



**Tecnológico  
de Monterrey**

# **Diseño de compiladores**

**Proyecto N°4**

**Campus Santa Fe**

**Enrique Lira Martínez A01023351**

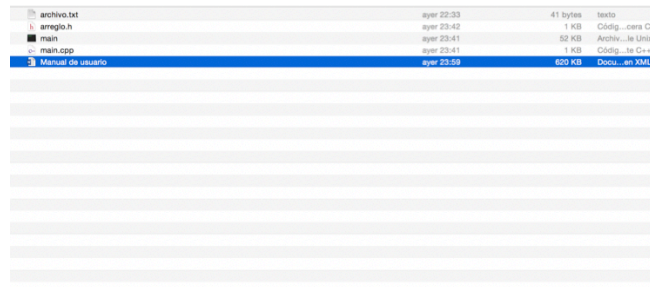
**Profesor: Dr. Víctor Manuel de la Cueva H**

**13 de mayo de 2019**

Para esta materia se decidió crear un compilador en Python para código escrito en C minus , el compilador creado genera código ensamblador entendible para una maquina con el set de instrucciones de la arquitectura MIPS. El código generado se puede correr en una máquina con estas características o bien, en algún simulador, se recomienda usar QtSpim por su compatibilidad y su uso fácil y por ser software abierto.

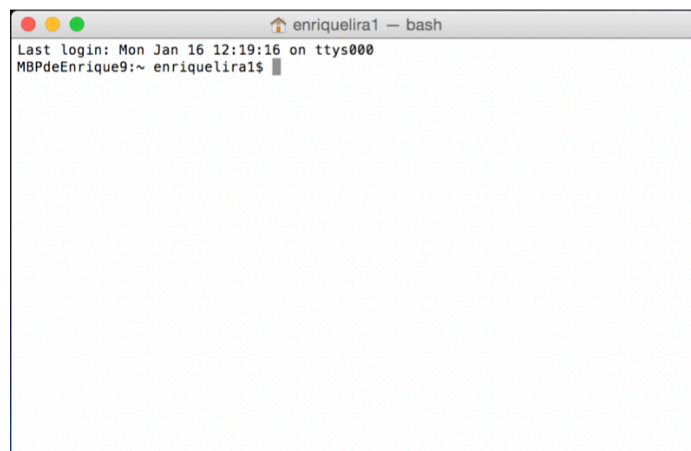
## Manual del usuario

- Al momento de descargar el archivo deberá de ubicarlo en una carpeta o una ubicación que este a la mano (Desktop).



archivo.txt	ayer 22:33	41 bytes	texto
arreglo.h	ayer 23:42	1 KB	Código...era C
main	ayer 23:41	52 KB	Archiv...le Unix
main.cpp	ayer 23:41	1 KB	Código...le C++
Manual del usuario	ayer 23:58	609 KB	Docu...en XML

- A continuación deberá de abrir la terminal(mac) o un IDE para correr programas de Python.

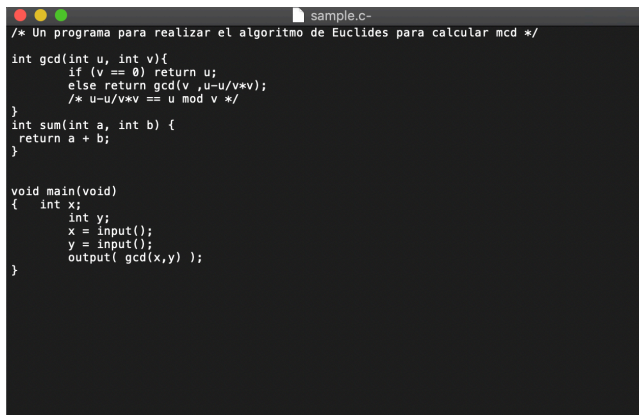


- Si abre la terminal deberá de escribir el comando cd para moverse de carpeta y a la vez deberá de escribir la dirección donde se encuentra el programa.



```
enrique1ira1 ~ bash
MBPdeEnrique9:~ enrique1ira1$ cd Desktop/4Semestre/Datos/tarea1/
```

- Para este programa usted deberá crear un archivo de c minus con un nombre conocido y en este archivo usted deberá contener los datos que sean necesarios para poder correr el programa a continuación esta un ejemplo de al archivo



```
sample.c-
/* Un programa para realizar el algoritmo de Euclides para calcular mcd */

int gcd(int u, int v){
    if (v == 0) return u;
    else return gcd(v, u-u/v*v);
    /* u-u/v*v == u mod v */
}

int sum(int a, int b) {
    return a + b;
}

void main(void)
{
    int x;
    int y;
    x = input();
    y = input();
    output( gcd(x,y) );
}
```

- A continuation se correr el archivo cgen.py el cual generar el codigo compilado en language ensamblador, esto codigo puede imprimir el AST formado en el parser y la semantica generada, como se muestra acontinuacion.

```

program
| declaration-list
| fun-declaration
| int
| ID
| params
| param
| int
| ID
| param
| int
| ID
| compound-stmt
| {
| empty
| statement-list
| statement
| selection-stmt
| if
| expression
| var
| ID
| simple-expression
| additive-expression
| term
| factor
| var
| ID
| ==
| additive-expression
| term
| factor
| num
| statement
| return-stmt
| return
| expression
| var
| ID
| simple-expression
| additive-expression
| term
| factor
| var
| ID
| ;
| else
| statement

```

NIVEL 0  
{'gcd': ['int', []], 'sum': ['int', []], 'main': ['void', []], 'x': 'int', 'y': 'int'}  
NIVEL 1  
{'gcd': ['int', []]}  
NIVEL 1  
{'sum': ['int', []]}  
NIVEL 1  
{'main': ['void', []], 'x': 'int', 'y': 'int'}

Por ultimo se genera un archivo con la extension .s generado por el compilador, acontinuacion un ejemplo.

```

.text
.align 2
.globl main
gcd:
    move $fp, $sp
    sw $ra 0($sp)
    addiu $sp, $sp, -4
    beq $t0, $t1, true_branch
false_branch:
    sw $fp 0($sp)
    addiu $sp, $sp, -4
    sw $fp 0($sp)
    addiu $sp, $sp, -4
    jal gcd
    la $v0, $t3
    lw $fp 0($sp)
    jr $ra
    b end_if
true_branch:
    la $v0, $t3
    lw $fp 0($sp)
    jr $ra
end_if:
    lw $ra 4($sp)
    addiu $sp, $sp, 2
    lw $fp 0($sp)
    jr $ra
sum:
    move $fp, $sp
    sw $ra 0($sp)
    addiu $sp, $sp, -4
    sw $a0 0($sp)
    addiu $sp, $sp, -4
    lw $t1 4($sp)
    add $a0, $t1, $a0
    addiu $sp, $sp, 4
    la $v0, $t3
    lw $fp 0($sp)
    jr $ra
    lw $ra 4($sp)
    addiu $sp, $sp, 2

```

**Expresiones regulares**  $ID = [a - Z]^+ [a - Z]^* [0 - 9]^*$

NUM = [0-9]\*

## DFA



## Gramática utilizada para Parser

- → program -> declaration-list
- → declaration-list -> declaration | declaration declaration-list
- → declaration -> var-declaration | fun-declaration
- → var-declaration -> type-specifier ID ; | type-specifier ID [NUM] ;
- → type-specifier -> int | void
- → fun-declaration -> type-specifier ID (params) compound-stmt
- → param-list -> param param-list'
- → param -> type-specifier ID | type-specifier ID []
- → compount-stmt -> { local-declarations statement-list }
- → local-declarations -> empty | var-declaration local-declarations
- → statement-list -> empty | statement statement-list

- → statement -> expression-stmnt | compound-stmt | selection-stmt | iteration-stmt | return stmt
- → expression-stmnt -> expression ; | ;
- → selection-stmt -> if ( expression ) statement | if ( expression ) statement else statement
- → iteration-stmt -> while ( expression ) statement
- → return-stmt -> return ; | return expression ;
- → expression -> var = expression | simple-expression
- → var -> ID | ID [expression]
- → simple-expression -> additive expression relop additive-expression | additive expression
- → relop-><=|<|>|>=|==|!=
- → additive-expression -> term additive-expression'
- → additive-expression' -> addop term additive-expression' | empty
- → addop->+|-
- → term -> factor | term'
- → term' -> empty | mulop factor term'
- → mulop->\*/
- → factor -> (expression ) | var | call | NUM
- → call -> ID (args)
- → args -> arg-list | empty
- → arg-list -> expression arg-list'

### Reglas lógicas de inferencia de tipos

Valores 1:int 2:int -> int + int -> return int

Valores 1:int 2:int -> int - int -> return int

Valores 1:int 2:int -> int \* int -> return int

Valores 1:int 2:int -> int / int -> return int

Valores T[] -> return int

Valores void -> return int

Valores void -> return int

Valores T -> return T

## Explicación de estructura de tabla de símbolos

Cada nodo del arbol representa un nivel ,el padre y el simbolo en cada tabla de símbolos se encuentran las variables y se coloca el ID como índice ,el tipo y si es un arreglo se pone su tamaño si es una function se agrega el ID como index, el tipo y los paramentro en un arreglo.

Identifier	Type	Values
gcd	int	[int,int]
main	void	void

Identifier	Type	Values
main	void	void
x	Int	
y	Int	

Identifier	Type	Values
gcd	int	[int,int]