



DISEÑO DE UN COMPILADOR

LENGUAJES DE PROGRAMACIÓN

```
1  int main()  
2  {  
3      printf("Hello World!");  
4      return 0;  
5  }  
6  
7
```

```
1  101010010  
2  00101010  
3  101010010  
4  111000111  
5  100100011  
6  100011101
```

GRUPO 2

Participants



1010101
0101000
11100011

ANGEL
FARRÉ

JAIME
PARRA

JOAN
FITÓ

SERGIO
RODRIGO

VICTOR
GARRIDO

Planteamiento

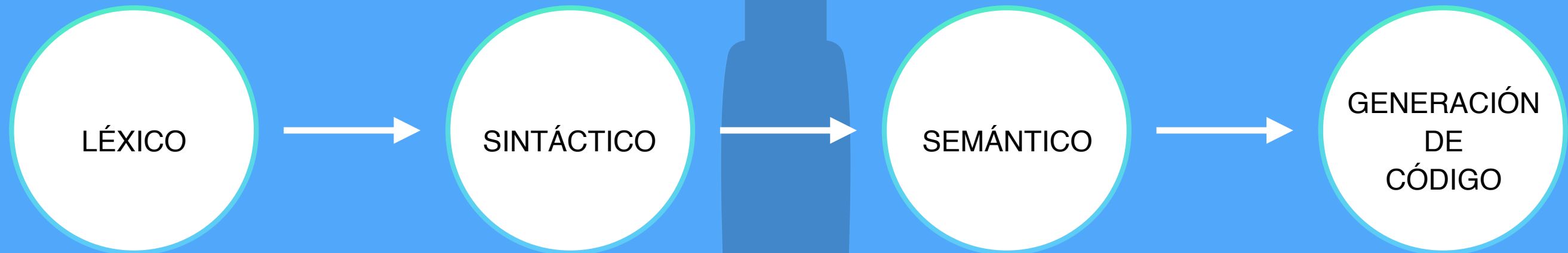


1010101
0101000
11100011

El propósito de esta práctica es desarrollar e implementar un compilador para un lenguaje libre de contexto

Proceso de desarrollo

1010101
0101000
11100011



Léxico



1010101
0101000
11100011

Necesitamos un conjunto de palabras
que formen parte de nuestro lenguaje

Léxico

1010101
0101000
11100011

IDENTIFICADORES

CONSTANTES

OPERADORES
RELACIONALES

TIPOS
PREDEFINIDOS

PALABRAS
CLAVE

SÍMBOLOS
ESPECIALES

Léxico



1010101
0101000
11100011

Para cada lexema de nuestro
lenguaje generaremos un token

Léxico

1010101
0101000
11100011

Token	Lexema
ID	{A-Z, a-z}
Operador	+ -
	=
Type	int

Sintáctico



1010101
0101000
11100011

Necesitamos un conjunto de reglas combinatorias de nuestros lexemas para poder generar expresiones

Sintáctico



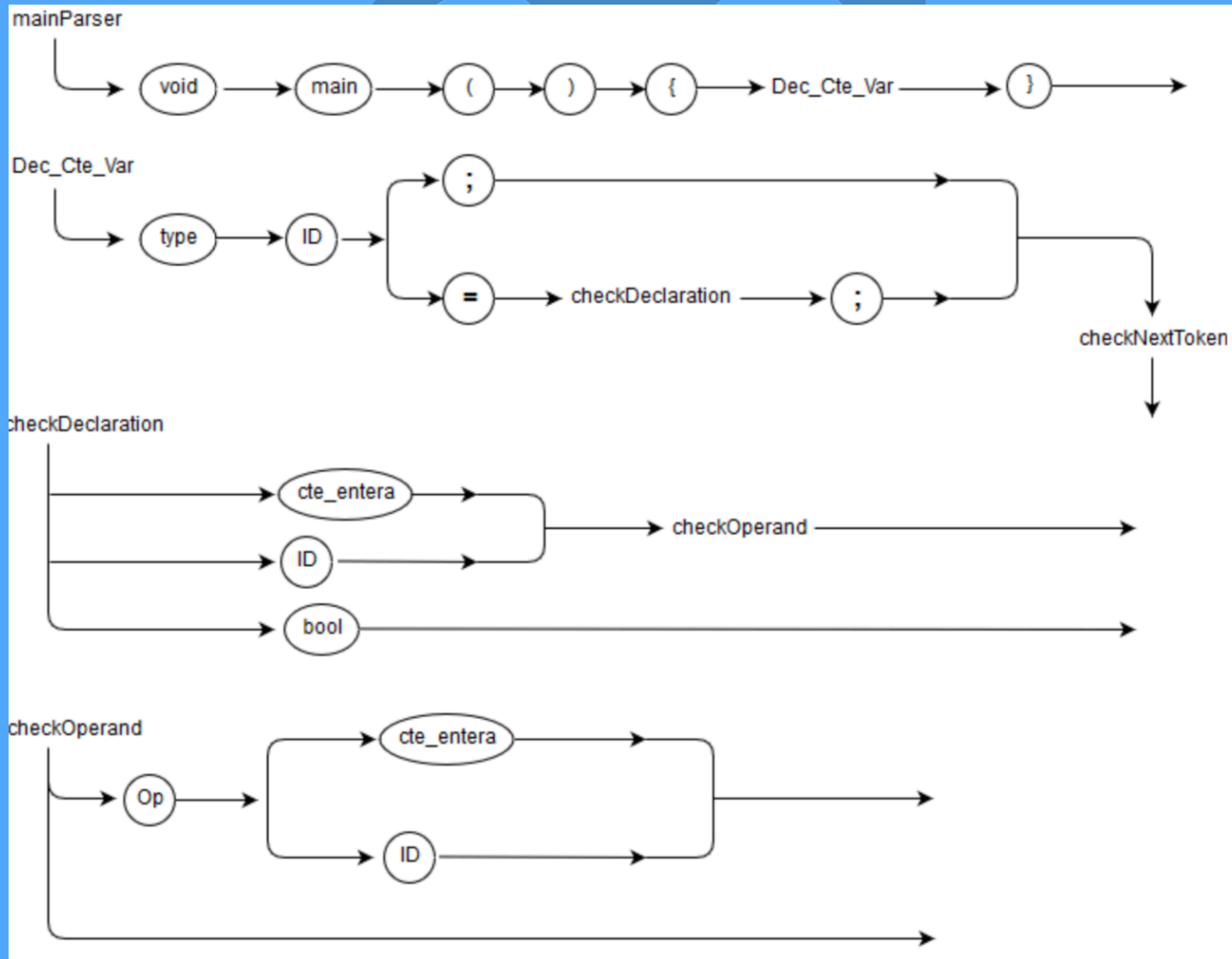
1010101
0101000
11100011

DIAGRAMAS
DE
CONWAY

BNF

Gramática de Conway

1010101
0101000
11100011



Gramática-BNF

1010101
0101000
11100011

3.2.1 MAINPARSER

mainParser \rightarrow <void> <main> <(> <)> <{ > Dec_Cte_Var <}>

3.2.2 DEC_CTE_VAR

Dec_Cte_Var \rightarrow <type> <ID> Dec_Cte_Var' checkNextToken

Dec_Cte_Var' \rightarrow <;>
 $\rightarrow \iff$ checkDeclaration <;>

3.2.3 CHECKDECLARATION

checkDeclaration \rightarrow <cte_Entera> checkOperand
 \rightarrow <ID> checkOperand
 \rightarrow <cte_booleana>

3.2.4 CHECKOPERAND

checkOperand \rightarrow <Op> checkOperand'
 $\rightarrow \lambda$

checkOperand' \rightarrow <cte_entera>
 \rightarrow <ID>

3.2.5 CHECKNEXTTOKEN

checkNextToken \rightarrow Dec_Cte_Var
 \rightarrow Equ_Cte_Var
 \rightarrow conditional
 \rightarrow loop
 $\rightarrow \lambda$

Semántico



1010101
0101000
11100011

Necesitamos unas reglas para poder dar significado a nuestras expresiones

Semántico



1010101
0101000
11100011

1

No está permitido declarar variables conflictivas

2

El tipo de los elementos usados en una expresión deben coincidir

3

Declarar variables previo a su uso

Proceso de compilación

1010101
0101000
11100011

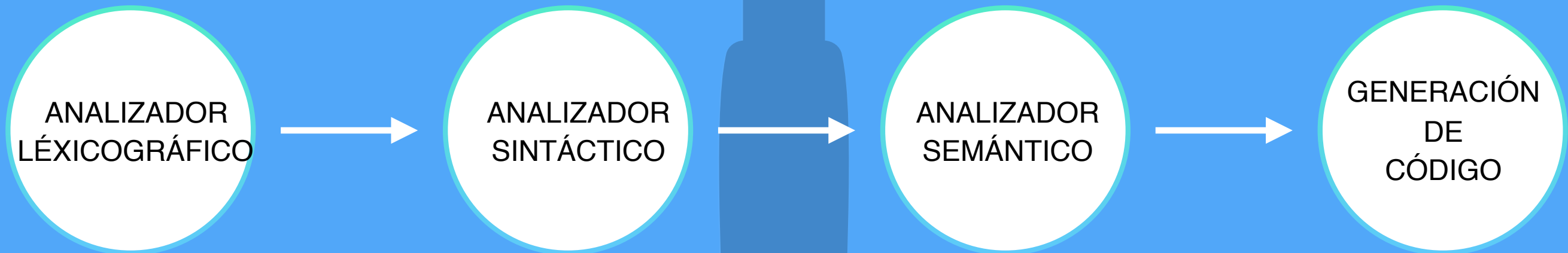


Tabla de símbolos

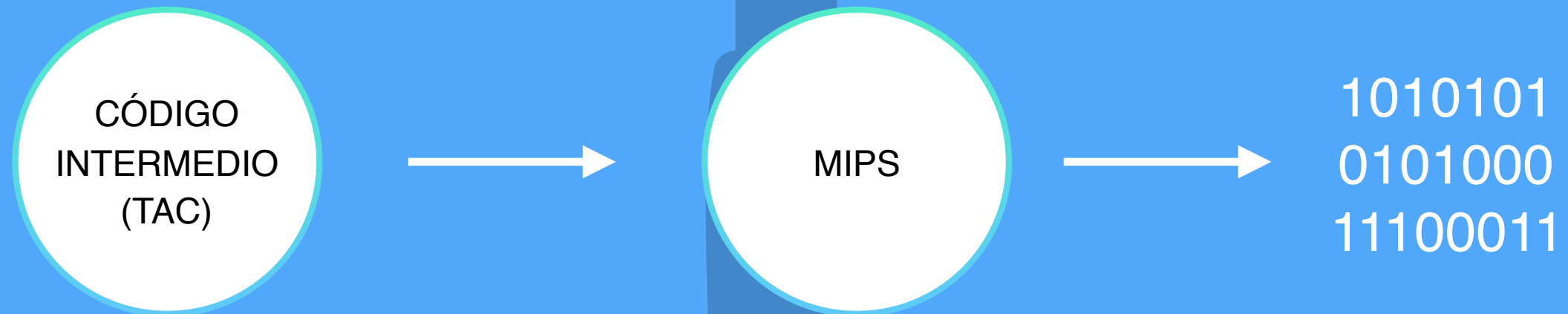


1010101
0101000
11100011

- Nombre de la variable
- Tipo de variable
- Si se ha pedido o no memoria para generar el código máquina

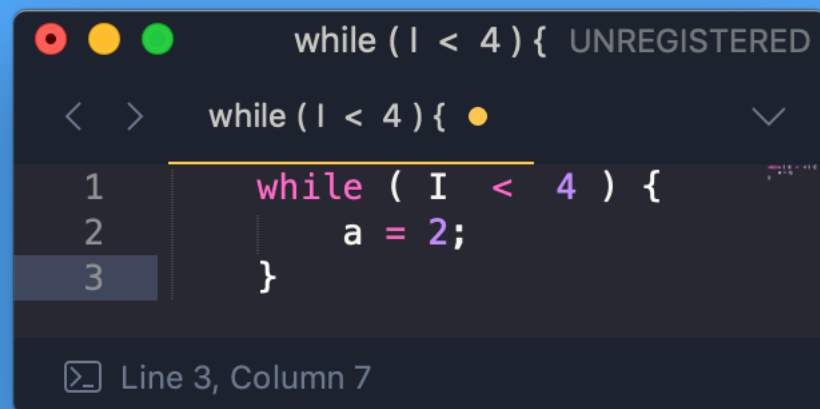
Generación de código

1010101
0101000
11100011



CÓDIGO INTERMEDIO

1010101
0101000
11100011



```
while ( I < 4 ) { UNREGISTERED  
< > while ( I < 4 ) {  
1 while ( I < 4 ) {  
2     a = 2;  
3 }  
Line 3, Column 7
```

L1

$t1 \leftarrow i < 4$

$t2 \leftarrow if(t1) goto L2$

if (t2) goto L3

L2

$a = 2$

goto L1

L3

MIPS

1010101
0101000
11100011

```
test_final.asm  UNREGISTERED
test_final.asm
1  .data #declare storage for variables
2  vara: .word 0
3  varb: .word 0
4  varc: .word 0
5
6  .text
7  L1:
8
9  bgt $t0,$t1,L2      #branch if first parameter > second parameter
10
11 #Space for else body (non-usuable for while statements)
12
13 b FinL2             #go to flag FinL2 if condition is not met
14
15 L2:
16
17 sw $t0,varb         #store word: store word from source register into RAM destination
18 li $t1,2            #load integer: loads the an integer to the specified register
19 beq $t0,$t1,I2      #branch if first parameter == second parameter
20
21 #Space for else body (non-usuable for while statements)
22
23 b FinI2             #go to flag FinI2 if condition is not met
24
25 I2:
26
27 lw $t0,varb         #load address: loads the memory address from RAM destination to the specified variable
28 lw $t1,vara         #load address: loads the memory address from RAM destination to the specified variable
29 div $t2,$t0,$t1     #divides two registers and stores the result in the first register
30 sw $t2,vart2        #store word: store word from source register into RAM destination
31
32 lw $t0,vart2        #load address: loads the memory address from RAM destination to the specified variable
33 sw $t0,varb         #store word: store word from source register into RAM destination
34
35 FinI2:
36
37 b L1
38
39 FinL2:
40
41 end:
```

Line 6, Column 10

Tab Size: 4 MIPS

Conclusiones



1010101
0101000
11100011

Hemos aprendido a definir un lenguaje personalizado de programación desde el principio siguiendo las diversas pautas aprendidas en clase con un resultado satisfactorio

Ampliaciones



1010101
0101000
11100011

boolean

```
boolean test = true;
```

Procedimientos

```
void main(){  
}
```

Ampliaciones

1010101
0101000
11100011

Condiciones

```
if(condicion){  
}
```

Bucles

```
while(condicion){  
}
```

Líneas de futuro



1010101
0101000
11100011

- Añadir comentarios de código
- Añadir bucles adicionales
- Añadir más condicionales
- Añadir funciones

Resumen

1010101
0101000
11100011

Lenguaje de Programación GRUPO-2

ID	{A-Z, a-z}
Tipos	Entero, Booleano
Operadores	Suma, Resta, Multiplicación, Asignación, División, Igual, Menor, Mayor
Sentencias	Condicionales, Iterativas
Funciones	void main