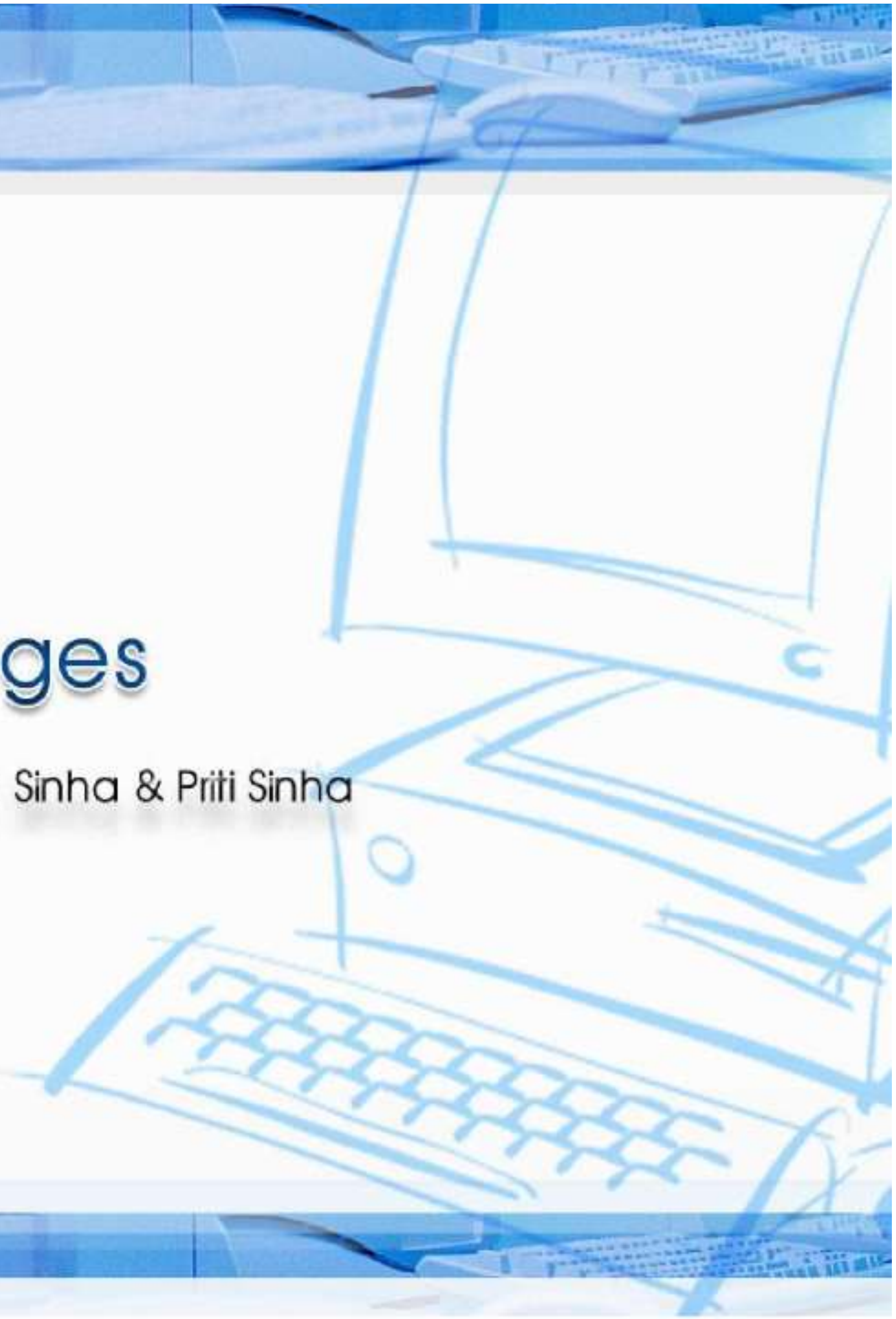


Chapter 12

Computer Languages

Computer Fundamentals - Pradeep K. Sinha & Priti Sinha

Learning Objectives

In this chapter you will learn about:

- Computer languages or programming languages
- Three broad categories of programming languages – machine, assembly, and high-level languages
- Commonly used programming language tools such as assembler, compiler, linker, and interpreter
- Concepts of object-oriented programming languages
- Some popular programming languages such as FORTRAN, COBOL, BASIC, Pascal, C, C++, C#, Java, RPG, LISP and SNOBOL
- Related concepts such as Subprogram, Characteristics of a good programming language, and factors to consider while selecting a language for coding an application

Broad Classification of Computer Languages

- Machine language
- Assembly language
- High-level language

Machine Language

- Only language of a computer understood by it without using a translation program
- Normally written as strings of binary 1s and 0s
- Written using decimal digits if the circuitry of the computer being used permits this

A Typical Machine Language Instruction Format

OPCODE (operation code)	OPERAND (Address/Location)

- OPCODE tells the computer which operation to perform from the instruction set of the computer
- OPERAND tells the address of the data on which the operation is to be performed

A Sample Machine Language Program

001000000000001100111001	10001471
001100000000010000100001	14002041
011000000000011100101110	30003456
101000111111011100101110	50773456
000000000000000000000000	00000000
In Binary (Difficult to read and understand)	In Decimal (Easier to read and understand)

Advantages & Limitations of Machine Language

Advantage

- Can be executed very fast

Limitations

- Machine Dependent
- Difficult to program
- Error prone
- Difficult to modify

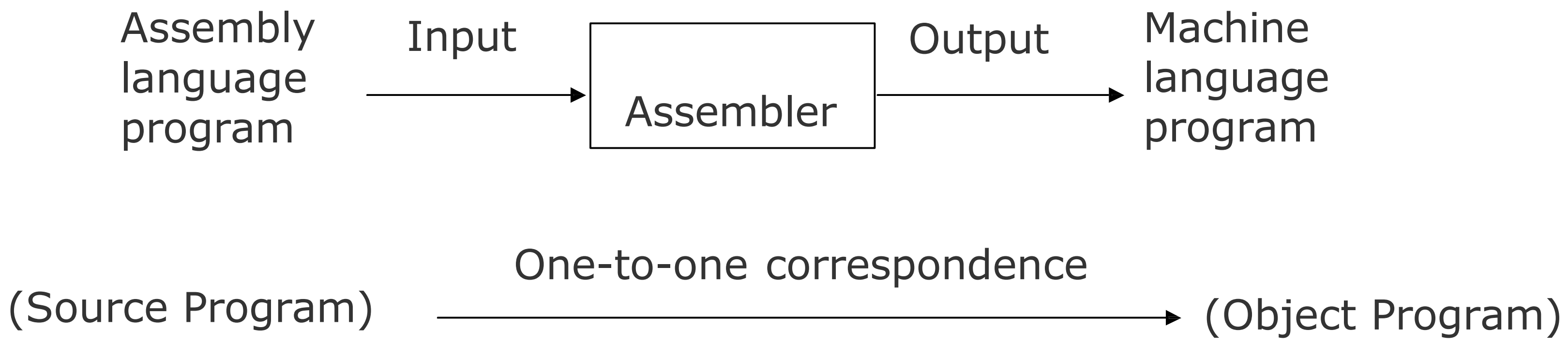
Assembly/Symbolic Language

Programming language that overcomes the limitations of machine language programming by:

- Using alphanumeric mnemonic codes instead of numeric codes for the instructions in the instruction set
  e.g. using ADD instead of 1110 (binary) or 14 (decimal) for instruction to add
- Allowing storage locations to be represented in form of alphanumeric addresses instead of numeric addresses
  e.g. representing memory locations 1000, 1001, and 1002 as FRST, SCND, and ANSR respectively
- Providing pseudo-instructions that are used for instructing the system how we want the program to be assembled inside the computer's memory
  e.g. START PROGRAM AT 0000; SET ASIDE AN ADRESS FOR FRST

Assembler

- Software that translates as assembly language program into an equivalent machine language program of a computer



An Example of Assembly Language Program

Mnemonic	Opcode	Meaning
HLT	00	Halt, used at the end of program to stop
CLA	10	Clear and add into A register
ADD	14	Add to the contents of A register
SUB	15	Subtract from the contents of A register
STA	30	Store A register

A subset of the set of instructions supported by a computer

An Example of Assembly Language Program

```
START  PROGRAM AT 0000
START  DATA AT 1000
SET ASIDE AN ADDRESS FOR FRST
SET ASIDE AN ADDRESS FOR SCND
SET ASIDE AN ADDRESS FOR ANSR
CLA FRST
ADD SCND
STA ANSR
HLT
```

Sample assembly language program for adding two numbers and storing the result

An Example of Assembly Language Program

Symbolic name	Memory location
FRST	1000
SCND	1001
ANSR	1002

Mapping table set up by the assembler for the data items of the assembly language program

An Example of Assembly Language Program

Memory location	Contents		Comments
	Opcode	Address	
0000	10	1000	Clear and add the number stored at FRST to A register
0001	14	1001	Add the number stored at SCND to the contents of A register
0002	30	1002	Store the contents of A register into ANSR
0003	00		Halt
-			
-			
-			
1000			Reserved for FRST
1001			Reserved for SCND
1002			Reserved for ANSR

Equivalent machine language program for the assembly language program

Advantages of Assembly Language Over Machine Language

- Easier to understand and use
- Easier to locate and correct errors
- Easier to modify
- No worry about addresses
- Easily relocatable
- Efficiency of machine language

Limitations of Assembly Language

- Machine dependent
- Knowledge of hardware required
- Machine level coding

Typical Uses of Assembly Language

- Mainly used today to fine-tune important parts of programs written in a high-level language to improve the program's execution efficiency

Assembly Languages with Macro Instructions

- Any assembly language instruction that gets translated into several machine language instructions is called a **macro instruction**
- Several assembly languages support such macro instructions to speed up the coding process
- Assemblers of such assembly languages are designed to produce multiple machine language instructions for each macro instruction of the assembly language

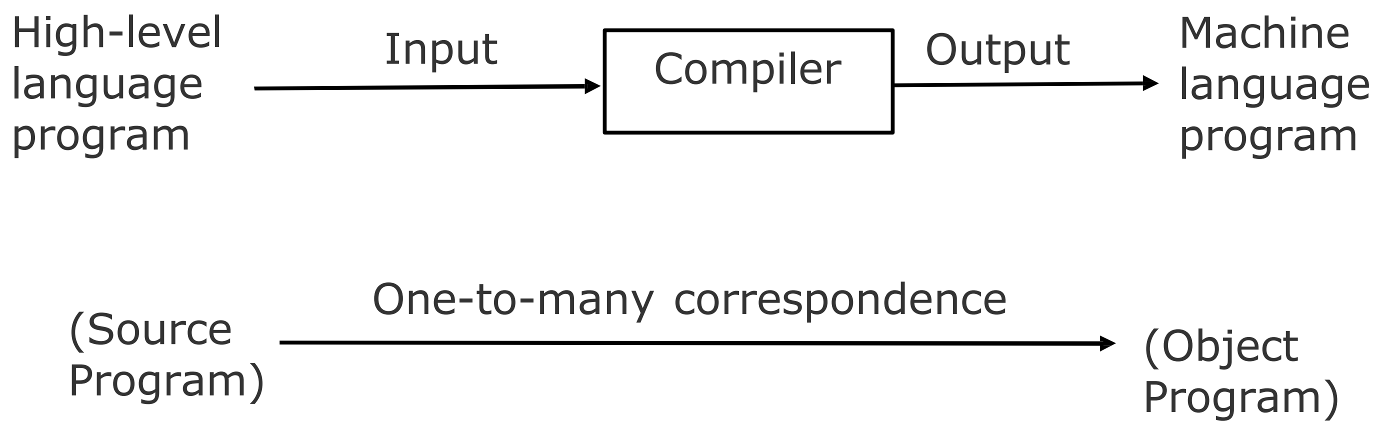
High-Level Languages

- Machine independent
- Do not require programmers to know anything about the internal structure of computer on which high-level language programs will be executed
- Deal with high-level coding, enabling the programmers to write instructions using English words and familiar mathematical symbols and expressions

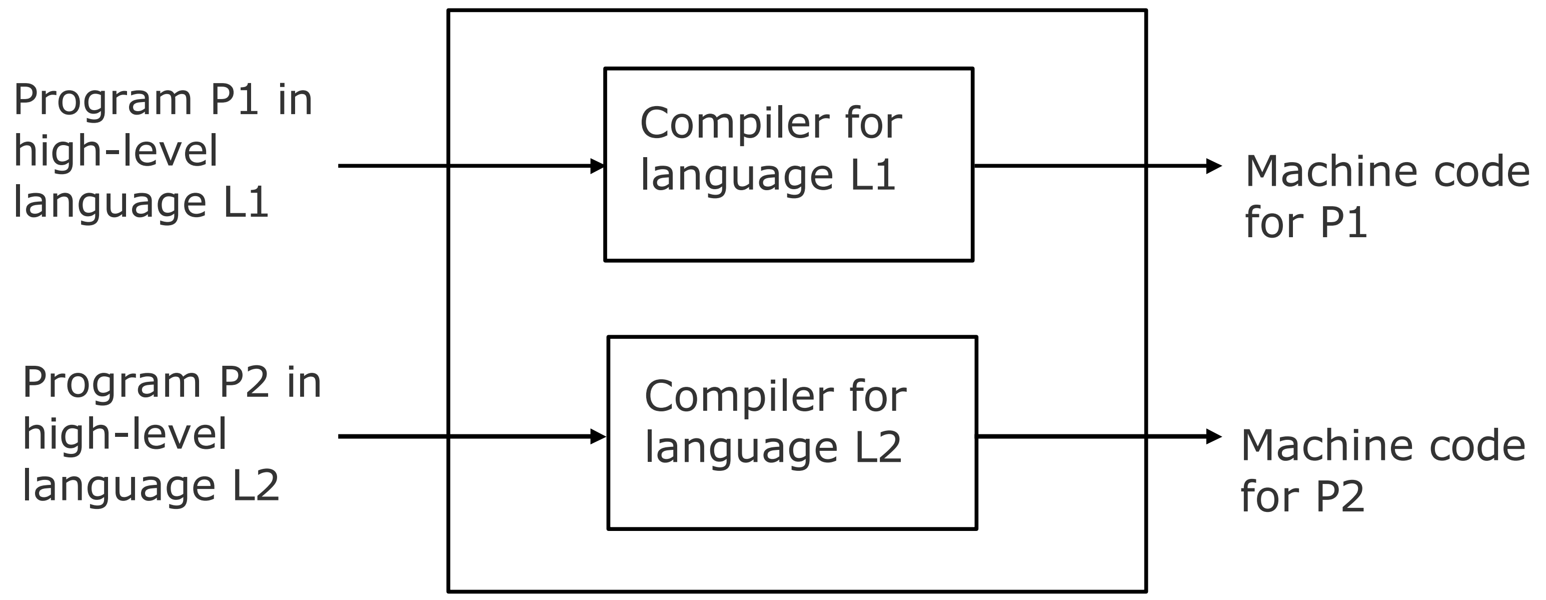
Compiler

- Translator program (software) that translates a high-level language program into its equivalent machine language program
- Compiles a set of machine language instructions for every program instruction in a high-level language

Compiler


High-level language program
Input
Compiler
Output
Machine language program
(Source Program)
One-to-many correspondence
(Object Program)

Compiler



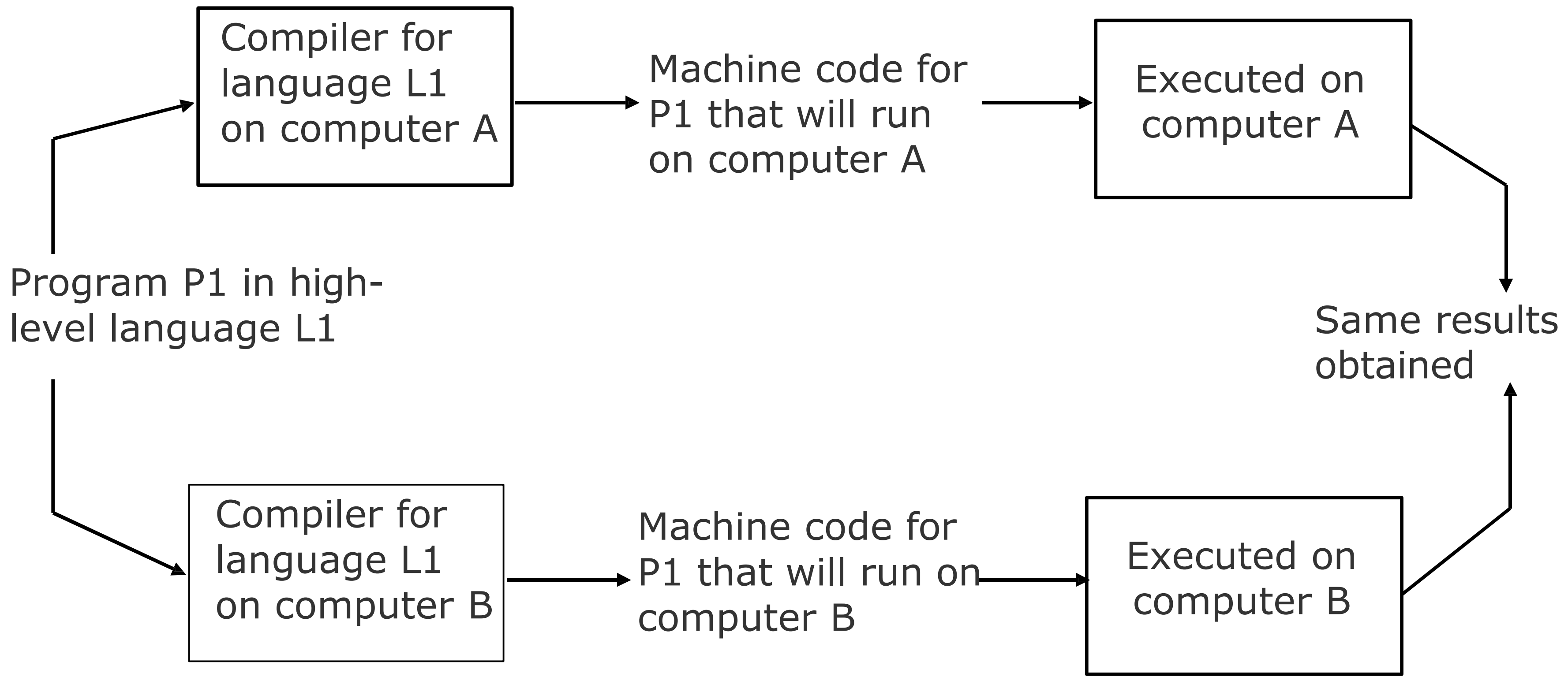
A computer supporting languages L1 and L2

Illustrating the requirement of a separate compiler for each high-level language supported by a computer

(Continued on next slide)

Compiler

(Continued from previous slide..)


Compiler for language L1 on computer A
Machine code for P1 that will run on computer A
Executed on computer A
Program P1 in high-level language L1
Same results obtained
Compiler for language L1 on computer B
Machine code for P1 that will run on computer B
Executed on computer B

Illustrating the machine independence characteristic of a high-level language. Separate compilers are required for the same language on different computers

Syntax Errors

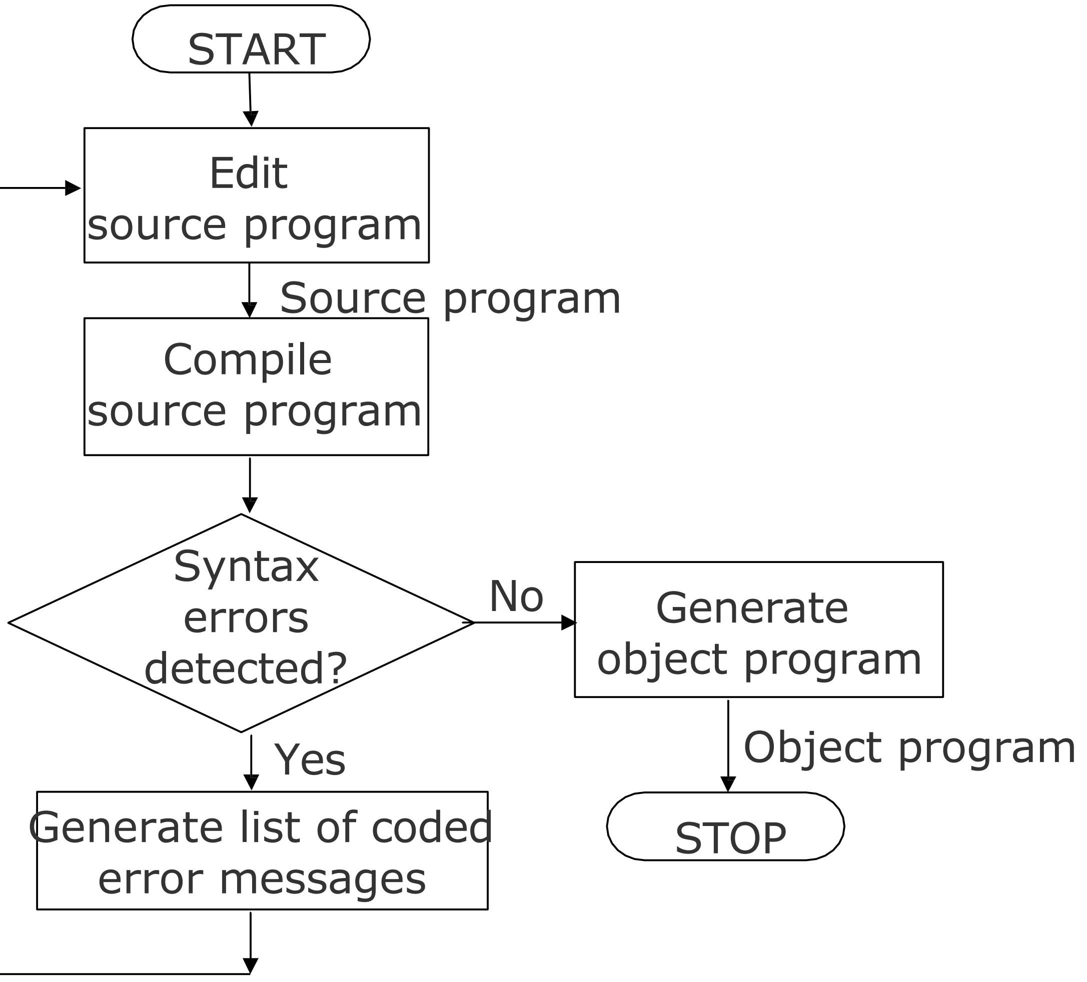
In addition to doing translation job, compilers also automatically detect and indicate syntax errors

Syntax errors are typically of following types:

- Illegal characters
- Illegal combination of characters
- Improper sequencing of instructions in a program
- Use of undefined variable names

Note : A compiler cannot detect logic errors in a program

The Process of Removing Syntax Errors From A Source Program



Linker

- For a large software, storing all the lines of program code in a single source file will be:
  - Difficult to work with
  - Difficult to deploy multiple programmers to concurrently work towards its development
  - Any change in the source program would require the entire source program to be recompiled

- Hence, a modular approach is generally adapted to develop large software where the software consists of multiple source program files
- No need to write programs for some modules as it might be available in library offering the same functionality

(Continued on next slide)

Linker

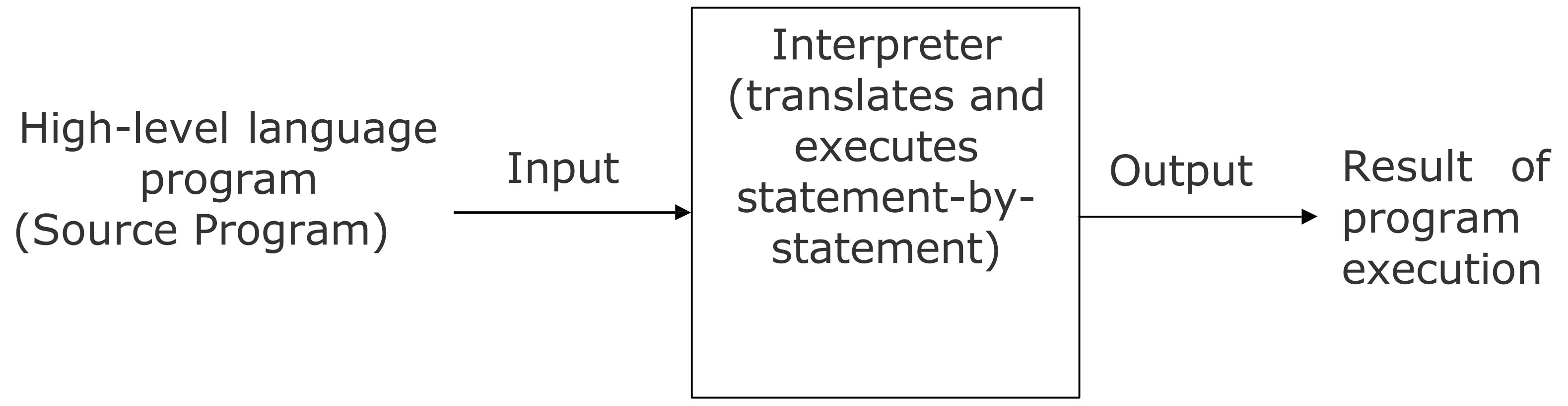
(Continued from previous slide..)

- Each source program file can be independently modified and compiled to create a corresponding object program file
- Linker program (software) is used to properly combine all the object program files (modules)
- Creates the final executable program (load module)

Interpreter

- Interpreter is a high-level language translator
- Takes one statement of a high-level language program, translates it into machine language instructions
- Immediately executes the resulting machine language instructions
- Compiler simply translates the entire source program into an object program and is not involved in its execution

Role of an Interpreter


High-level language program (Source Program)
Input
Interpreter (translates and executes statement-by-statement)
Output
Result of program execution

Intermediate Language Compiler & Interpreter

- New type of compiler and interpreter combines the speed, ease, and control of both compiler and interpreter
- Compiler first compiles the source program to an *intermediate* object program
- Intermediate object program is not a machine language code but written in an intermediate language that is virtually machine independent
- Interpreter takes intermediate object program, converts it into machine language program and executes it

Benefits of Intermediate Language Compiler & Interpreter

- Intermediate object program is in compiled form and thus is not original source code, so safer and easier to share
- Intermediate object program is based on a standard Intermediate Definition Language (IDL)
- Interpreter can be written for any computer architecture and operating system providing virtual machine environment to the executing program
- Newer Interpreter compiles intermediate program, in memory, into final host machine language program and executes it
- This technique is called *Just-In-Time (JIT) Compilation*

Advantages of High-Level Languages

- Machine independent
- Easier to learn and use
- Fewer errors during program development
- Lower program preparation cost
- Better documentation
- Easier to maintain

Limitations of High-Level Languages

- Lower execution efficiency
- Less flexibility to control the computer’s CPU, memory and registers

Object-Oriented Programming Languages

- Programming languages are used for simulating real-world problems on computers
- Much of the real world is made up of objects
- Essence of OOP is to solve a problem by:
  - Identifying the real-world objects of the problem
  - Identifying processing required of them
  - Creating simulations of objects, processes, and their communications

FORTRAN

- Stands for **FOR**mula **TRAN**slation
- Originally developed by John Backus and his team at IBM followed by several revisions
- Standardized by ANSI as FORTRAN-77 and FORTRAN-90
- Designed for solving scientific & engineering problems
- Oriented towards solving problems of a mathematical nature
- Popular language amongst scientists and engineers

A Sample FORTRAN Program

```
C   FORTRAN PROGRAM TO COMPUTE
C   THE SUM OF 10 NUMBERS
      SUM = 0
      DO 50 I = 1, 10
      READ (5, 10) N
      SUM = SUM + N
50    CONTINUE
      WRITE (6, 20) SUM
10    FORMAT (F6.2)
20    FORMAT (1X, 'THE SUM OF GIVEN NUMBERS = ',
      F10.2)
      STOP
      END
```

COBOL

- Stands for **CO**mmon **B**usiness **O**riented **L**anguage
- Originally developed started under Grace Hopper followed by COnference on DAta SYstems Languages (CODASYL)
- Standardized by ANSI as COBOL-74, COBOL-85, and COBOL-2002
- Designed for programming business data processing applications
- Designed to have the appearance and structure of a business report written in English, hence often referred to as a self-documenting language

A Sample COBOL Program

```
IDENTIFICATION DIVISION.
PROGRAM_ID. SUMUP.
AUTHOR. P K SINHA.
* THIS PROGRAM COMPUTES AND PRINTS
* THE SUM OF GIVEN NUMBERS.

ENVIROMENT DIVISION.
CONFIGURATION SECTION.
SOURCE_COMPUTER.   BURROUGHS_6700.
OBJECT_COMPUTER.   BURROUGHS_6700.
INPUT_OUTPUT SECTION.
FILE_CONTROL.
            SELECT DATA_FILE ASSIGN TO DISK.
            SELECT OUTPUT_FILE ASSIGN TO PRINTER.

DATA DIVISION.
FILE SECTION.
```

(Continued on next slide)

A Sample COBOL Program

(Continued from previous slide..)

```
FD DATA_FILE
            RECORD CONTAINS 80 CHARACTERS
            LABEL RECORD IS OMITTED
            DATA    RECORD    IS    INPUT_DATA_
RECORD.

01          INPUT_DATA_RECORD.
             05 NPICTURE 9(6)V99.
             05   FILLER PICTURE X(72).

FD          OUTPUT_FILE
             RECORD CONTAINS 132 CHARACTERS
             LABEL RECORD IS OMITTED
            DATA RECORD IS OUTPUT_RECORD.
```

(Continued on next slide)

A Sample COBOL Program

(Continued from previous slide..)

```
01          OUTPUT_RECORD.
             05 FILLER PICTURE X.
             05TITLE PICTURE X(25).
             05 SUM PICTURE 9(10)V99.
05 FILLER PICTURE X(94).
WORKING_STORAGE SECTION.
77MESSAGE PICTURE X(25)
VALUE IS “THE SUM OF GIVEN NUMBERS=”.

PROCEDURE DIVISION.
OPEN_FILES.
            OPEN INPUT DATA_FILE.
            OPEN OUTPUT OUTPUT_FILE.

INITIALIZATION.
            MOVE SPACES TO OUTPUT_RECORD.
            MOVE ZERO TO SUM.
```

(Continued on next slide)

A Sample COBOL Program

(Continued from previous slide..)

```
PROCESS_LOOP.
        READ DATA_FILE AT END GO TO PRINT_PARA.
        ADD N TO SUM.
        GO TO PROCESS_LOOP.

PRINT_PARA.
         MOVE MESSAGE TO TITLE.
         WRITE OUTPUT_RECORD.

END_OF_JOB.
         CLOSE DATA_FILE.
         CLOSE OUTPUT_FILE.
         STOP RUN.
```

BASIC

- Stands for **B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode
- Developed by Professor John Kemeny and Thomas Kurtz at Darmouth College in the United States
- Standardized by ANSI as BASIC-78
- Designed to be an interactive language and to use an interpreter instead of a compiler
- Simple to implement, learn and use language. Hence, it is a widely used language on personal computers
- Flexible and reasonably powerful language and can be used for both business and scientific applications

A Sample BASIC Program

```
5    REM PROGRAM TO COMPUTE
6    REM THE SUM OF 10 NUMBERS
10   LET S = 0
20    FOR I = 1 TO 10
30   READ N
40   LET S = S + N
50   NEXT I
60   PRINT "THE SUM OF GIVEN NUMBERS = "; S
70   DATA 4, 20, 15, 32, 48
80   DATA 12, 3, 9, 14, 44
90   END;
```

Pascal

- Named after the famous seventeenth-century French mathematician Blaise Pascal
- Developed by Professor Nicklaus Wirth of Federal Institute of Technology in Zurich
- Encourages programmers to write well-structured, modular programs, instills good program practices
- Recognized as an educational language and is used to teach programming to beginners
- Suitable for both scientific & business applications
- Has features to manipulate numbers, vectors, matrices, strings, sets, records, files, and lists

A Sample Pascal Program

```
PROGRAM SUMNUMS (INPUT, OUTPUT);
(* PROGRAM TO COMPUTE THE SUM OF 10 NUMBERS *)

(* DECLARATION OF VARIABLES *)
VAR SUM, N : REAL;
VAR I : INTEGER;

(* MAIN PROGRAM LOGIC STARTS HERE *)
BEGIN
    SUM := 0;
    FOR I := 1 TO 10 DO
    BEGIN
        READ (N);
        SUM := SUM + N;
    END;
    WRITELN (‘THE SUM OF GIVEN NUMBERS=’, SUM);
END;
```

C

- Developed in 1972 at AT&T's Bell laboratories, USA by Dennis Ritchie and Brian Kernighan
- Standardized by ANSI and ISO as C89, C90, C99
- High-level programming languages (mainly machine independence) with the efficiency of an assembly language
- Language of choice of programmers for portable systems software and commercial software packages like OS, compiler, spreadsheet, word processor, and database management systems

A Sample C Program

```
 /* PROGRAM TO COMPUTE THE SUM OF 10 NUMBERS */
/* Directives to include standard library and header */
#include <stdlib.h>
#include <stdio.h>
/* Main function starts here */
void main ( )
{
       /* Declaration of variables */
       float Sum = 0.0, N = 0.0;
       int Count = 0;
       for (Count = 0; Count < 10; Count++)
      {
            printf("\nGive a number:");
            scanf("%f", N);
            Sum += N;
      }
     printf("THE SUM OF GIVEN NUMBERS =   %f", &Sum);
}
```

C++

- Named C++ as ++ is increment operator and C language is incremented to its next level with C++
- Developed by Bjarne Stroustrup at Bell Labs in the early 1980s
- Contains all elements of the basic C language
- Expanded to include numerous object-oriented programming features
- Provides a collection of predefined classes, along with the capability of user-defined classes

Java

- Development started at Sun Microsystems in 1991 by a team led by James Gosling
- Developed to be similar to C++ with fewer features to keep it simple and easy to use
- Compiled code is machine-independent and developed programs are simple to implement and use
- Uses *just-in-time* compilation
- Used in embedded systems such as hand-held devices, telephones and VCRs
- Comes in two variants – Java Runtime Engine (JRE) and Java Software Development Kit (SDK)

C# (C Sharp)

- Object-oriented programming language developed by Anders Hejlsberg and released by Microsoft as part of Microsoft's .NET technology initiative
- Standardized by ECMA and ISO
- Syntactically and semantically very close to C++ and adopts various object-oriented features from both C++ and Java
- Compilers target the Common Language Infrastructure (CLI) implemented by Common Language Runtime (CLR) of .NET Framework
- CLR provides important services such as, memory management, exception handling, and security

RPG

- Stands for **R**eport **P**rogram **G**enerator
- Developed by IBM to meet customer requests for an easy and economic mechanism for producing reports
- Designed to generate the output reports resulting from the processing of common business applications
- Easier to learn and use as compared to COBOL
- Programmers use very detailed coding sheets to write specifications about input, calculations, and output

LISP

- Stands for **LIS**t **P**rocessing
- Developed in 1959 by John McCarthy of MIT
- Designed to have features for manipulating non-numeric data, such as symbols and strings of text
- Due to its powerful list processing capability, it is extensively used in the areas of pattern recognition, artificial intelligence, and for simulation of games
- Functional programming language in which all computation is accomplished by applying functions to arguments

SNOBOL

- Stands for **S**tri**N**g **O**riented sym**BO**lic **L**anguage
- Used for non-numeric applications
- Powerful string manipulation features
- Widely used for applications in the area of text processing

Characteristics of a Good Programming Language

- Simplicity
- Naturalness
- Abstraction
- Efficiency
- Structured Programming Support
- Compactness
- Locality
- Extensibility
- Suitability to its environment

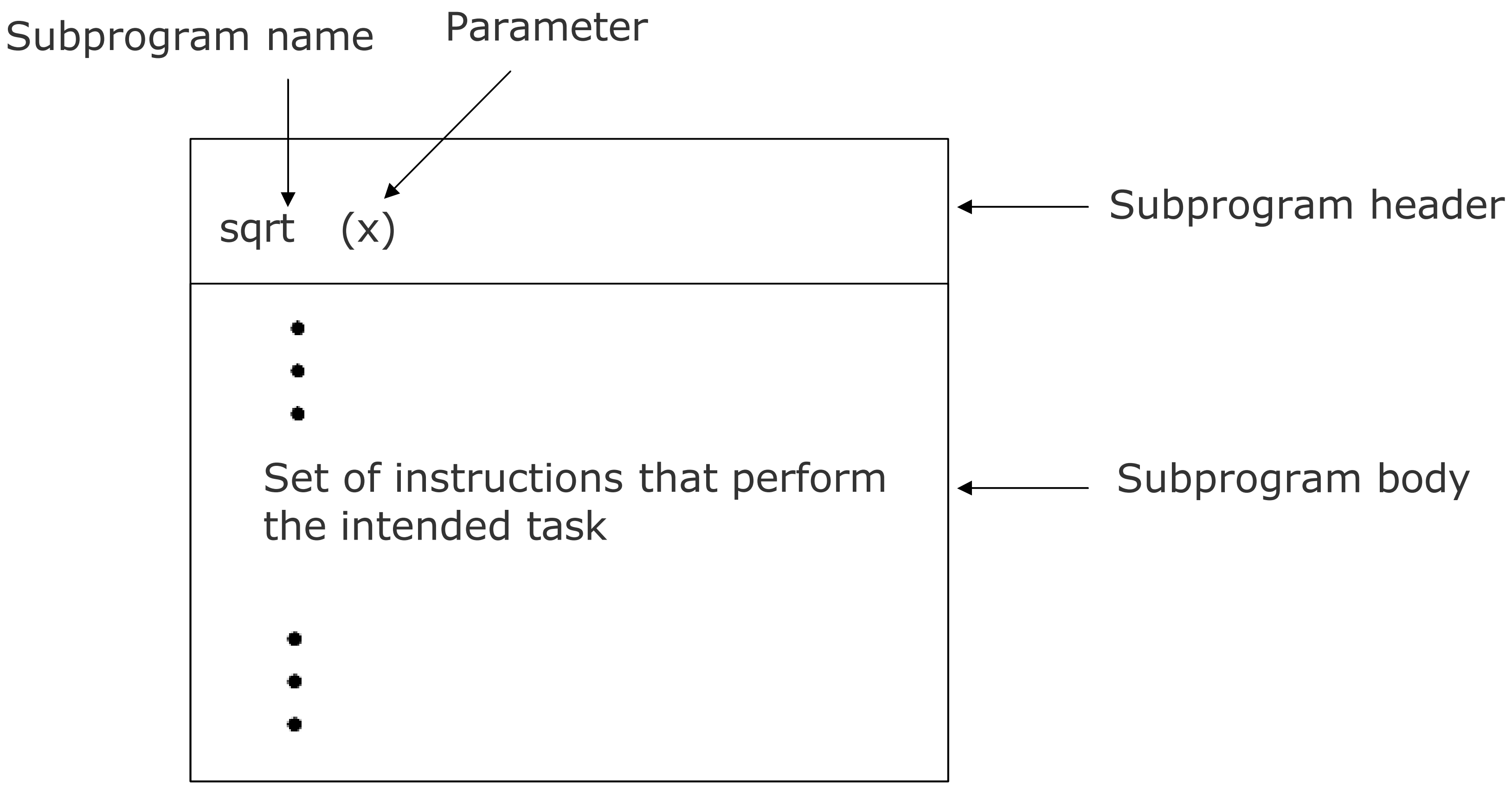
Factors for Selecting a Language for Coding an Application

- Nature of the application
- Familiarity with the language
- Ease of learning the language
- Availability of program development tools
- Execution efficiency
- Features of a good programming language

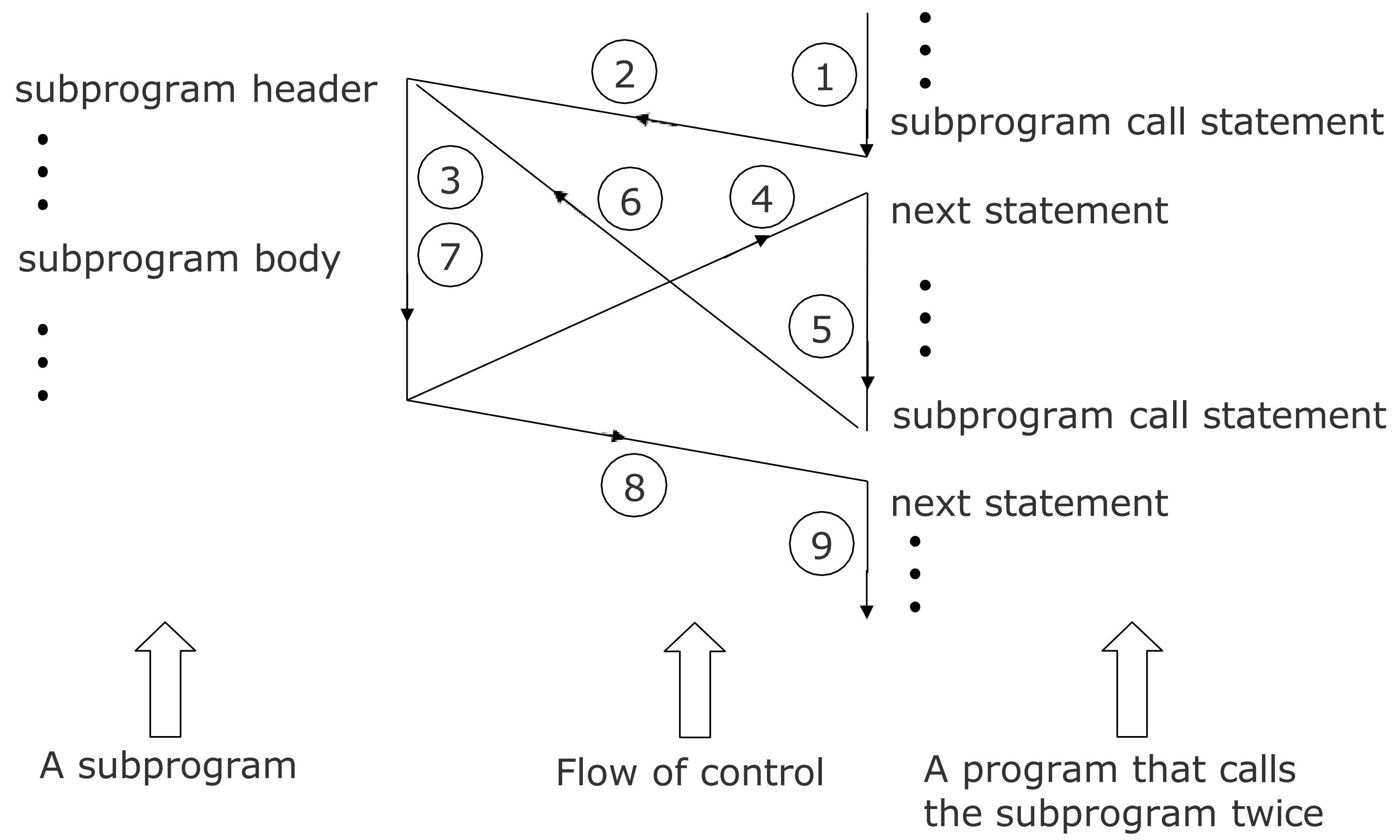
Subprogram

- Program written in a manner that it can be brought into use in other programs and used whenever needed without rewriting
- Also referred to as *subroutine*, *sub-procedure*, or *function*
- Subprogram call statement contains the name of the subprogram followed by a list of parameters enclosed within a pair of parentheses
- *Intrinsic subprograms* (also called *built-in-functions*) are those provided with the programming language
- *Programmer-written subprograms* are written and used as and when they are needed

Structure of a Subprogram



Flow of Control in Case of Subprogram Calls



Key Words/Phrases

- Assembler
- Assembly language
- BASIC
- Built-in function
- C
- C++
- C#
- COBOL
- Coding
- Compiler
- Computer language
- FORTRAN
- Function
- High-level language
- HotJava Interpreter
- Intrinsic subprogram
- Intermediate compiler and Interpreter
- Java
- Just-in-time compilation
- Language processor
- Linker
- LISP
- Load module
- Logic error
- Low-level language
- Machine language
- Macro instructions
- Object program
- Object-oriented programming
- Opcode
- Operand
- Pascal
- Programmer
- Programming
- Programming language
- Pseudo instruction
- RPG
- Self-documenting language

(Continued on next slide)

Key Words/Phrases

(Continued from previous slide..)

- SNOBOL
- Source program
- Sub-procedure
- Subprogram
- Subroutine
- Symbolic language
- Syntax error
- Syntax rules
- Written subprograms