

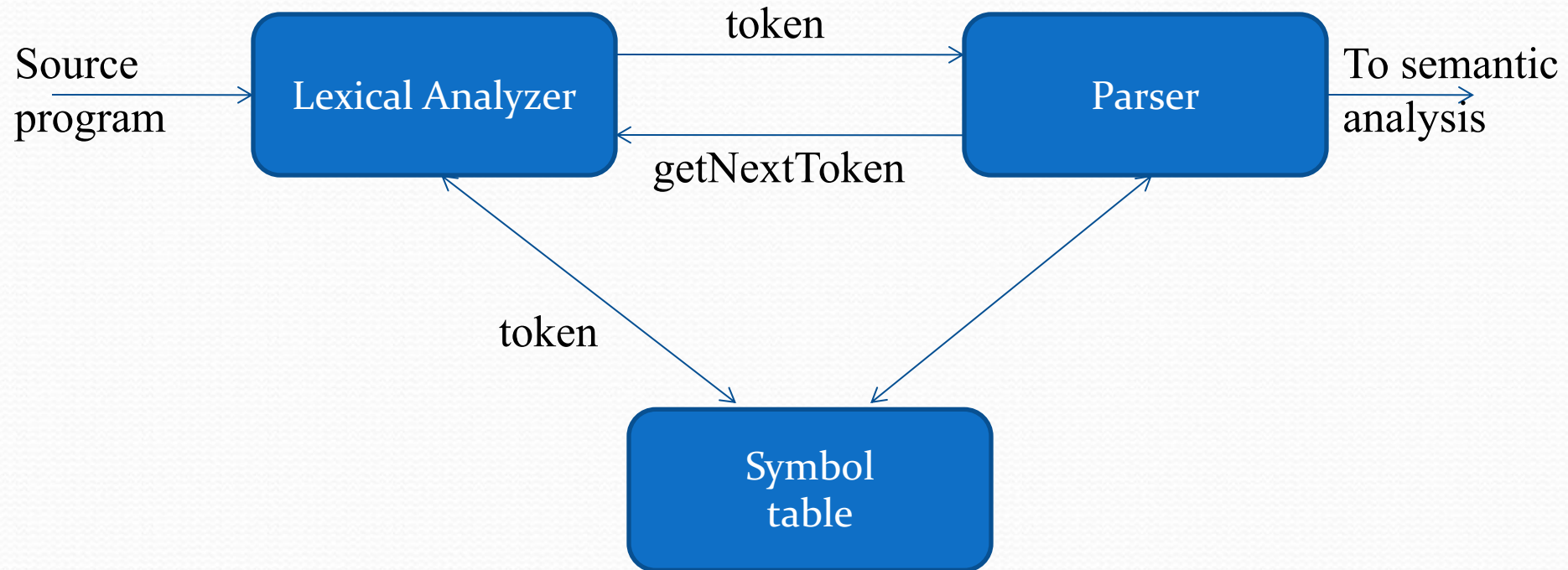
Programación de sistemas de base I

Análisis Léxico

Introducción

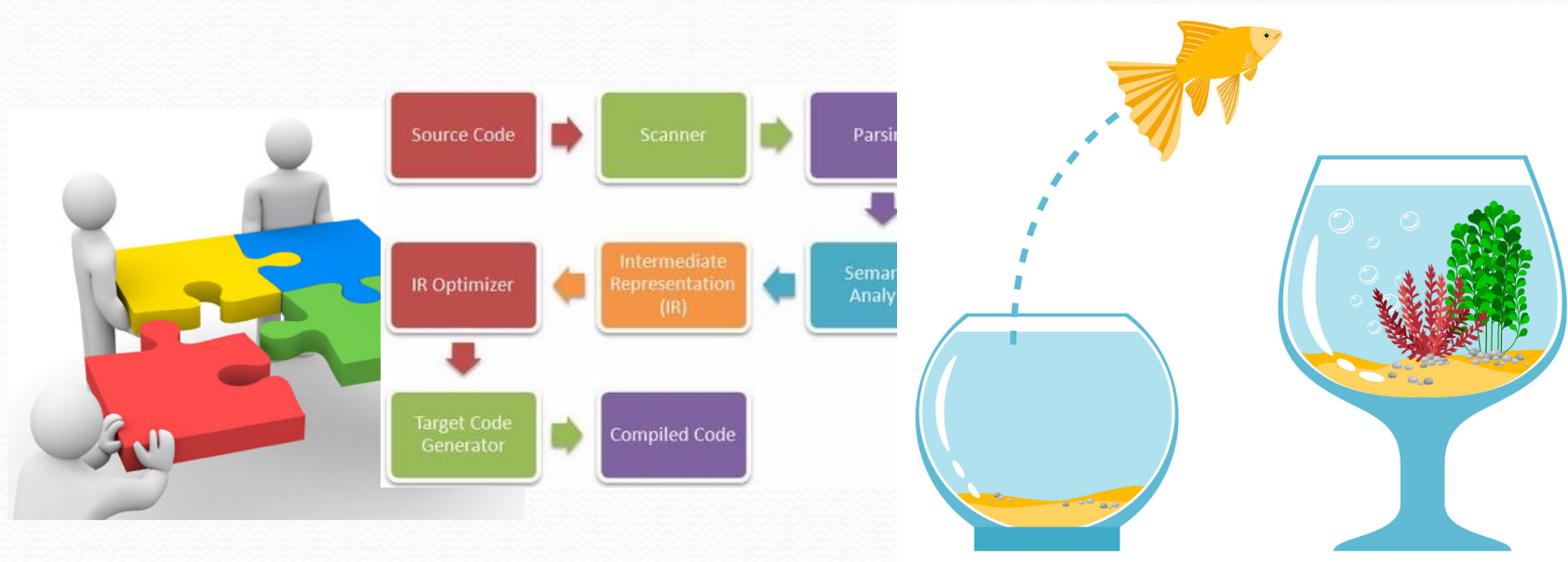
- Función del analizador léxico
- Especificación de los tokens
- Reconocimiento de tokens
- Autómatas finitos

Función del analizador léxico



¿Por qué separar el análisis léxico y el análisis sintáctico?

1. Simplicidad de diseño
2. Mejora de la eficiencia del compilador
3. Mejora de la portabilidad del compilador



Metas del análisis léxico

- Convertir descripciones físicas de un programa a secuencias de **tokens**
 - Cada token representa una pieza lógica del archivo fuente– una keyword, el nombre de una variable, etc.
- Cada token esta asociado con un **lexema**
 - El texto actual del token: “137”, ”int”, etc.
- Cada token puede tener **atributos** opcionales
 - Información adicional derivada del texto –tal vez un valor numérico
- La secuencia de tokens se utilizará en el analizador sintáctico para recuperar la estructura del programa.

Tokens, patrones y lexemas

- Token (informática), también llamado componente léxico es una cadena de caracteres que tiene un significado coherente en cierto lenguaje de programación.
- Un token es un par: un nombre de token y un valor de token opcional

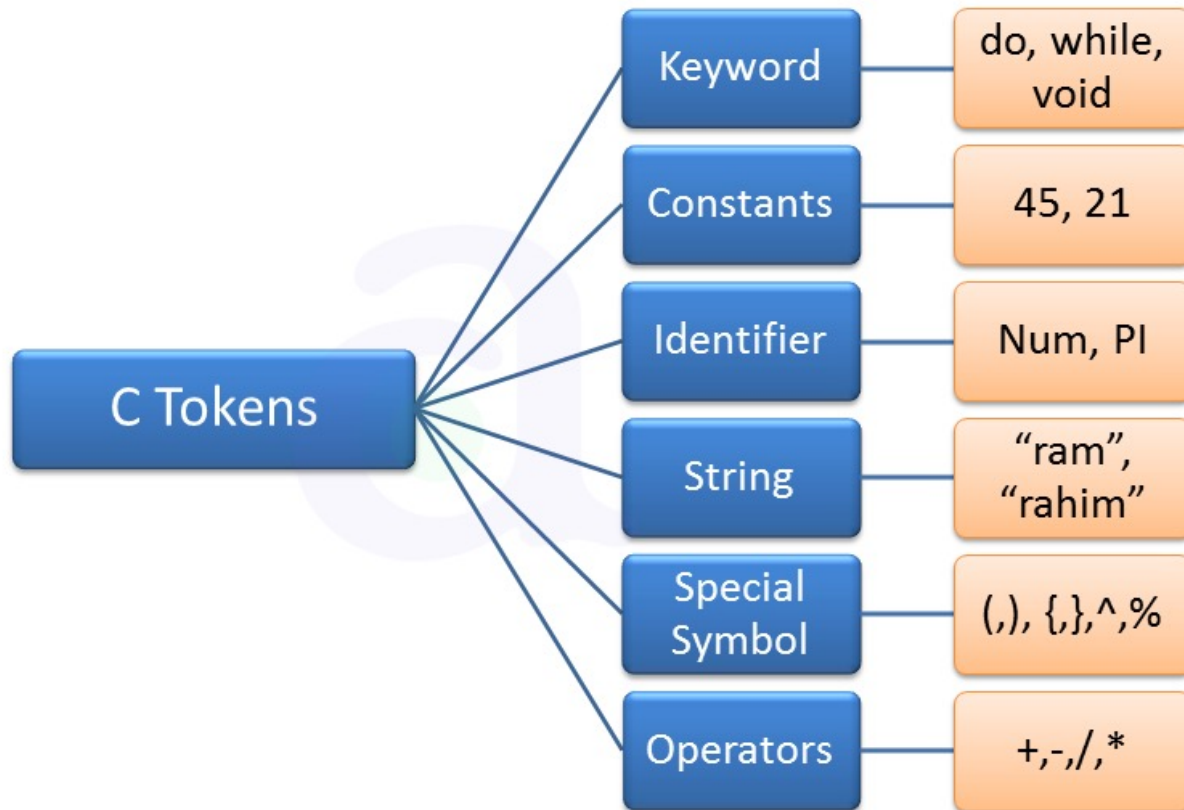
Tokens, patrones y lexemas

- Un **patrón** es una descripción de la forma que pueden adoptar los lexemas de un token
- Un **lexema** es una secuencia de caracteres en el programa fuente que coincide con el patrón de un token

Tipos de tokens

- Keywords. - Las palabras clave son palabras de la propia estructura del lenguaje, como `while` o `class` o `true`
- Los identificadores son los nombres de las variables, funciones, clases y otros elementos del código elegidos por el programador.
- Los números pueden ser formateados como enteros, o valores de punto flotante, o fracciones, o en bases alternativas como binario, octal, etc.
- Las cadenas son secuencias literales de caracteres que deben distinguirse claramente de las palabras clave o los identificadores. “ ””
- Los comentarios y los espacios en blanco se utilizan para dar formato a un programa y hacerlo visualmente claro.

Tipos de tokens



Muy bien ahora ya me sé los tokens

- Ahora solo es como deletrear...




No es tan simple ☹️



¿Cómo lo haría?

¿Con qué?



```
while (ip < z)
    ++ip;
```



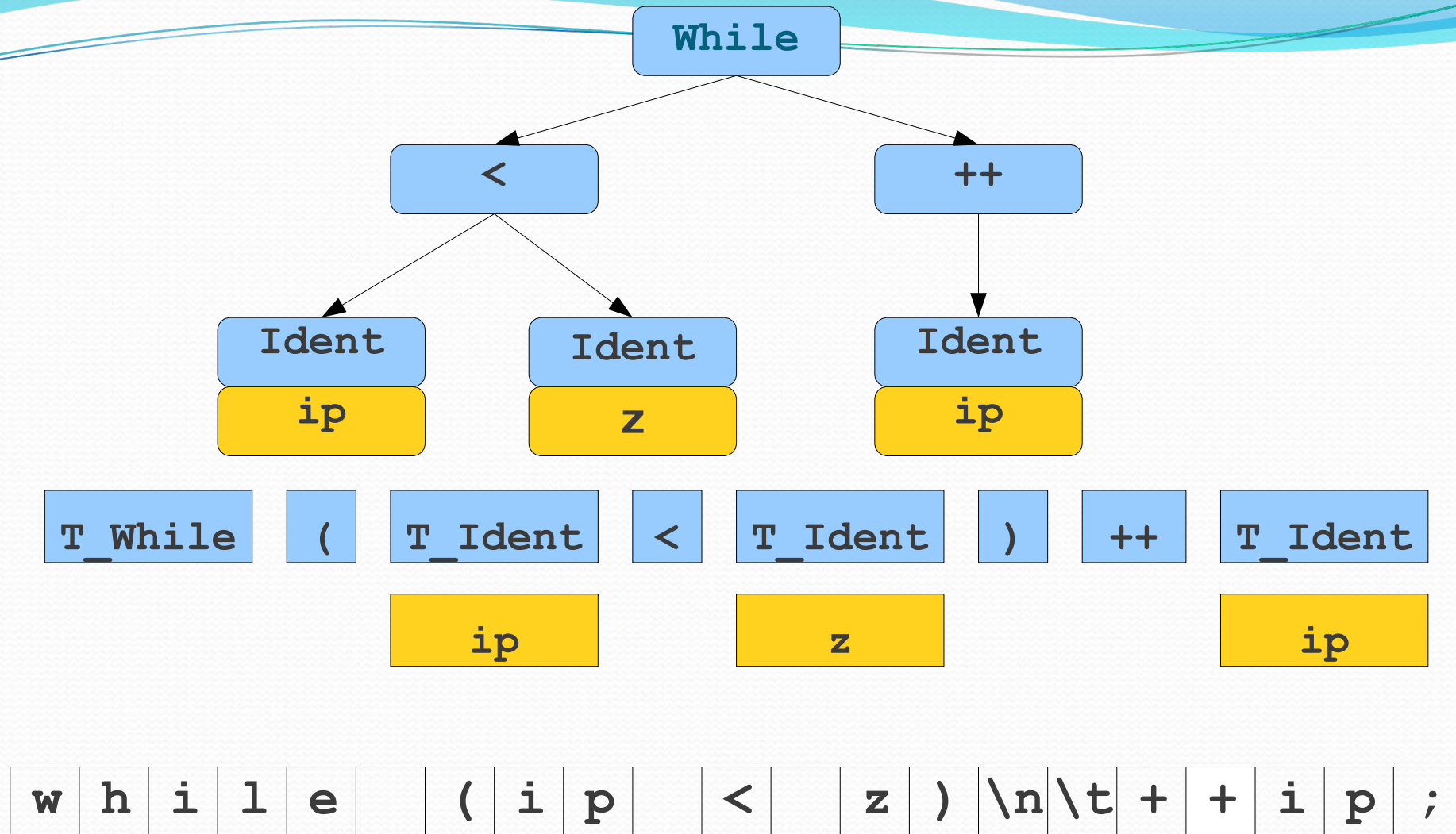
w	h	i	l	e		(i	p		<		z)	\n	\t	+	+	i	p	;
---	---	---	---	---	--	---	---	---	--	---	--	---	---	----	----	---	---	---	---	---

```
while (ip < z)
    ++ip;
```

T_While (T_Ident < T_Ident) ++ T_Ident
ip z ip

w	h	i	l	e		(i	p		<		z)	\n	\t	+	+	i	p	;
---	---	---	---	---	--	---	---	---	--	---	--	---	---	----	----	---	---	---	---	---

```
while (ip < z)
    ++ip;
```

```
while (ip < z)
    ++ip;
```

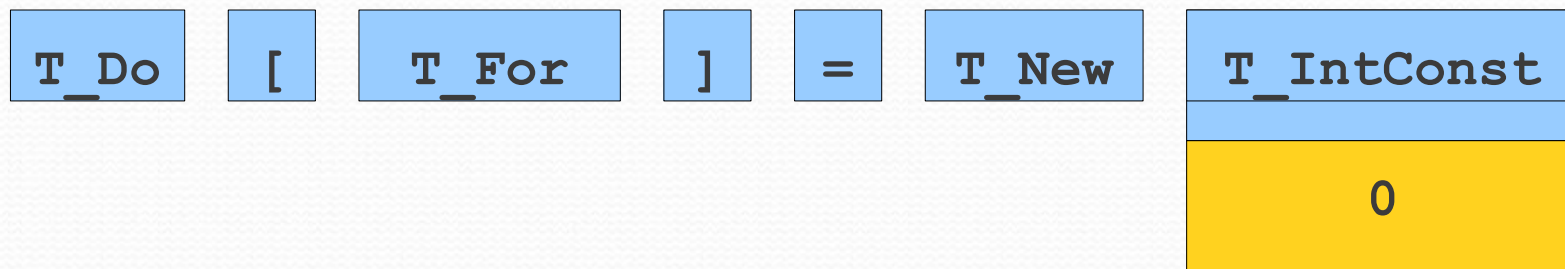


```
do[for] = new 0;
```



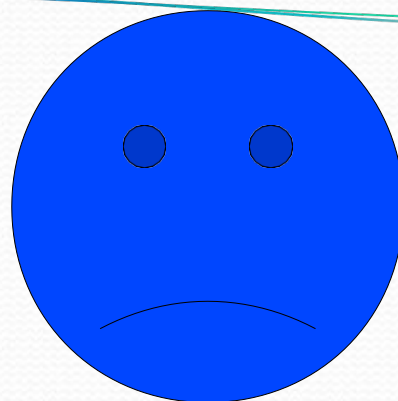
d	o	[f	o	r]		=		n	e	w		0	;
---	---	---	---	---	---	---	--	---	--	---	---	---	--	---	---

`do[for] = new 0;`



d	o	[f	o	r]		=		n	e	w		0	;
---	---	---	---	---	---	---	--	---	--	---	---	---	--	---	---

`do[for] = new 0;`



T_Do	[T_For]	=	T_New	T_IntConst
						0

d	o	[f	o	r]		=		n	e	w		0	;
---	---	---	---	---	---	---	--	---	--	---	---	---	--	---	---

do[for] = new 0;

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

T_While

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

La pieza del programa original a partir del cual hemos hecho el token se llama **lexema**.

T_While

Esto se llama **token**. Puedes pensar en él como un tipo enumerado que representa la entidad lógica que leemos del código fuente.

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

T_While

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

T_While

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

T_While

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

T_While

Algunas veces se descarta un lexema en vez de almacenarlo. Aquí se ignora un espacio en blanco dado que no tiene significado con el programa

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

T_While

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

T_While

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

T_While

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

T_While	(
---------	---

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

T_While	(
---------	---

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

T_while	(
---------	---

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

T_While	(
---------	---

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

T_while	(
---------	---

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

T_While	(
---------	---

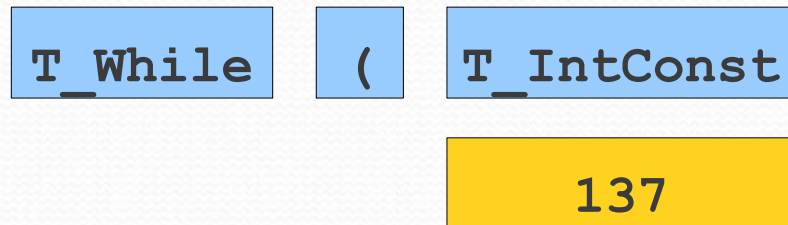
Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---

T_While	(T_IntConst
		137

Scanning a Source File

w	h	i	l	e		(1	3	7		<		i)	\n	\t	+	+	i	;
---	---	---	---	---	--	---	---	---	---	--	---	--	---	---	----	----	---	---	---	---



Algunos **tokens** tienen atributos que almacenan información adicional. Aquí almacenamos cuál entero esta representado.



Seleccionando Tokens

Cuáles Tokens son útiles aquí?

```
for (int k = 0; k < myArray[5]; ++k) {  
    cout << k << endl;  
}
```

Cuáles Tokens son útiles aquí?

```
for (int k = 0; k < myArray[5]; ++k) {  
    cout << k << endl;  
}
```

for	{
int	}
<<	;
=	<
([
)]
++	

Cuáles Tokens son útiles aquí?

```
for (int k = 0; k < myArray[5]; ++k) {  
    cout << k << endl;  
}
```

for	{
int	}
<<	;
=	<
([
)]
++	

Identifier

IntegerConstant

Escogiendo “buenos” Tokens

- Depende mucho del lenguaje
- Típicamente:
 - Keywords
 - Símbolos de puntuación
 - Agrupar lexemas que representan identificadores, constantes numéricas, cadenas, etc.
 - Descartar información irrelevante
 - Espacios
 - comentarios

“Escanear” es complicado

- FORTRAN: Los espacios en blanco son irrelevantes

```
DO 5 I = 1,25
```

```
DO 5 I = 1.25
```


“Escanear” es complicado

- FORTRAN: Los espacios en blanco son irrelevantes

```
DO 5 I = 1,25
```

```
DO5I = 1.25
```


“Escanear” es complicado

- FORTRAN: Los espacios en blanco son irrelevantes

DO 5 I = 1,25

DO5I = 1.25

- Puede ser difícil saber cuándo hay que dividir la entrada.

“Escanear” es complicado

- C++: Declaraciones de plantilla anidadas

```
vector<vector<int>> myVector
```


“Escanear” es complicado

- C++: Declaraciones de plantilla anidadas

```
vector < vector < int >> myVector
```


“Escanear” es complicado

- C++: Declaraciones de plantilla anidadas

```
(vector < (vector < (int >> myVector) ) )
```

“Escanear” es complicado

- C++: Declaraciones de plantilla anidadas

```
(vector < (vector < (int >> myVector) ) )
```

- De nuevo, puede ser complicado determinar dónde dividir.

“Escanear” es complicado

- PL/1: Keywords pueden usarse como identificadores.

“Escanear” es complicado

- PL/1: Keywords pueden usarse como identificadores.

```
IF THEN THEN THEN = ELSE; ELSE ELSE = IF
```

“Escanear” es complicado

- PL/1: Keywords pueden usarse como identificadores.

```
IF THEN THEN THEN = ELSE; ELSE ELSE = IF
```


“Escanear” es complicado

- PL/1: Keywords pueden usarse como identificadores.

IF THEN **THEN** THEN = ELSE; **ELSE** ELSE = IF

- Puede ser complicado determinar cómo etiquetar lexemas

Desafíos al escanear

- ¿Cómo determinamos qué lexemas están asociados a cada token?
- Cuando hay múltiples formas de escanear la entrada, ¿cómo sabemos cuál elegir?
- ¿Cómo podemos resolver estos problemas de forma eficaz?




Asociación de lexemas con Tokens

Lexemas y Tokens

- Los tokens permiten clasificar los lexemas según la información que proporcionan.
- Algunos tokens pueden estar asociados a un solo lexema:
 - Tokens para keywords como **if** y **while** probablemente solo empatan con esos lexemas exactamente
- Algunos tokens podrían estar asociados con muchos lexemas distintos
 - Todos los nombres de variables, posibles números, todas las posibles cadenas, etc.

Conjuntos de lexemas

- Idea: Asociar un conjunto de lexemas a cada token.
- Podríamos asociar el token "número" con el conjunto { 0, 1, 2, ..., 10, 11, 12, ... }
- Podríamos asociar el token "cadena" con el conjunto { "", "a", "b", "c", ... }
- Podríamos asociar el token de la palabra clave **while** con el conjunto { **while** }.



¿Cómo describimos qué conjunto
(potencialmente infinito) de lexemas
está asociado a cada tipo de token?

Lenguajes Formales

Un **lenguaje formal** es un conjunto de cadenas.

Muchos lenguajes infinitos tienen descripciones finitas:

- Definir el lenguaje mediante un autómata.
- Definir el lenguaje mediante una gramática.
- Definir el lenguaje mediante una expresión regular.

Podemos utilizar estas descripciones compactas del lenguaje para definir conjuntos de cadenas.

Expresiones Regulares

- **Expresiones regulares** son una familia de descripciones que pueden utilizarse para capturar determinados lenguajes (los lenguajes regulares).
- A menudo proporcionan una descripción compacta y legible para lenguaje.
- Se utilizan como base de numerosos sistemas de software, incluida la herramienta flex que utilizaremos en este curso.