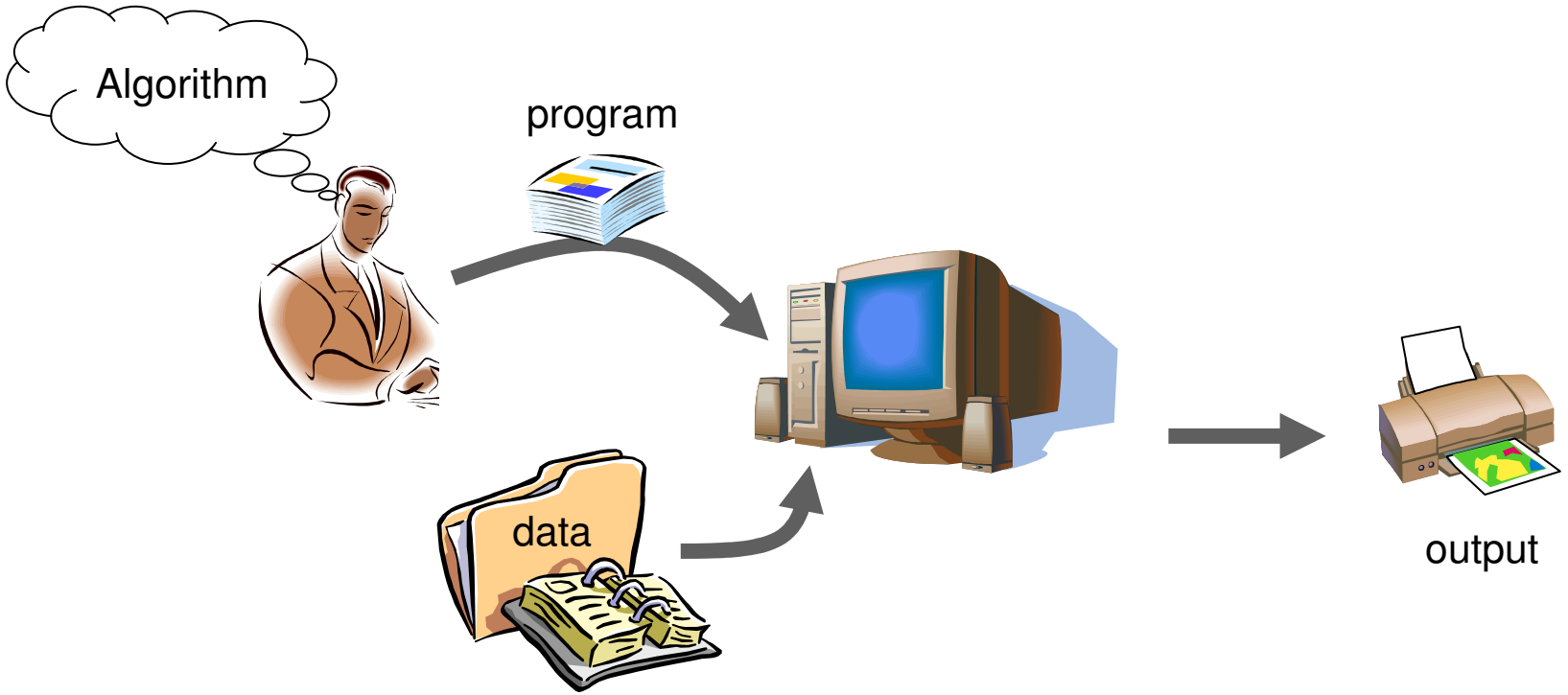
Programming Languages

National Chiao Tung University
Chun-Jen Tsai
05/4/2012

Programming Language

- Programming Language (PL) is a language that can "precisely describe" an algorithm to a computer so that it can execute the algorithm:



Design Considerations

- There are two extremes in designing a PL:
  - Use human language
  - Use machine code
- Human languages as programming languages
  - Imprecise
  - Inefficient (for computer as well as human)
  - Easy to use
  - Hard to debug
- Machine instruction code as programming languages
  - Precise
  - Efficient for computers
  - Verbose to use
  - Hard to debug

Assembly Language

- Since machine codes are too hard to remember, each processor manufacture designs an “easy-to-remember” names for each op-code
- Assembly language – a mnemonic system for representing machine instruction codes
  - Mnemonic names for op-codes
  - Names for all registers
  - Identifiers: descriptive names for memory locations, chosen by the programmers
- Assembly language is referred to as the 2nd generation of programming language

Assembly Language Characteristics

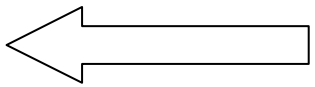
- One-to-one correspondence between machine instructions and assembly instructions
  - Programmer must think like the machine
- Inherently machine-dependent
- Before execution by a computer, we must translate a machine language program into machine codes by an assembler

Assembly Language Example

Machine language program

```
156C
166D
5056
30CE
C000
```

assembler



Assembly language program

```
LD   R5, [Price]
LD   R6, [ShippingCharge]
ADDI R0, R5 R6
ST   R0, [TotalCost]
HLT
;
ORG  6Ch
Price           db 25
ShippingCharge  db 5
TotalCost       db 00
```

Definition of mnemonics:

LD	means “load”
ADDI	means “Integer addition”
ST	means “store”
HLT	means “halt”
ORG	means “origin”
db	means “define byte”

Third Generation Languages

- Uses high-level primitives
- Machine independent (mostly)
- Early examples:
  - FORTRAN – for numerical computations
  - COBOL – for financial computations and database systems
- Each primitive corresponds to a short sequence of machine instruction codes
- Can be translated into machine codes by a ***compiler***

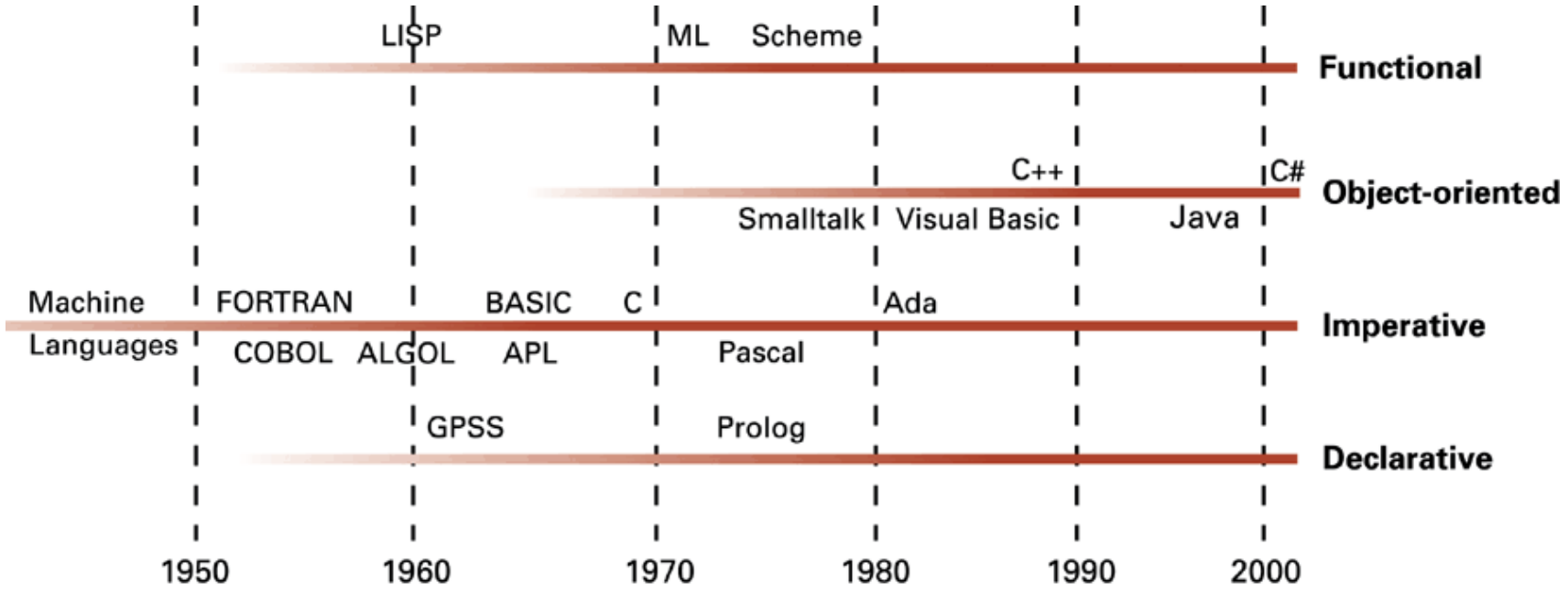
Language Translators

- There are several kinds of programming language translators
  - Assemblers
    - perform one-to-one mapping from assembly code to machine code
  - Compilers
    - perform translation from *a high-level (machine-independent) statement* to an equivalent *short sequence of machine codes*
  - Interpreters
    - perform translation and execution of high-level statements at the same time; note that there is no intermediate machine code being generated

Formal Languages

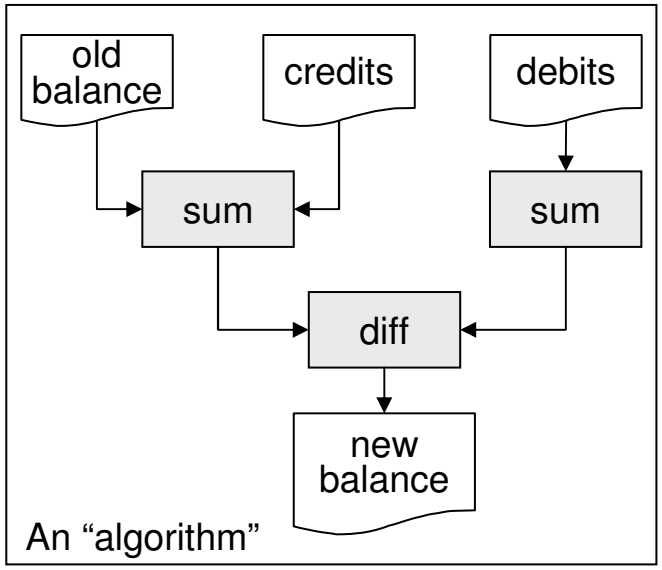
- Programming languages are “formal languages” since they are artificial languages defined precisely by grammars
- Natural (human) languages are not precisely defined by grammars, instead, grammars are created afterwards to “summarize” the language usage
  - Esperanto is an “formal” human language artificially developed in late 1870s.

Timeline of Programming Languages



Programming Paradigms (1/2)

- Imperative (procedural) programming language
  - A program is a sequence of commands
  - Earliest way of programming
- Functional programming language
  - A program is a description of a data flow (connections of functional units)



A program in LISP programming language:

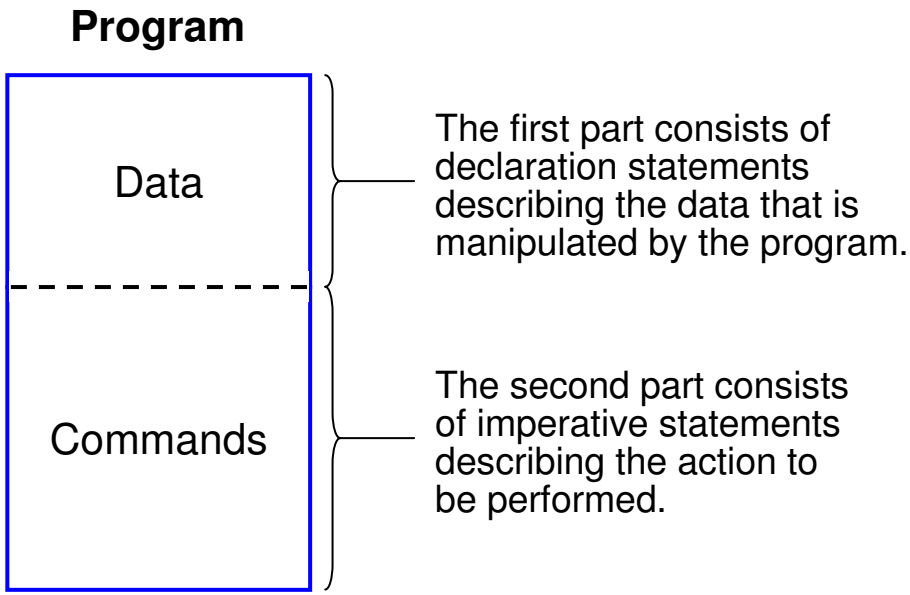
```
(diff (sum old_balance credits) (sum debits))
```

Programming Paradigms (2/2)

- Declarative programming language
  - Describes conditions that satisfy the intended solution; the specific steps needed to arrive at that solution are up to an unspecified interpreter
  - Only works for a specific domain of problems (e.g. for knowledge-based inference)
- Object-oriented programming language
  - A “data-centric” programming language
  - Operations are attached to data
  - A program is composed of a list of objects, each annotated by a list of permissible operations of that object

Imperative Programming Language

- The imperative programming paradigm is the most intuitive and effective way of expressing our commands to computers



Example of Data Declaration

- Variable (data) declarations in C, C++, C#, and Java are as follows:
  - Scalar data declaration:

```
float Length, Width;
int Price, Tax, Total;
char Symbol;
```

  - Aggregate data declarations:

```
int Scores[2][9];
```

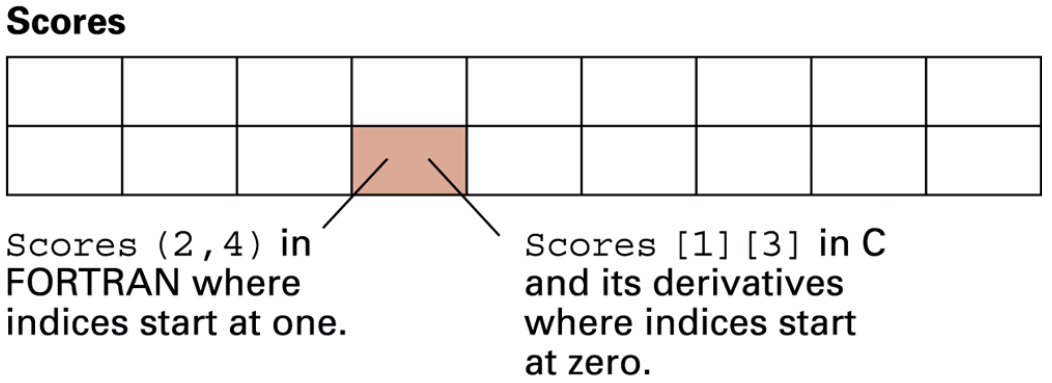
⇨ array

```
Struct {
    char Name[8];
    int Age;
    float SkillRating;
} Employee;
```

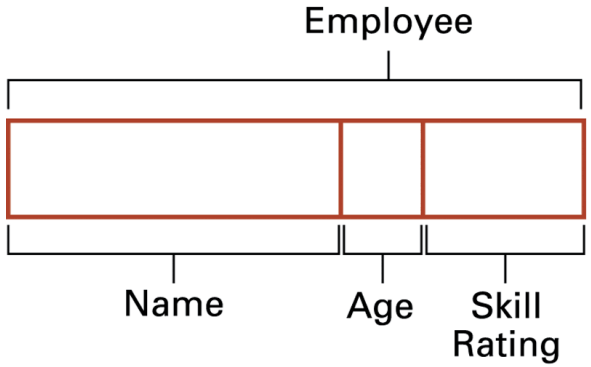
⇨ structure (heterogeneous array)

Memory Layout of Aggregate Data

- A two-dimensional array with two rows and nine columns:



- A structure:

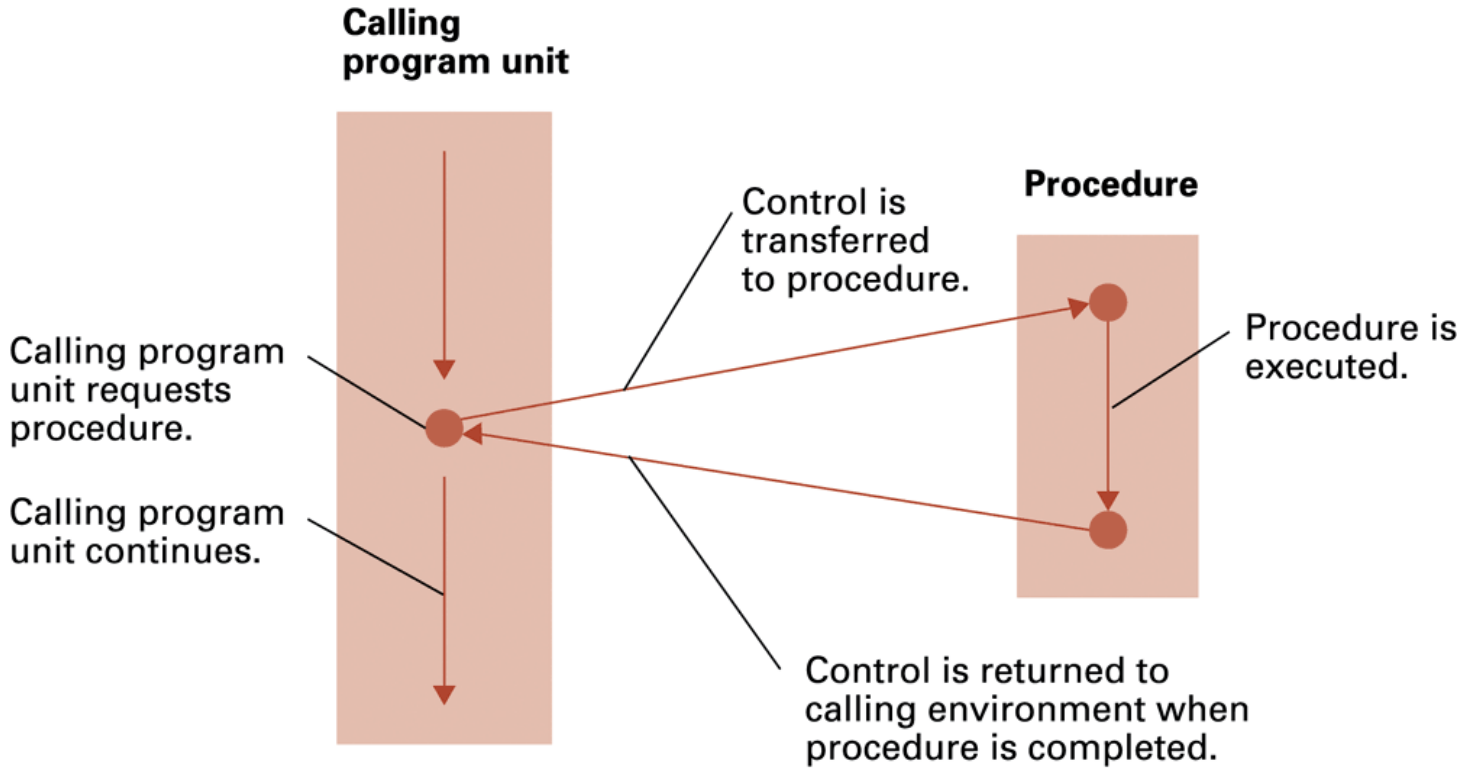


Elements of an Imperative PL

- An imperative programming language provides statements to:
  - Express constants and literals
  - Assign values to variables
  - Control the execution sequence of the program
    - Conditional control
    - Looping control
  - Commenting the program
  - Call procedural units

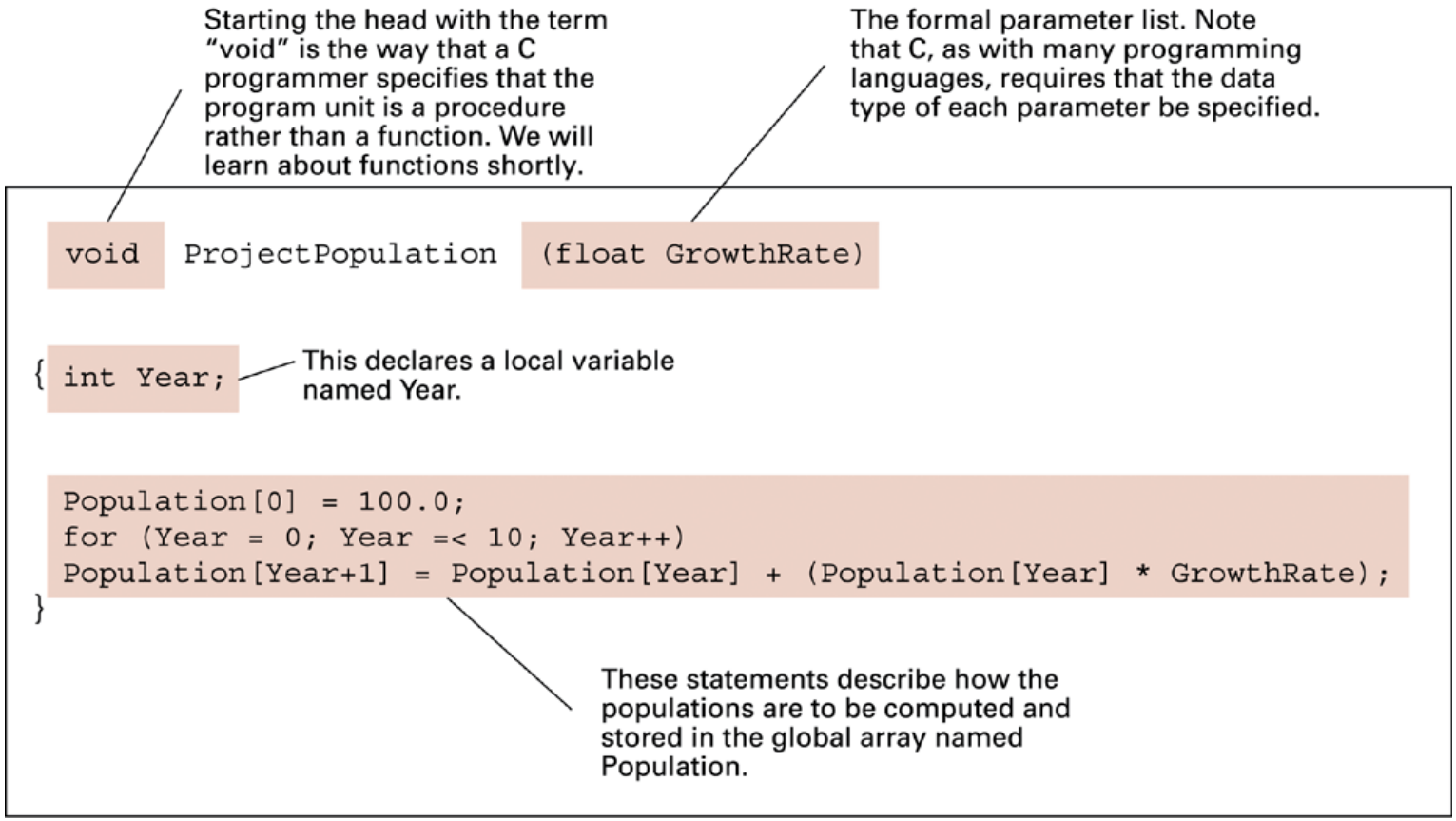
Procedural Calls (1/2)

- Procedural calls for imperative languages:



Procedural Calls (2/2)

- Description of a procedure in C:



Parameter Passing Methods

- There are several ways to pass a parameter from the calling program unit to the called procedure:
  - Call-by-value (passed by value in the textbook)
  - Call-by-reference (passed by reference in the textbook)
  - Call-by-name
    - not mentioned in the textbook, and not popular anymore
    - similar to macro expansion in C/C++, but it's a real function call

```
int x = 1, y = 2;

my_func()
{
     f1(x, x+y);
}

f1(p, q)
{
     int s;
     p = q;
     s = q;
}
```

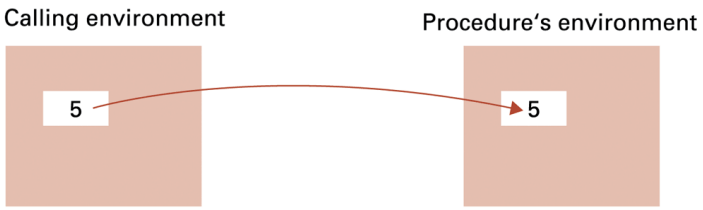
This is equal to `x = x+y;` and 3 will be assigned to **`p`** and **`x`**.

Here, 5 will be assigned to **`s`**.

Call by Value & Call by Reference

- Call by value

a. When the procedure is called, a copy of the data is given to the procedure

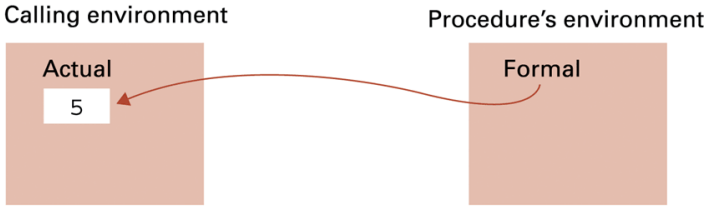


b. and the procedure manipulates its copy.

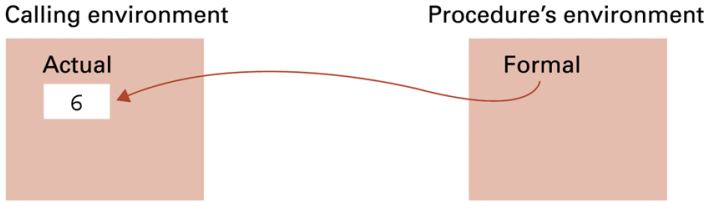
c. Thus, when the procedure has terminated, the calling environment has not been changed.

- Call by reference

a. When the procedure is called, the formal parameter becomes a reference to the actual parameter.



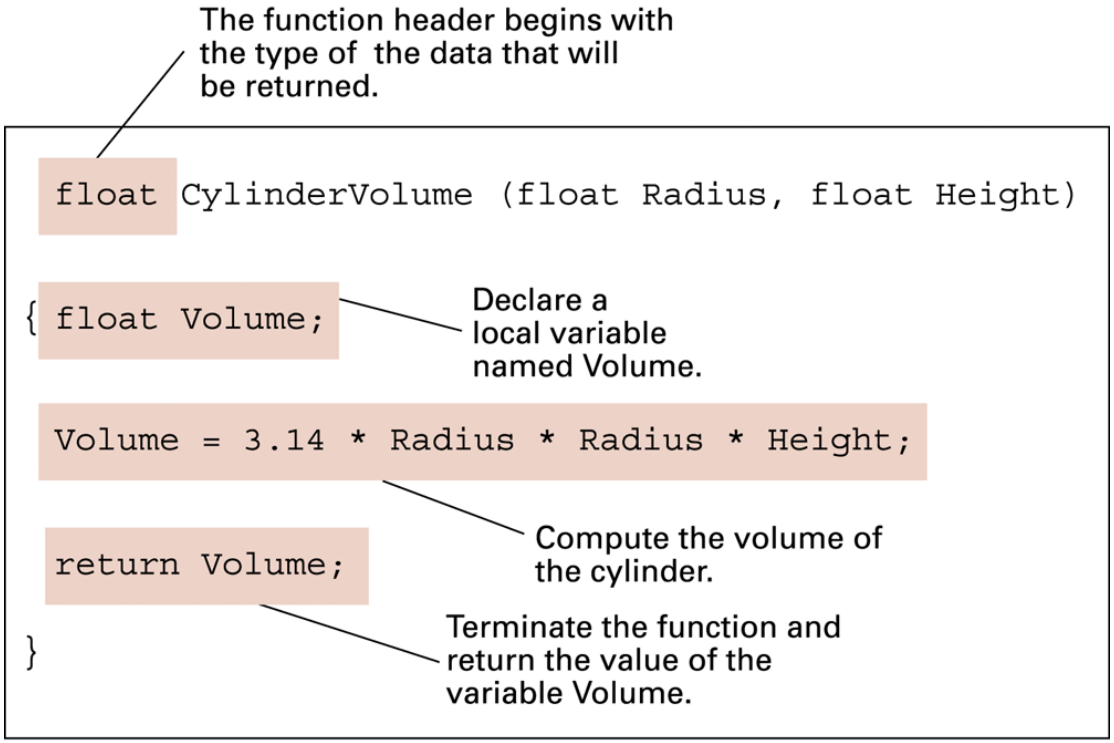
b. Thus, changes directed by the procedure are made to the actual parameter



c. and are, therefore, preserved after the procedure has terminated.

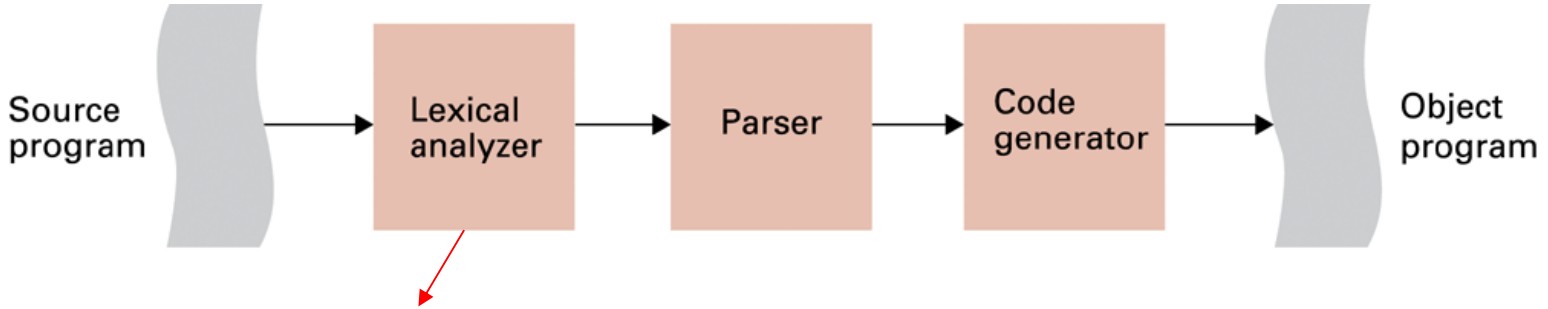
Function Calls

- A function is a special type of procedure that returns a value:



Translating Program to Executable

- A compiler translates a program into machine codes via the following steps:



Lexical analyzer converts alpha-numerical symbols in the source program to tokens; for example, if each token is specified by a 16-bit number, a lexical analyzer may perform the following conversion:

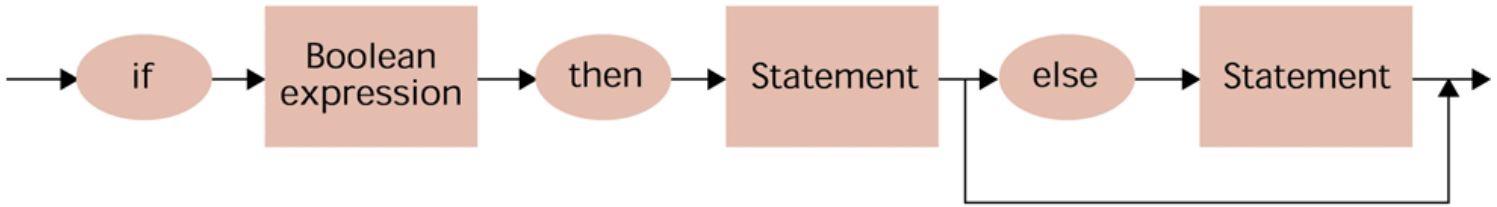
```
position = x_coord + y_coord * 7 → 0003 1001 0001 1002 0002 1003 2001
```

The first byte specifies the type of token, 0 – variables, 1 – operators, 2 – constants

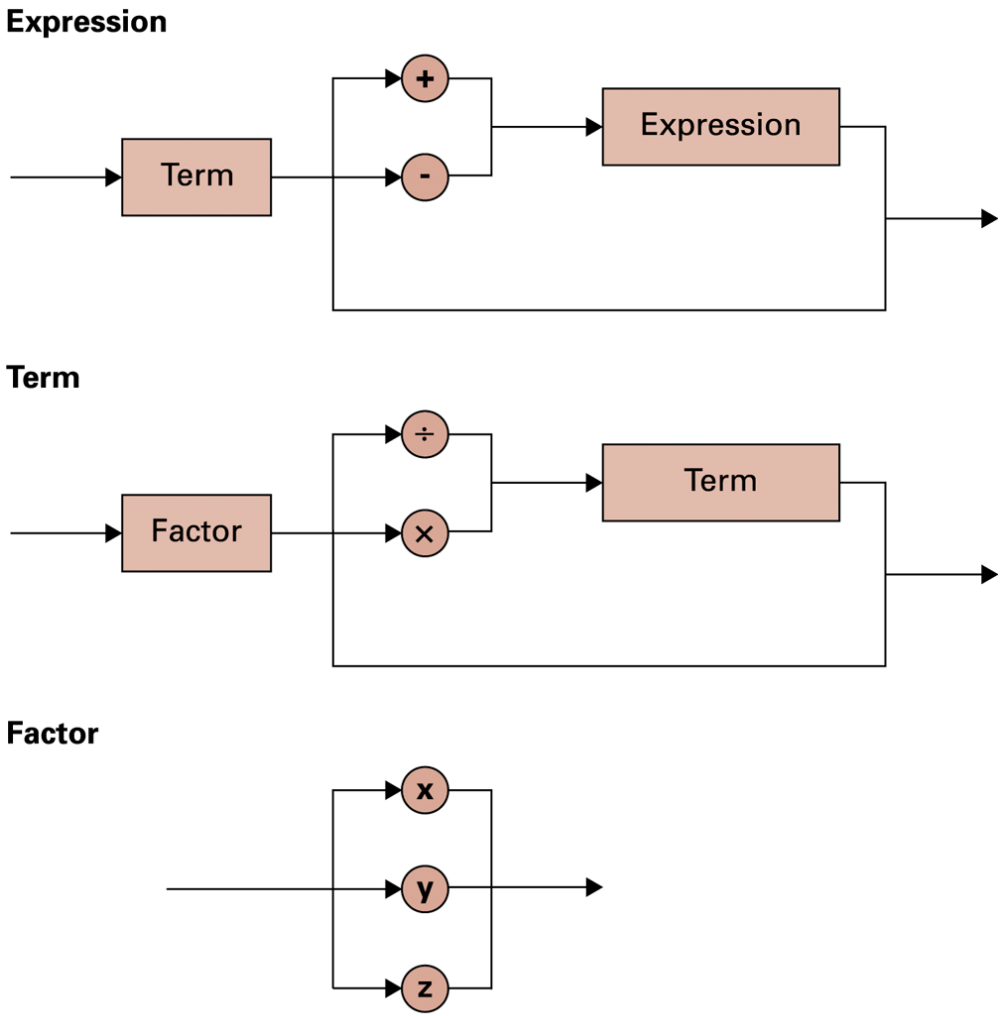
The remaining bytes compose an index to the token value tables

Syntax Diagram

- The parsing process is based on a set of rules that define the syntax of the programming language
  - The rules are called grammar
  - The rules can be expressed by syntax diagrams
- A syntax diagram of the “if-then-else” statement is as follows:

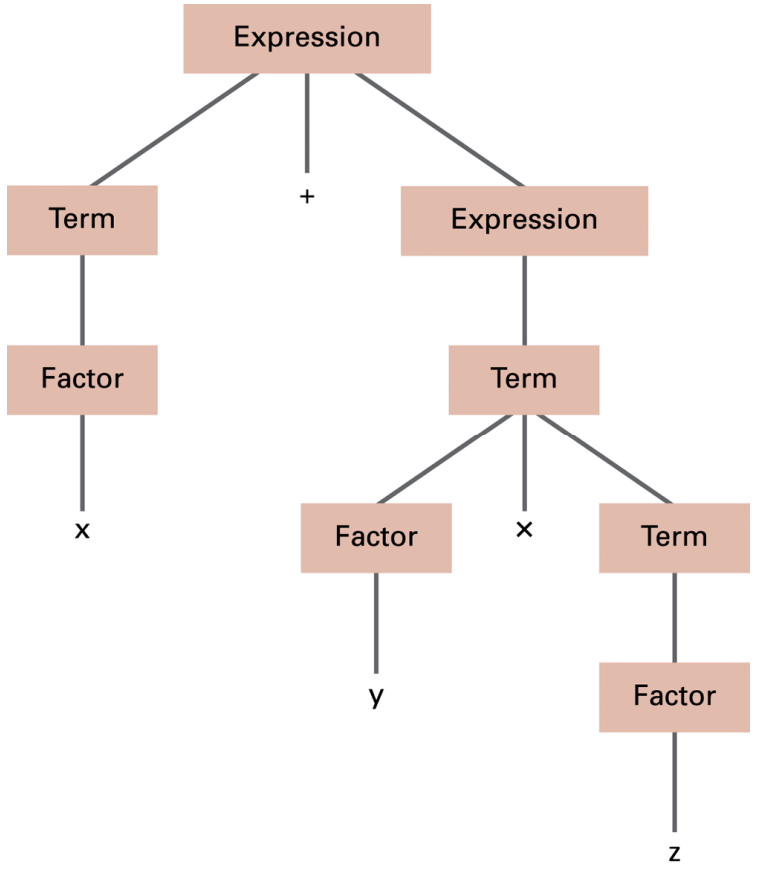


Algebraic Expression Syntax



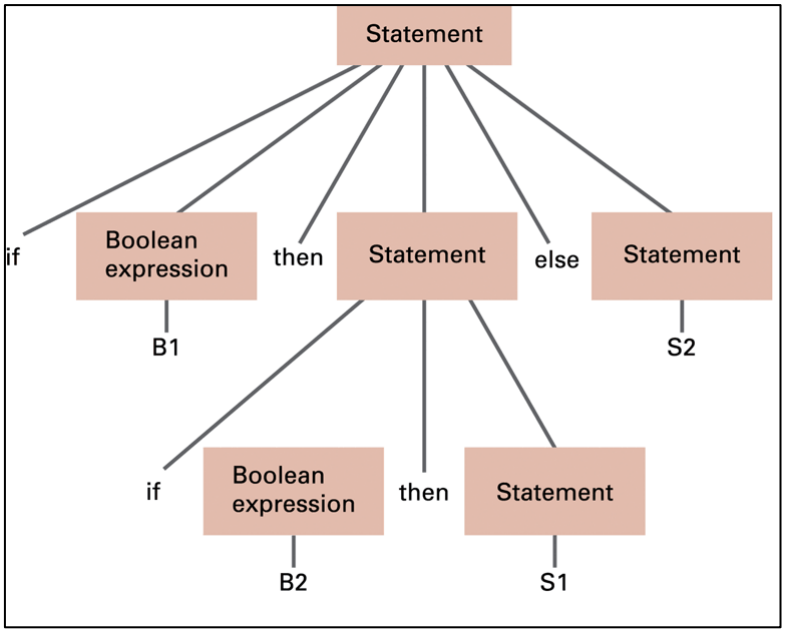
Example of Parsing An Expression

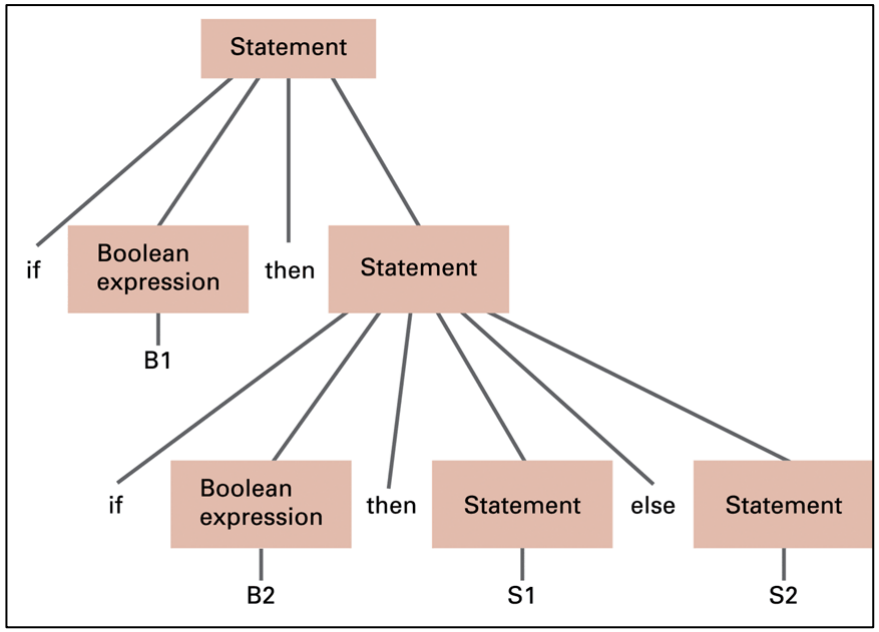
- The parser generates a parse tree for a statements $x + y \times z$:



Ambiguous Parse Trees

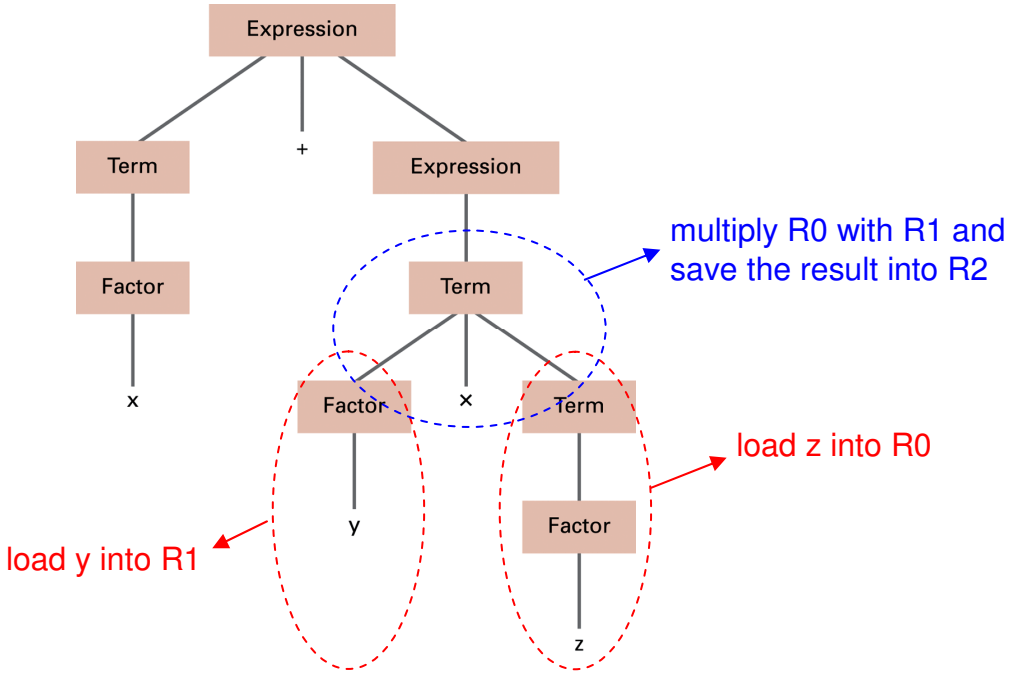
- For “if B1 then if B2 then S1 else S2” we could have two possible parse trees:





Code Generation and Optimization

- Once the parse tree is done, one must generate machine codes for each sub-tree or node, for example, in bottom-up manner

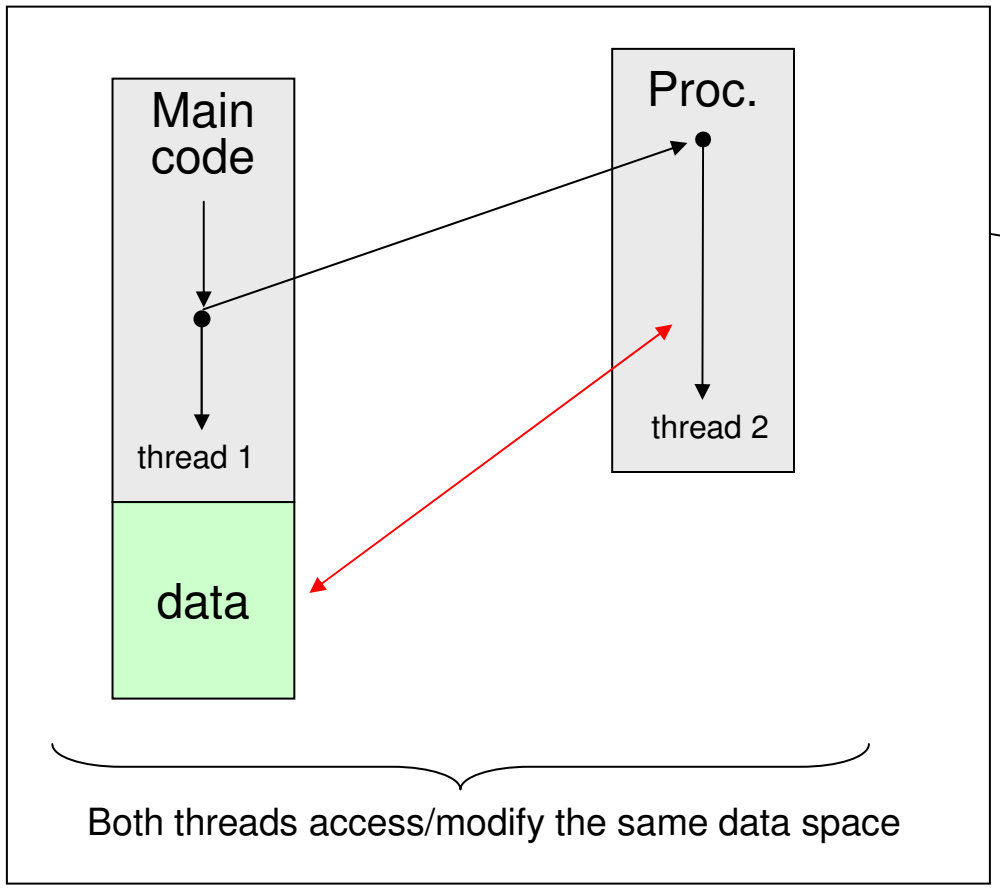


- Code optimization is a technique for finding the best way to generate codes

Concurrent Programming

- Concurrent programming is the simultaneous execution of multiple processes/threads
  - If the computing system has only one CPU, simultaneous execution can be simulated using time-sharing techniques
  - If the computing system has multiple CPUs, each process/threads will be assigned to one CPU for execution
- The difference between processes and threads can be loosely defined as follows:
  - Program (static) $\rightarrow$ Process (runtime)
  - Procedure (static) $\rightarrow$ Thread (runtime)

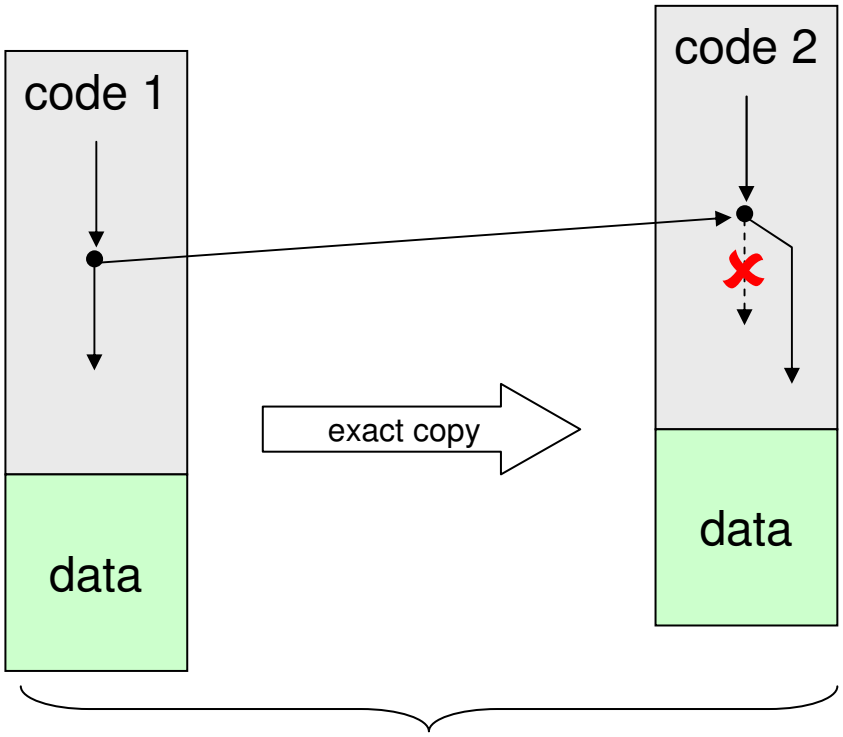
Spawning A Thread (in a Process)



This whole thing is still considered as a single process space

Spawning A Process

- Spawning (or forking) of a process is done as follows:

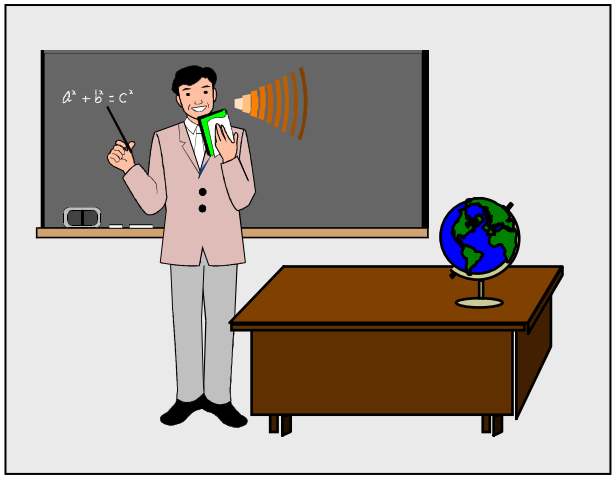


Both processes are running in the main memory at the same time

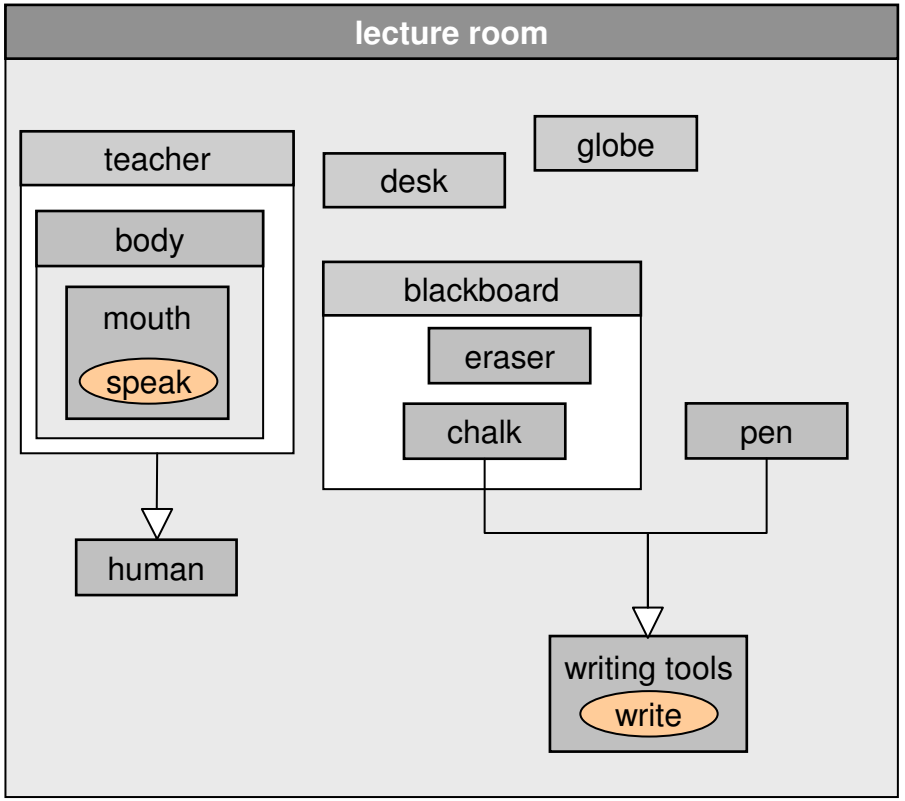
Object-Oriented Programming

- An object-oriented (OO) language composed of a hierarchical structure of objects
  - Class: the static definition of an objects
  - Object: an active substance inside a running process
- An OO program is composed of the declaration of different types of static description of substances (i.e. classes), and how these substances are created (become active) and interact with each others
- In OO terminology, an object is an instance of a class

OO View of Physical World



An object-oriented
description of the lecturing process



Object-Oriented Terminologies

- Data Encapsulation
  - Access to the internal components of an object are restricted
  - You can use an object, but you cannot modify its behavior and internal data
- Inheritance
  - Define new classes in terms of previously defined classes
  - Facilitate hierarchical structure of an object-oriented process
- Polymorphism
  - Implementation details of the behaviors (or operators) of an object are interpreted by the object that perform that behavior

Functional Programming

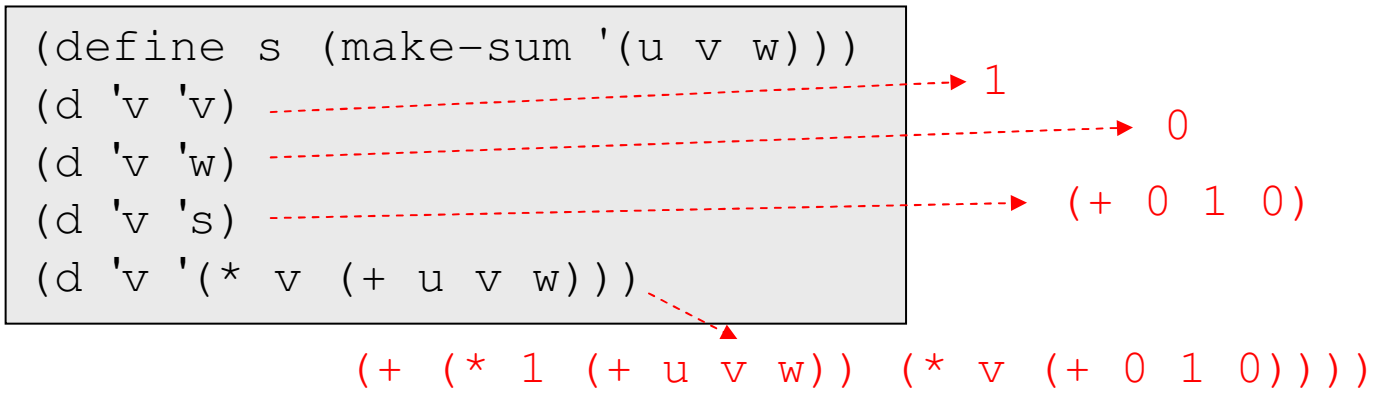
- Principle of functional programming:
  - The value of an expression depends only on the values of its sub-expressions, if any
- Any language must be defined in some sort of notation, called meta-language or defining language
  - Meta-language tends to be a functional description
- Functional programming becomes popular due to the invention of LISP, a list processing language, by John McCarthy in 1958

Features of a Functional Language

- In functional language, program and data can be treated almost the same:
  - (it seems that you liked me)
  - Unification of code and data is an important concept in many modern languages
- Lots of parentheses are used to modify the structure of a program:
  - (it seems that you liked me)
    and
    ((it seems that) you (liked) me)
    are different
  - Some people jokingly call LISP: Lots of Silly Parentheses

Example: Differentiation

- Differentiation can be computed in LISP as follows:



- The function “d” is defined using the rules:

```
d(x, x) = 1
d(x, not x) = 0
d(x, E1 + E2) = d(x, E1) + d(x, E2)
d(x, E1 * E2) = d(x, E1)*E2 + E1 * d(x, E2)
```

Declarative Programming

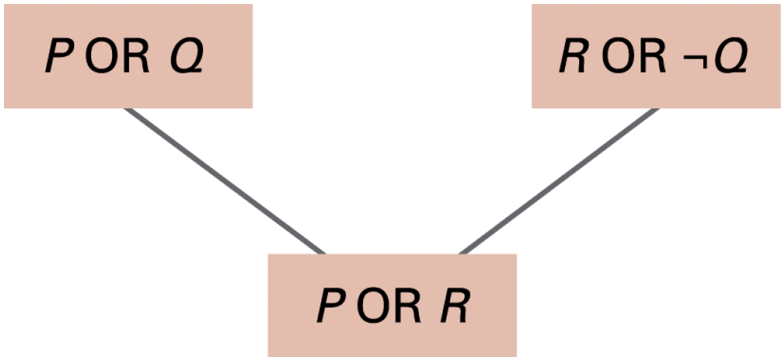
- Declarative programming is also referred to as Logic programming:
  - The use of facts and rules to represent information
  - The use of deduction to answer queries
- In declarative programming, the programmer supplies facts and rules; while the computer use deduction to find the answer
- The language that makes declarative programming well-known is Prolog, developed in 1972
  - The application domain for Prolog is similar to that for LISP: artificial intelligence, expert systems, etc.

Prolog Language Elements

- In Prolog, all statements must be facts or rules
- Fact:
  - *predicateName*(*arguments*)
  - Example: `parent(Bill, Mary)`
- Rule:
  - *conclusion* `:-` *premise* (note that `:-` stands for “if”)
  - Example: `wise(x) :- old(x)`
  - Example: `faster(x,z) :- faster(x,y),faster(y,z)`

Deduction Methods

- Resolution
  - Combining two or more statements to produce a new, logically equivalent statement



- Unification
  - Assigning a value to a variable in a statement

Example of Deduction

- Resolving the statements:
  (P OR Q), (R OR ¬Q), ¬R, ¬P