

# **COP 3402 Systems Software**

---

## **Compilers And Interpreters**

# Outline

---

1. Compiler and interpreters
2. Compilation process
3. Interpreters
4. PL/0 Symbols (tokens)

# Compilers / Interpreters

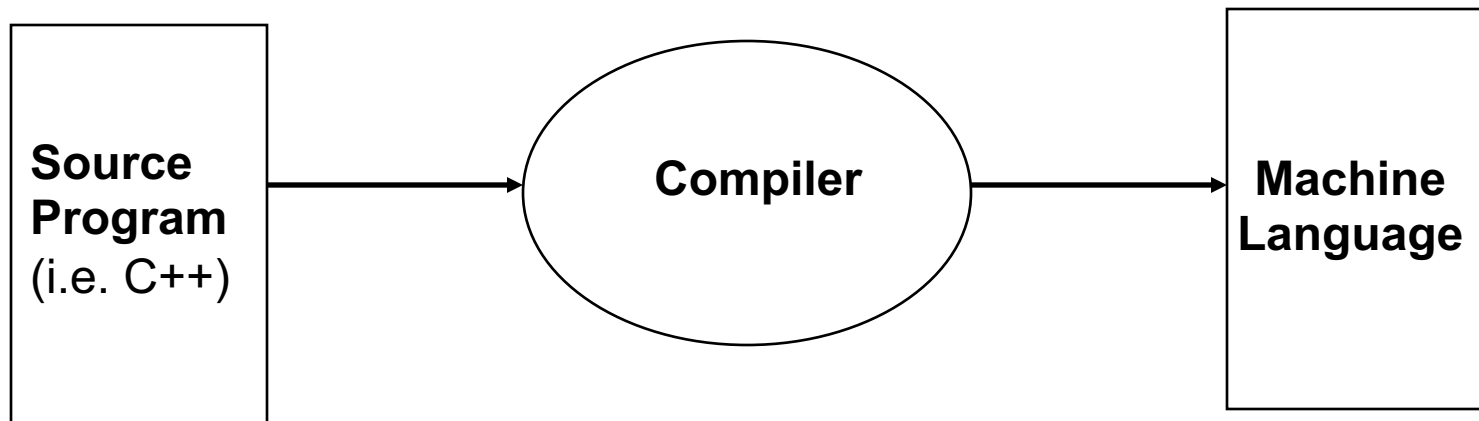
---

- Programming languages are notations for describing computations to people and to machines.
- Programming languages can be implemented by any of three general methods:
  1. Compilation
  2. Interpretation
  3. Hybrid Implementation

# Compilers

---

A compiler is a program that takes high level languages (i.e. Pascal, C, ML) as input, and translates it to a low-level representation (machine language).



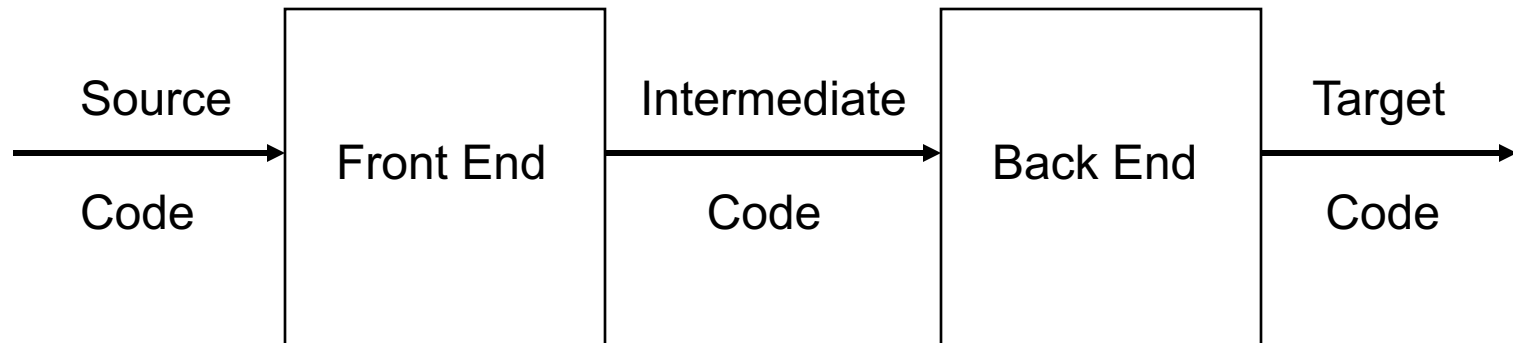
# Compilers

---

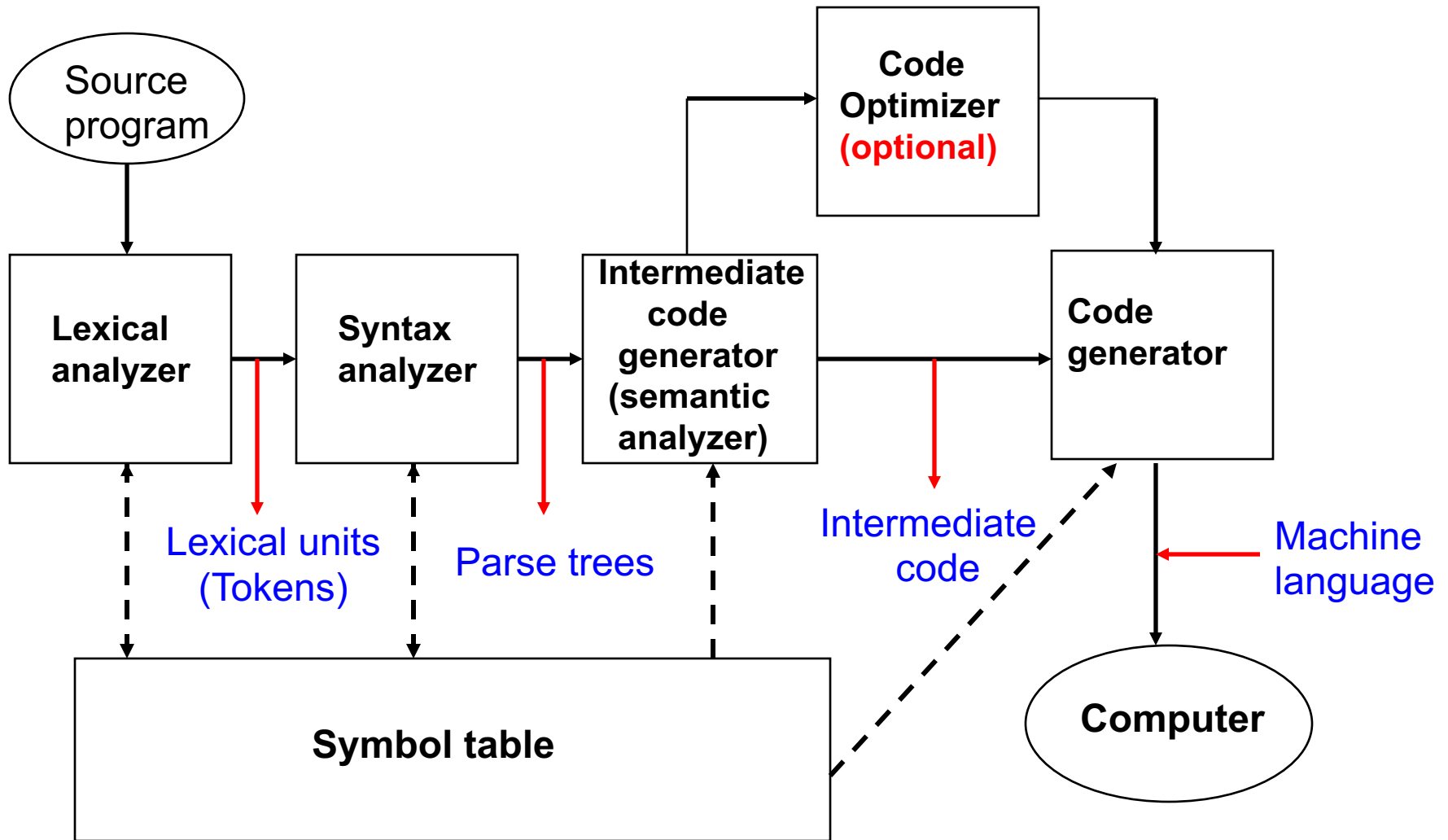
The process of compilation and program execution take place in several phases:

Front end: Scanner → Parser → Semantic Analyzer

Back end: Code generator



# Compilers



## EXAMPLE:

Fahrenheit := 32 + celsious \* 1.8

| f | a | h | r | e | n | h | e | i | t | : | = | 3 | 2 | + | c | e | l | s | i | o | u | s | \* | 1 | . | 8 | ; |

Getchar()

Lexical analyzer (scanner)

(converts from character stream into  
a stream of tokens.)

[ id, 1 ] [ : = ] [ int, 32 ] [ + ] [ id, 2 ] [ \* ] [ int, 1.8 ] [ ; ]

Symbol Table

|   |            |      |
|---|------------|------|
| 1 | fahrenheit | real |
| 2 | celsious   | real |

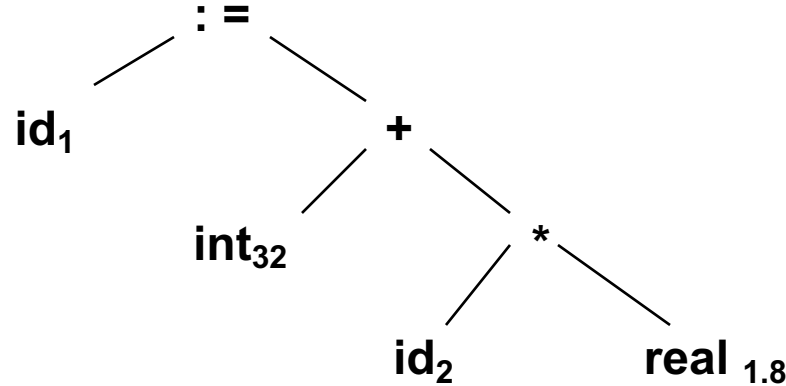
name

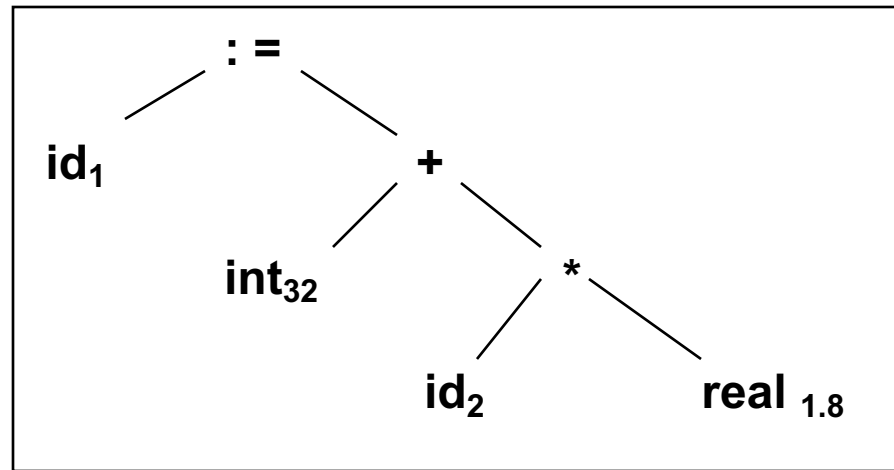
attribute

Syntax analyzer (parser)

(Construct syntactic structure of the program)

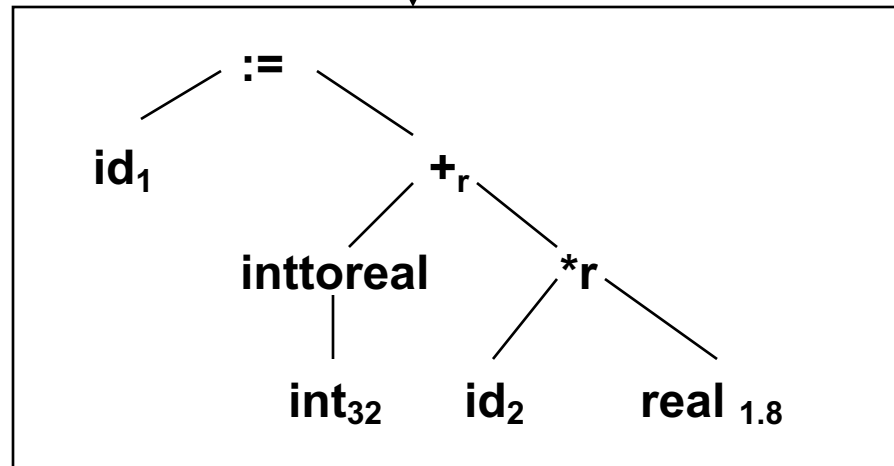
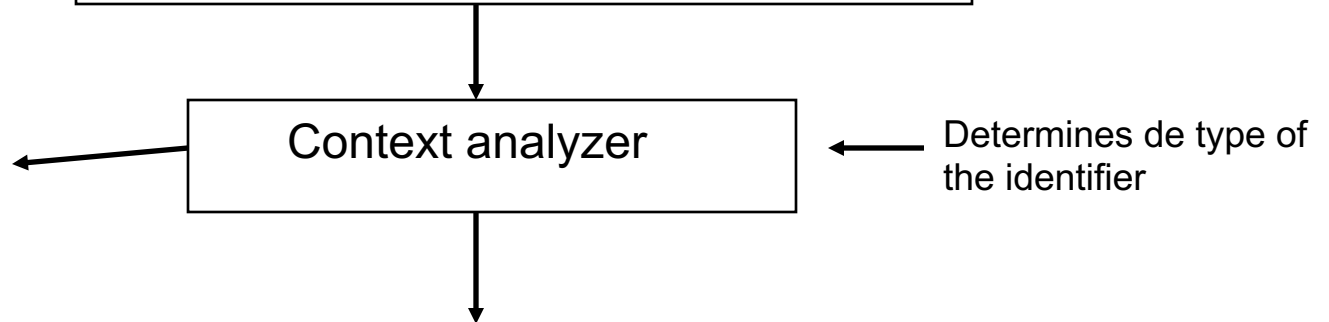
index in symbol table



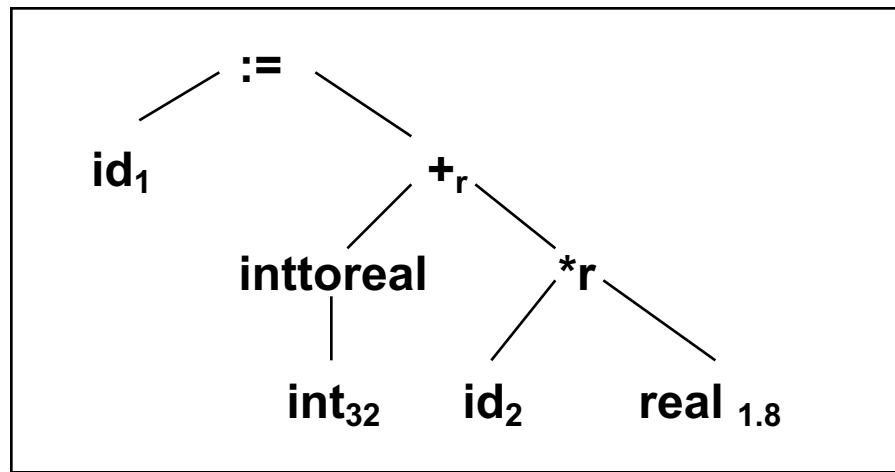


Symbol Table

|   |            |      |
|---|------------|------|
| 1 | fahrenheit | real |
| 2 | celsious   | real |







Symbol Table

|   |            |      |
|---|------------|------|
| 1 | fahrenheit | real |
| 2 | celsious   | real |

Intermediate code generator

```

Temp1 := inttoreal(32)
Temp2 := id2
Temp2 := Temp2 * 1.8
Temp1 := Temp1 + Temp2
id1 := Temp1
  
```

Intermediate code

```
Temp1 := inttoreal(32)
Temp2 := id2
Temp2 := Temp2 * 1.8
Temp1 := Temp1 + Temp2
id1 := Temp1
```

← Intermediate code

Code optimizer

```
Temp1 := id2
Temp1 := Temp1 * 1.8
Temp1 := Temp1 + 32.0
id1 := Temp1
```

← optimized code

Symbol Table

|   |            |      |
|---|------------|------|
| 1 | fahrenheit | real |
| 2 | celsious   | real |

```
Temp1 := id2
Temp1 := Temp1 * 1.8
Temp1 := Temp1 + 32.0
id1 := Temp1
```

← optimized code

Code generator

```
movf id2, r1
mulf #1.8, r1
addf #32.0, r1
movf r1, id1
```

← assembly instructions

Symbol Table

|   |            |      |
|---|------------|------|
| 1 | fahrenheit | real |
| 2 | celsious   | real |

# Compilers

---

## Lexical analyzer:

Gathers the characters of the source program into lexical units.

Lexical units of a program are:

- identifiers

- special words (reserved words)

- operators

- special symbols

- Comments are ignored!**

## Syntax analyzer:

Takes lexical units from the lexical analyzer and use them to construct a hierarchical structure called **parse tree**

Parse trees represent the syntactic structure of the program.

# Compilers

---

## Intermediate code:

Produces a program in a different language representation:

- Assembly language

- Similar to assembly language

- Something higher than assembly language

Note: semantic analysis is an integral part of the intermediate code generator

## Optimization:

Makes programs smaller or faster or both.

Most optimization is done in the intermediate code.  
(i.e. tree reduction, vectorization)

# Compilers

---

## Code generator:

Translate the optimized intermediate code into machine language.

## The symbol table:

Serve as a database for the compilation process.

Contents type and attribute information of each user-defined name in the program.

Symbol Table

|   |            |      |  |
|---|------------|------|--|
| 1 | fahrenheit | real |  |
| 2 | celsious   | real |  |

↑      ↑      ↑      ↑  
Index   name   type   attributes

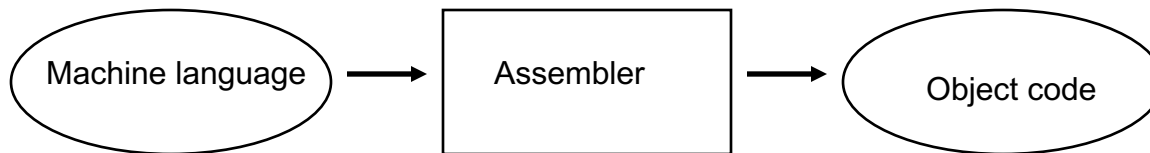
# Compilers

---

## Machine language

A program in its machine language form needs in general

- To be translated to object code for execution
- To translate the program from its machine language form (assembly language) into object code, an assembler is required.
- An assembler is a program that translate machine code into object code



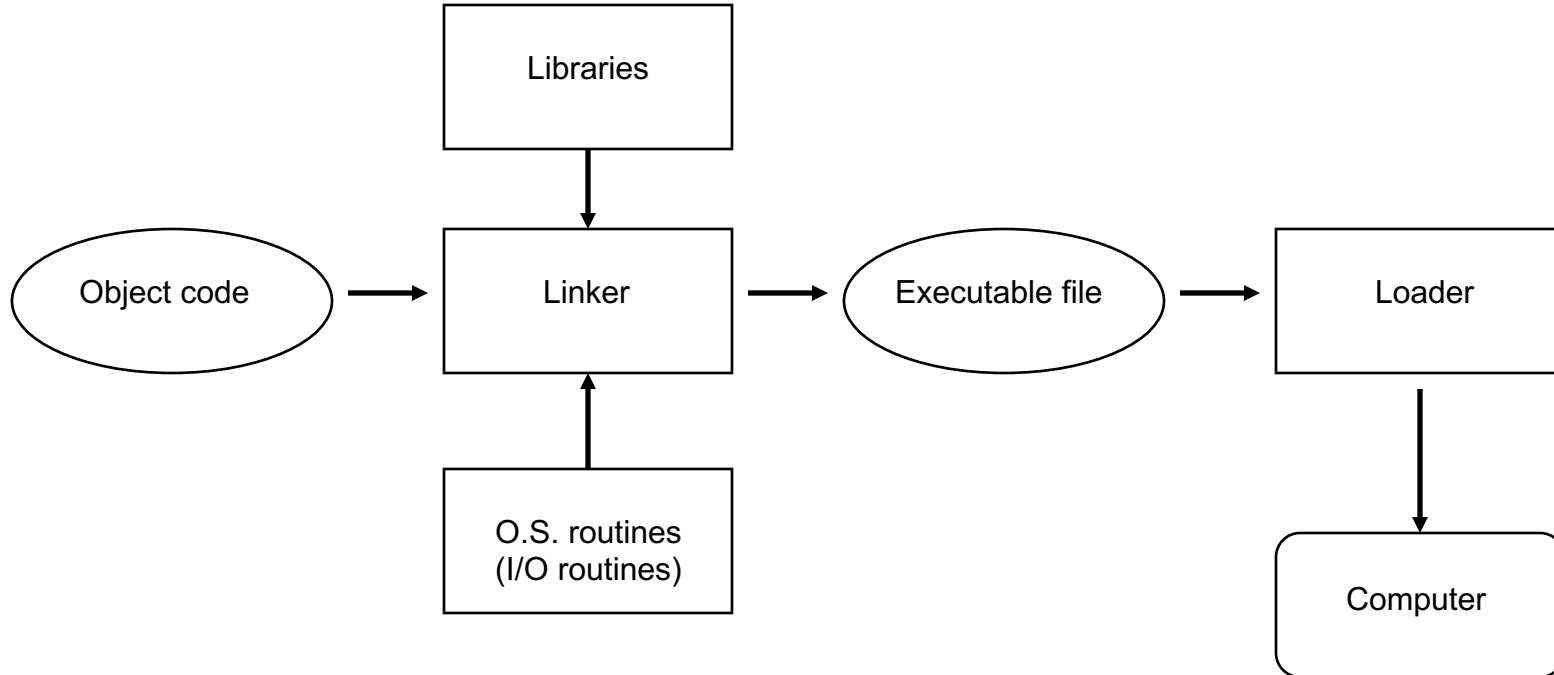
# Compilers

---

## Machine language

To run a program in its object code form, it needs in general

- some other code (libraries)
- programs from the O.S. (i.e. input/output routines)





# Interpreters

---

**Programs are interpreted (executed) by another program called the interpreter.**

**Advantages: Easy implementation of many source-level debugging operations, because all run-time errors operations refer to source-level units.**

**Disadvantages: 10 to 100 times slower because statements are interpreted each time the statement is executed.**

**Background:**

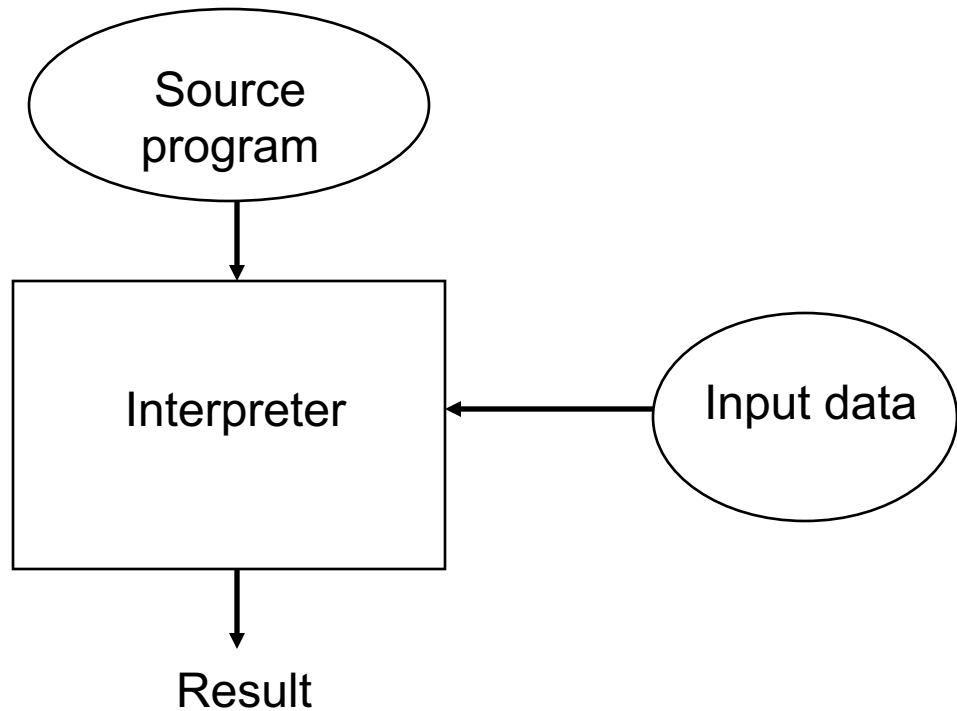
**Early sixties → APL, SNOBOL, Lisp.**

**By the 80s → rarely used.**

**Recent years → Significant comeback ( some Web scripting languages: JavaScript, php)**

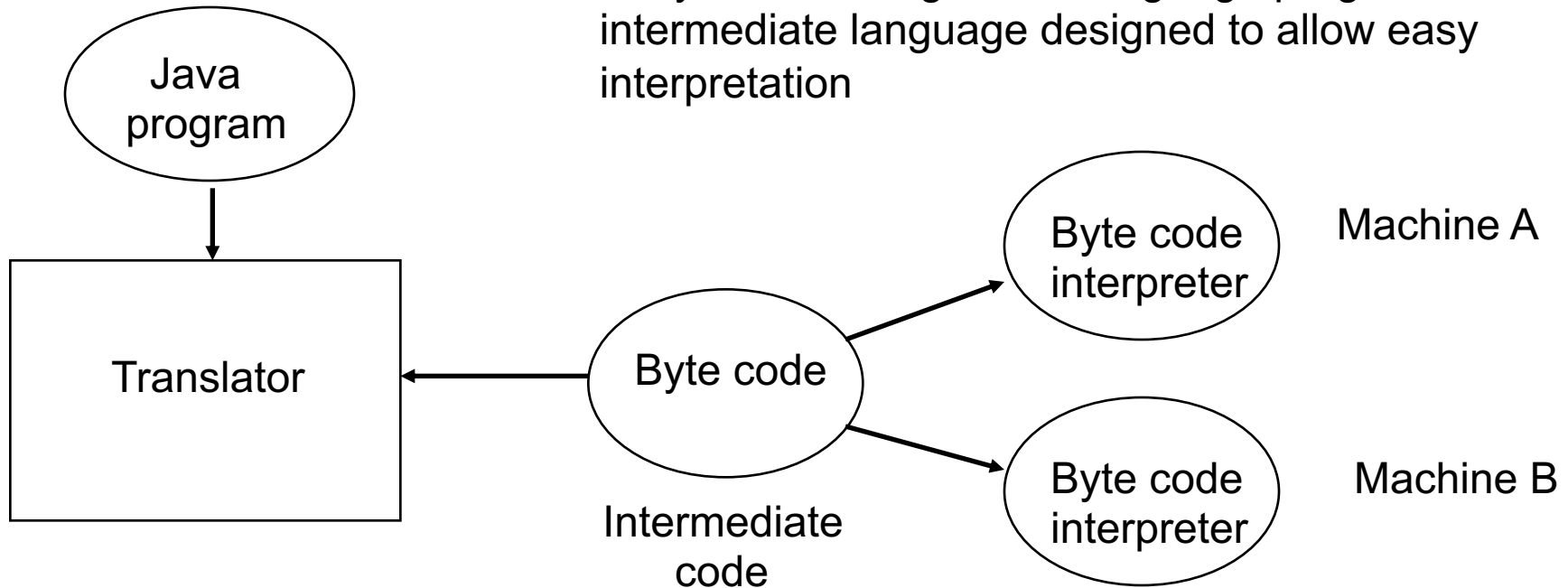
# Interpreters

---



# Hybrid implementation systems

They translate high-level language programs to an intermediate language designed to allow easy interpretation



Example: PERL and initial implementations of Java

# Interpreters

---

## Just-In-Time (JIT) implementation

Programs are translated to an intermediate language.

During execution, it compiles intermediate language methods into machine code when they are called.

The machine code version is kept for subsequent calls.

.NET and Java programs are implemented with JIT system.

# PL/0 Symbols

---

Given the following program written in PL/0:

```
const m = 7, n = 85;
var i,x,y,z,q,r;
procedure mult;
  var a, b;
begin
  a := x; b := y; z := 0;
  while b > 0 do
    begin
      if odd x then z := z+a;
      a := 2*a;
      b := b/2;
    end
  end;
begin
  x := m;
  y := n;
  call mult;
end.
```

As in any language, in PL/0 we need to identify what is the vocabulary and what are the valid names and special symbols that we accept as valid:

# PL/0 Symbols

---

Given the following program written in PL/0:

```
const m = 7, n = 85;
var i,x,y,z,q,r;
procedure mult;
  var a, b;
begin
  a := x; b := y; z := 0;
  while b > 0 do
    begin
      if odd x then z := z+a;
      a := 2*a;
      b := b/2;
    end
  end;
begin
  x := m;
  y := n;
  call mult;
end.
```

As in any language, in PL/0 we need to identify what is the vocabulary and what are the valid names and special symbols that we accept as valid:

For instance, in the on the example we notice that there are many reserved words (keywords)

# PL/0 Symbols

---

Given the following program written in PL/0:

```
const m = 7, n = 85;
var i,x,y,z,q,r;
procedure mult;
  var a, b;
begin
  a := x; b := y; z := 0;
  while b > 0 do
    begin
      if odd x then z := z + a;
      a := 2 * a;
      b := b / 2;
    end
  end;
begin
  x := m;
  y := n;
  call mult;
end.
```

Also there are some operators and special symbols:

a) Operators ( +, -, \*, /, <, =, >, <=, <>, >=, :=)

# PL/0 Symbols

---

Given the following program written in PL/0:

```
const m = 7, n = 85;
var i, x, y, z, q, r;
procedure mult;
  var a, b;
begin
  a := x; b := y; z := 0;
  while b > 0 do
    begin
      if odd x then z := z + a;
      a := 2 * a;
      b := b / 2;
    end
  end;
end;
begin
  x := m;
  y := n;
  call mult;
end.
```

Also there are some operators and special symbols:

- a) Operators ( +, -, \*, /, <, =, >, <=, <>, >=, :=)
- b) Special symbols  
( , ), [ , ] , , . , : , ;



# PL/0 Symbols

---

Given the following program written in PL/0:

```
const m = 7, n = 85;
var i, x, y, z, q, r;
procedure mult;
  var a, b;
begin
  a := x; b := y; z := 0;
  while b > 0 do
    begin
      if odd x then z := z + a;
      a := 2 * a;
      b := b / 2;
    end
  end;
begin
  x := m;
  y := n;
  call mult;
end.
```

There are also:  
Numerals such as : 5, 0, 85, 2, 346, . . .

# PL/0 Symbols

---

Given the following program written in PL/0:

```
const m = 7, n = 85;
var i, x, y, z, q, r;
procedure mult;
  var a, b;
begin
  a := x; b := y; z := 0;
  while b > 0 do
    begin
      if odd x then z := z + a;
      a := 2 * a;
      b := b / 2;
    end
  end;
begin
  x := m;
  y := n;
  call mult;
end.
```

There are also:  
Numerals such as : 5, 0, 85, 2, 346, . . .

And names (identifiers):

A letter

or a letter followed by more letters

or a letter followed by more letters or digits.

Examples: x, m, celsious, mult, intel486

# Scanner

---

Given the following program written in PL/0:

```
const m = 7, n = 85;
var i,x,y,z,q,r;
procedure mult;
  var a, b;
begin
  a := x; b := y; z := 0;
  while b > 0 do
    begin
      if odd x then z := z+a;
      a := 2*a;
      b := b/2;
    end
  end;
begin
  x := m;
  y := n;
  call mult;
end.
```

In addition there are also:  
**Comments:**

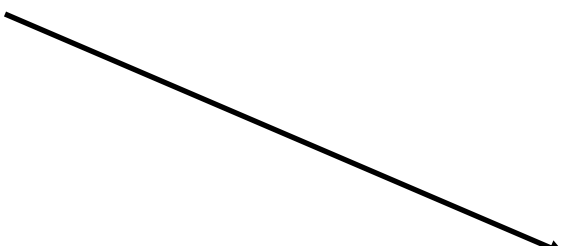
*/\* in C \*/*

*(\* in Pascal \*)*

**Separators:**

white spaces

invisible characters like: tab “\t”  
new line “\n”



**Example:** \t a :=  2 \* a;\n

# Scanner

---

Given the following program written in PL/0:

```
const m = 7, n = 85;
var i,x,y,z,q,r;
procedure mult;
  var a, b;
begin
  a := x; b := y; z := 0;
  while b > 0 do
    begin
      if odd x then z := z+a;
      a := 2*a;
      b := b/2;
    end
  end;
begin
  x := m;
  y := n;
  call mult;
end.
```

**Every language has an alphabet  
(a finite set of characters)**

PL/0 alphabet { a, b, c, d, e, f, g, h, i, j, k, l, m, n,  
o, p, q, r, s, t, u, v, w, x, y, z, 0, 1, 2,  
3, 4, 5, 6, 7, 8, 9, +, -, \*, /, <, =, >, :,  
. , , ; }

**Using concatenation (joining two or more  
characters) we obtain a string of symbols.**

# Scanner

---

A **language L**, is simply any set of strings over a fixed alphabet.

Alphabet

$\{0,1\}$

$\{a,b,c\}$

$\{A, \dots, Z\}$

$\{A, \dots, Z, a, \dots, z, 0, \dots, 9, \\ +, -, \dots, <, >, \dots\}$

Languages

$\{0,10,100,1000,100000\dots\}$

$\{0,1,00,11,000,111,\dots\}$

$\{abc,aabbcc,aaabbbccc,\dots\}$

$\{TEE,FORE,BALL,\dots\}$

$\{FOR,WHILE,GOTO,\dots\}$

$\{ \text{All legal PASCAL progs} \}$

$\{ \text{All grammatically correct}$

$\text{English sentences} \}$

Special Languages:  $\emptyset$  - EMPTY LANGUAGE

$\in$  - contains  $\in$  string only

# Scanner

---

**The purpose of the lexical analyzer (scanner) is to decompose the source program into its elementary symbols or tokens:**

- 1. Read input characters of the source program.**
- 2. Group them into lexemes ( a lexeme is a sequence of characters that matches the pattern for a token).**
- 3. Produce a token for each lexeme**

**A lexeme (lowest level syntactic unit) is a sequence of characters in the source program**

# Scanner

---

**Scan Input**

**Remove WS, NL, ...**

**Identify Tokens**

**Create Symbol Table**

**Insert Tokens into ST**

**Generate Errors**

**Send Tokens to Parser**

**A lexeme (lowest level syntactic unit) is  
a sequence of characters in the source  
program**

# Scanner

## ASCII Character Set

X

The ordinal number of a character **ch** is computed from its coordinates (X,Y) in the table as:

$$\text{ord}(\text{ch}) = 16 * X + Y$$

Example:

$$\text{ord}('A') = 16 * 4 + 1 = 65$$

$$\text{ord}('0') = 16 * 3 + 0 = 48$$

$$\text{ord}('5') = 16 * 3 + 5 = 53$$

Y

|       | 0   | 1   | 2  | 3 | 4 | 5 | 6 | 7   |
|-------|-----|-----|----|---|---|---|---|-----|
| 0     | NUL | DLE | SP | 0 | @ | P | ` | p   |
| 1     | SOH | DC1 | !  | 1 | A | Q | a | q   |
| 2     | STX | DC2 | "  | 2 | B | R | b | r   |
| 3     | ETX | DC3 | #  | 3 | C | S | c | s   |
| 4     | EOT | DC4 | \$ | 4 | D | T | d | t   |
| 5     | ENQ | NAK | %  | 5 | E | U | e | u   |
| 6     | ACK | SYN | &  | 6 | F | V | f | v   |
| 7     | BEL | ETB | '  | 7 | G | W | g | w   |
| 8     | BS  | CAN | (  | 8 | H | X | h | x   |
| 9     | HT  | EM  | )  | 9 | I | Y | i | y   |
| 10(A) | LF  | SUB | *  | : | J | Z | j | z   |
| 11(B) | VT  | ESC | +  | ; | K | [ | k | {   |
| 12(C) | FF  | FS  | ,  | < | L | \ | l |     |
| 13(D) | CR  | GS  | -  | = | M | ] | m | }   |
| 14(E) | SO  | RS  | .  | > | N | ^ | n | ~   |
| 15(F) | SI  | US  | /  | ? | O | _ | o | DEL |



## ASCII character table

| Dec | Hex | ASCII                     | Dec | Hex | ASCII                           | Dec | Hex | ASCII      |
|-----|-----|---------------------------|-----|-----|---------------------------------|-----|-----|------------|
| 0   | 00  | NUL (null)                | 16  | 10  | DLE (data link escape)          | 32  | 20  | SP (space) |
| 1   | 01  | SOH (start of heading)    | 17  | 11  | DC1 (device control 1)          | 33  | 21  | !          |
| 2   | 02  | STX (start of text)       | 18  | 12  | DC2 (device control 2)          | 34  | 22  | "          |
| 3   | 03  | ETX (end of text)         | 19  | 13  | DC3 (device control 3)          | 35  | 23  | #          |
| 4   | 04  | EOT (end of transmission) | 20  | 14  | DC4 (device control 4)          | 36  | 24  | \$         |
| 5   | 05  | ENQ (enquiry)             | 21  | 15  | NAK (negative acknowledge)      | 37  | 25  | %          |
| 6   | 06  | ACK (acknowledge)         | 22  | 16  | SYN (synchronous idle)          | 38  | 26  | &          |
| 7   | 07  | BEL (bell)                | 23  | 17  | ETB (end of transmission block) | 39  | 27  | '          |
| 8   | 08  | BS (backspace)            | 24  | 18  | CAN (cancel)                    | 40  | 28  | (          |
| 9   | 09  | HT (horizontal tab)       | 25  | 19  | EM (end of medium)              | 41  | 29  | )          |
| 10  | 0A  | LF (line feed)            | 26  | 1A  | SUB (substitute)                | 42  | 2A  | *          |
| 11  | 0B  | VT (vertical tab)         | 27  | 1B  | ESC (escape)                    | 43  | 2B  | +          |
| 12  | 0C  | FF (form feed)            | 28  | 1C  | FS (file separator)             | 44  | 2C  | ,          |
| 13  | 0D  | CR (carriage return)      | 29  | 1D  | GS (group separator)            | 45  | 2D  | -          |
| 14  | 0E  | SO (shift out)            | 30  | 1E  | RS (record separator)           | 46  | 2E  | .          |
| 15  | 0F  | SI (shift in)             | 31  | 1F  | US (unit separator)             | 47  | 2F  | /          |

## ASCII character table

| Dec | Hex | ASCII | Dec | Hex | ASCII | Dec | Hex | ASCII |
|-----|-----|-------|-----|-----|-------|-----|-----|-------|
| 48  | 30  | 0     | 64  | 40  | @     | 80  | 50  | P     |
| 49  | 31  | 1     | 65  | 41  | A     | 81  | 51  | Q     |
| 50  | 32  | 2     | 66  | 42  | B     | 82  | 52  | R     |
| 51  | 33  | 3     | 67  | 43  | C     | 83  | 53  | S     |
| 52  | 34  | 4     | 68  | 44  | D     | 84  | 54  | T     |
| 53  | 35  | 5     | 69  | 45  | E     | 85  | 55  | U     |
| 54  | 36  | 6     | 70  | 46  | F     | 86  | 56  | V     |
| 55  | 37  | 7     | 71  | 47  | G     | 87  | 57  | W     |
| 56  | 38  | 8     | 72  | 48  | H     | 88  | 58  | X     |
| 57  | 39  | 9     | 73  | 49  | I     | 89  | 59  | Y     |
| 58  | 3A  | :     | 74  | 4A  | J     | 90  | 5A  | Z     |
| 59  | 3B  | ;     | 75  | 4B  | K     | 91  | 5B  | [     |
| 60  | 3C  | <     | 76  | 4C  | L     | 92  | 5C  | \     |
| 61  | 3D  | =     | 77  | 4D  | M     | 93  | 5D  | ]     |
| 62  | 3E  | >     | 78  | 4E  | N     | 94  | 5E  | ^     |
| 63  | 3F  | ?     | 79  | 4F  | O     | 95  | 5F  | _     |

## ASCII character table

---

| Dec | Hex | ASCII | Dec | Hex | ASCII |
|-----|-----|-------|-----|-----|-------|
| 96  | 60  | `     | 112 | 70  | p     |
| 97  | 61  | a     | 113 | 71  | q     |
| 98  | 62  | b     | 114 | 72  | r     |
| 99  | 63  | c     | 115 | 73  | s     |
| 100 | 64  | d     | 116 | 74  | t     |
| 101 | 65  | e     | 117 | 75  | u     |
| 102 | 66  | f     | 118 | 76  | v     |
| 103 | 67  | g     | 119 | 77  | w     |
| 104 | 68  | h     | 120 | 78  | x     |
| 105 | 69  | i     | 121 | 79  | y     |
| 106 | 6A  | j     | 122 | 7A  | z     |
| 107 | 6B  | k     | 123 | 7B  | {     |
| 108 | 6C  | l     | 124 | 7C  |       |
| 109 | 6D  | m     | 125 | 7D  | }     |
| 110 | 6E  | n     | 126 | 7E  | ~     |
| 111 | 6F  | o     | 127 | 7F  | DEL   |