben de ser de tipo int

Ejemplo valido

```
int a[10];

a[3]=5;
```

Ejemplo no valido

```
a[9]=5; //Error: Variable no declarada
```

Funciones

Las funciones se definen con:

- Tipo de retorno
- Nombre
- Lista de parámetros (opcional)
- Cuerpo de la función

Restricciones:

- No puede haber funciones duplicadas
- El tipo de retorno y el valor retornado deben de coincidir

- La función main:
  - Debe existir
  - Debe retornar int
  - No debe recibir parámetros

Ejemplo

```
int suma (int a, int b) {
	return a + b;
}
```

Estructuras de control

Soporta las siguientes estructuras:

- If / else
- While
- Do while
- For
- Return
- Las condiciones de control **deben de ser booleanas**

Ejemplo valido

```
if (x<10) {
	x = x + 1;
}
```

Ejemplo no valido

```
if (y){} //Error si y no es booleana
```

Decisiones de diseño

Uso de Antlr4

Usado para análisis léxico y sintáctico y la generación automática de parser. Este nos permite una separación entre la gramática y la lógica que se usa en el compilador.

Construcción de AST

Se implemento de forma explícita para poder separar la sintaxis concreta del análisis semántico, facilitar la generación de IR y facilitar las futuras extensiones. Cada nodo del AST se implementó con el patrón de Visitor.

Manejo de ámbitos (SymbolTable)

Se diseño una tabla de símbolos jerárquica.

- Cada bloque crea un nuevo scope
- Los scopes internos heredan del padre
- Se permite shadowing controlado

Todo esto evita errores como la redeclaracion

```
int x;

int x; (Dentro del mismo ámbito)
```

Representación de arreglos

Se representan como tipos compuestos en forma de cadena (int[5][5]) ya que esto simplifica la tabla de símbolos, permite reducir dimensiones y tipos base y evita estructuras complejas adicionales. Todo esto nos permite verificar el numero correcto de índices, el tipo de acceso parcial o completo y las asignaciones validas a arreglos.

Verificación de retornos

Esto garantiza que las funciones no void siempre retornen un valor, los returns tengan el tipo correcto y sus cambios condicionales cubran todos los casos.

Representación intermedia (IR)

Genera una Representación Intermedia de tres direcciones, diseñada para ser fácilmente traducible a ensamblador MIPS.

MiniC

```
int x;

x = 3 + 4;
```

IR

```
t1 = 3

t2 = 4

t3 = t1 + t2

x = t3
```

Aquí podemos observar el cómo cada operación tiene como máximo dos operandos lo que facilita optimizaciones futuras y simplifica la traducción al código ensamblador.

Definición de la Representación Intermedia (IR)

Enfoque general

El IR está representado por:

- El AST tipado y validado semánticamente
- El uso de registros temporales virtuales
- La descomposición de expresiones complejas en operaciones elementales
- El uso de labels para control de flujo

Todo esto permite una traducción directa y clara del AST a MIPS.

Modelo IR basado en registros temporales

Utilizamos registros temporales MIPS ($t0 a $t9) como valores intermedios. Cada expresión devuelve el nombre de un registro que contiene su resultado.

```
t0 = load a

t1 = 11

t2 = t0 * t1

store t2 -> b
```

Manejo de memoria de la IR

Variables globales: se representan en la sección **.data** del código generado y se accede mediante etiquetas `_x: .word 0`

- Dirección: `la $t9, _x`
- Carga: `lw $t0, 0($t9)`
- Almacenamiento: `sw $t0, 0($t9)`

Variables locales y stack frame: se manejan mediante offsets relativos al $fp o $sp dependiendo del contexto:

- En main: offsets positivos respecto a $sp
- En otras funciones: offsets negativos respecto a $fp

Representación IR en expresiones

- Aritméticas y lógicas: se traducen de izquierda a derecha

MiniC

```
c = a * 11;
```

IR

```
t0 = load a

t1 = 11

t2 = t0 * t1

store t2 -> c
```

MIPS generado

```
lw $t0, 0($sp)

li $t1, 11

mul $t2, $t0, $t1

sw $t2, 12($sp)
```

- Operadores relacionales y lógicos: los operadores relacionales producen valores booleanos (0 o 1) mediante instrucciones como: slt, sgt, sle, sge, seq, sne, and, or

MiniC

```
a < b
```

IR

```
t2 = (a < b)
```

MIPS generado

```
slt $t2, $t0, $t1
```

Control de flujo

- If / else:

IR

```
if t0 == 0 goto L1

  then_block

goto L2

L1:

  else_block

L2:
```

MIPS generado

```
beqz $t0, L1

...

j L2

L1:

...

L2:
```

- Ciclos: while, for y do-while se implementan usando etiquetas de inicio y fin, manteniendo el flujo explícito del programa

Representación IR de arreglos

- Arreglos unidimensionales: calcula direcciones de memoria usando desplazamientos

IR

```
addr = base + index * 4

value = load addr
```

MIPS

```
sll $t8, index, 2

addu base, base, $t8

lw result, 0(base)
```

- Arreglos bidimensionales: utiliza la formula `offset = (i* columnas +j)*4`

MIPS

```
mul $t9, i, numCols

addu $t9, $t9, j

sll $t9, $t9, 2

addu base, base, $t9
```

Llamadas a funciones

Los primeros cuatro argumentos se pasan en $a0–$a3, luego los argumentos adicionales se pasan por stack, por último, el valor de retorno se coloca en $v0.

IR

```
call f(a, b)

t0 = return value
```

MIPS

```
move $a0, $t0

move $a1, $t1

jal f

move $t2, $v0
```