# Chapter (4)

# Developing Custom Software

## 4-1 The Software Developers

### 4-1-1 In-house Software Developers –Analysts, Programmers, and Users

Organizations with in-house computing staff often have their own employees who develop software.

In most medium-sized to large organization, systems analyst supervises software development undertaken by programmers. He serves as a kind of liaison between users (who understand business problems) and programmers (who actually develop S/W). He focuses on user needs and how the new software can be integrated effectively into Information systems.

In some companies, programmer analyst, who is computer professional, serves as both programmer and system analyst.

In small companies, users can do some program development if they are computer proficient.

## 4-1-2 Outside Developers

One main reason for using outside software consultants is that many of them are specialized in a particular area such as payroll, accounting, inventory, production, or retail sales.

See flg. table.

| In-House Developers | Outside Developers |
|---|---|
| Programmers are familiar with the company but not necessarily with the application area. | Programmers are hired based on their familiarity with a specific application area or with the type of company. |
| Because programmers are employees, the company has better control of the development process. | The price for the software is set in advance, typically on a contract or fixed-fee basis. |
| After programs are implemented, programmers are usually still available to answer queries or make modifications if the need arises. | Once a program is accepted, it may be more difficult to track down the developer if modifications are necessary. |

# 4-2 Software Development Cycle

We refer to the design steps as software development cycle because after programs have been written and used for a period of time, they may become outdated, or users may find their needs have changed. When these situations occur, the development cycle is repeated and new software is produced to meet changing needs.

Seven steps are included in the S/W development cycle:

1. Develop the program specifications with help of the users.
2. Design the logic to be used in the program.
3. Code the program and translate it into machine language.
4. Test the program until it is fully debugged.
5. Install, or implement, the program.
6. Maintain the program.
7. Document the program.

## 4-2-1 Develop the Program Specifications with the Help of the Users

An important first step is to have software developers and users agree on program specifications that indicate what the software is to accomplish.

## 4-2-2 Design the Program Logic

Once the problem is clearly defined by the user and systems analyst and is described as a set of program specifications, the programmer can begin to design a program. First, the developers create an algorithm. For example, here is an algorithm for preparing a customer invoice:

- Read the account number, the unit price of the item purchased, and the quantity purchased.

- Calculate the bill by multiplying the unit price by the quantity purchased.
- Print the account number and the total bill.
- Repeat the process for all customer invoices.

In addition to planning the logical steps, the programmer should include error-control procedures (for example: 2 < unit price < 300).

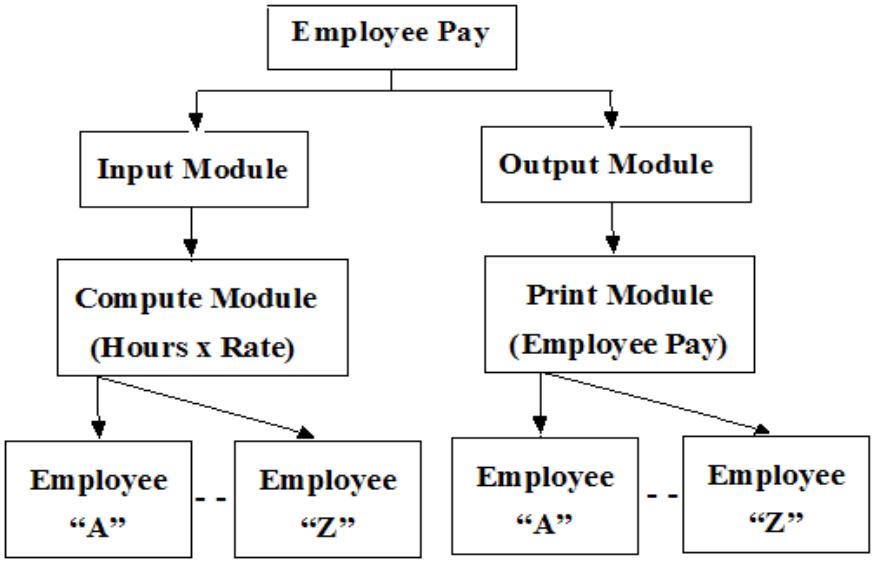
After creating the algorithm, the programmers will preferably use structured programming techniques. These techniques consist of the following:

- Top-down program design.
- Pseudocode.
- Flowcharts. &
- Logic structures.

❖ ***Top-Down Program Design***

Top-down program design is used to identify the program's processing steps. Such steps are called program modules (or just modules). Each module is made up of logically related program statements.

The program must pass in sequence from one module to the next until all modules have been processed by the computer. See flg. fig: Calculation of Gross Pay of employees.




- ***Pseudocode***

  Pseudocode is an outline of the logic of the program you will write. It is like doing a summary of the program before it is written. I.e. it need not indicate all the processing details, but it should carefully describe the overall *flow of program logic*. See flg. fig: Calculation of Gross Pay of employees.

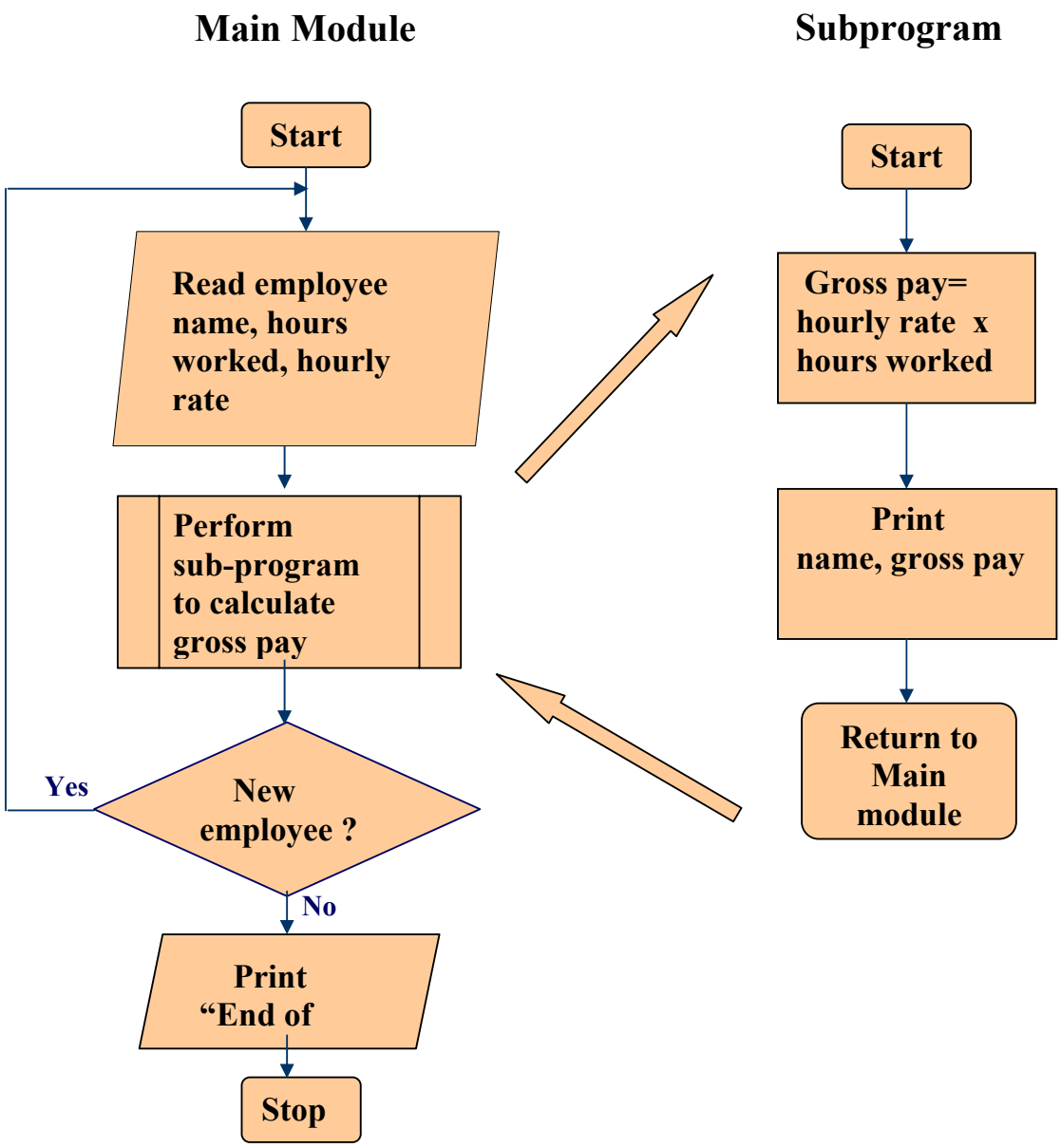
```
START
        READ Name, Hours, Rate
        PERFORM subprogram to calculate Pay
        PRINT Name, Pay
 STOP
```

- ***Flowchart***

  A flowchart presents graphically the detailed sequence of steps needed to solve a programming problem.

  It illustrates the major elements of the program and how they will logically integrate. The flowchart should be discussed with users before any actual

development of the programs. See flg. fig: Previous program Calculation of Gross Pay of employees: Compare it with the pseudocode.

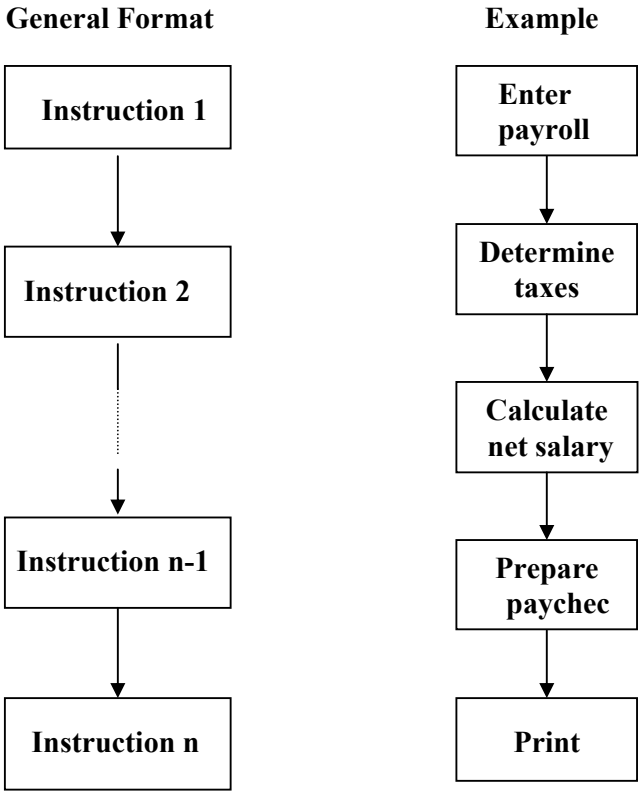



❖ ***Logic Structures***

How do you link the various parts of the flowchart? The best way is a combination of three logic structures called *sequence, selection,* and iteration (or *loop).* Using these arrangements enables you to write so-called structured program, which is a design technique that integrates, or ties together,

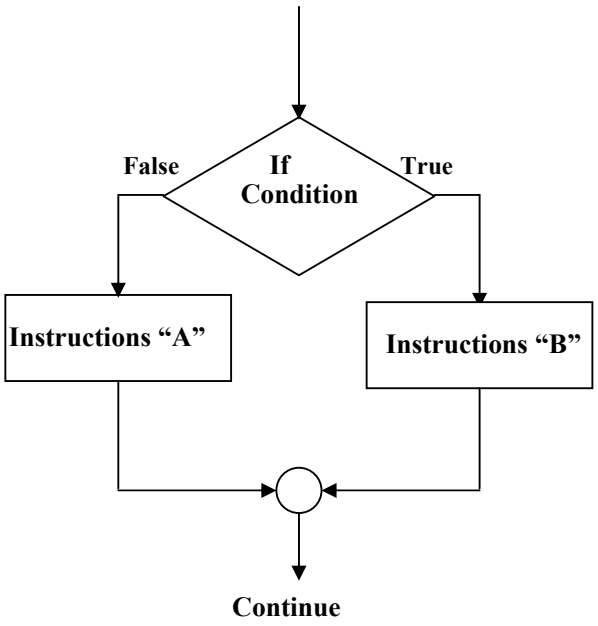
program modules in a standard way (There are no branch points (GO TOs) that make the logic difficult to follow).

- ***Sequence:*** Each instruction in a program is executed in sequence. See flg. fig: Calculate net salary and print the check.



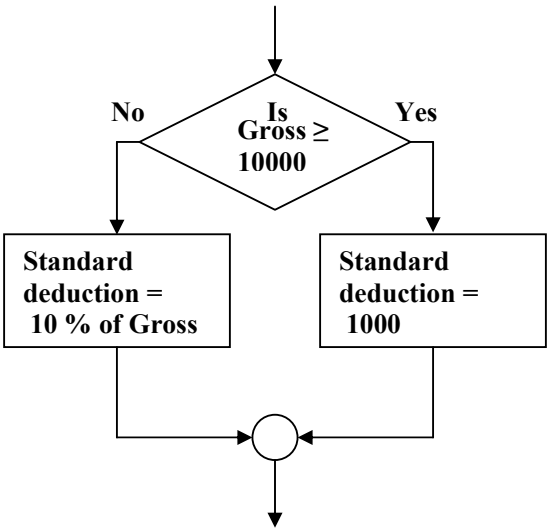

- ***Selection:*** Instructions can be executed ***selectively.*** If a given condition is *true* or *is met*, THEN specific instructions or modules will be executed. If the condition is *false* or *is not met*, THEN a different set of instructions is executed. See flg. fig: Calculate net salary; tacking into account that Tax=10% if Gross < 10000 $ and Tax= 1000 $ if Gross ≥ 10000 $.

**General Format**



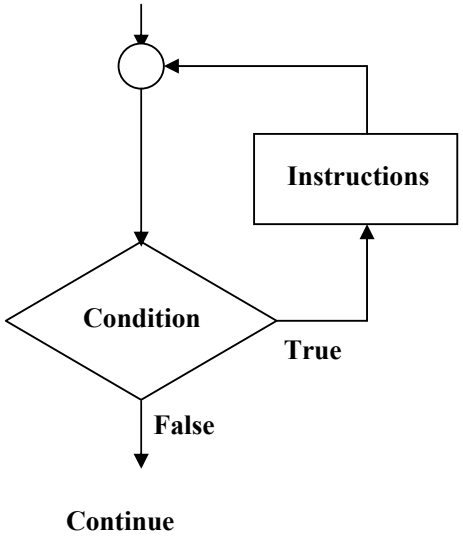

**Example**

(Calculate net salary module: Gross – Tax)



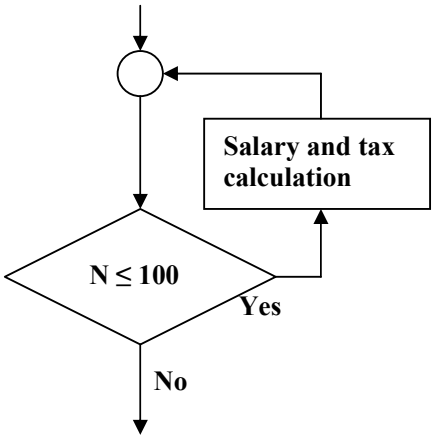

- ***Iteration or looping:*** A module can be run from another module either once or repeatedly. See flg. fig: Calculate salary and tax for 100 employees.

**General Format**




**Example**

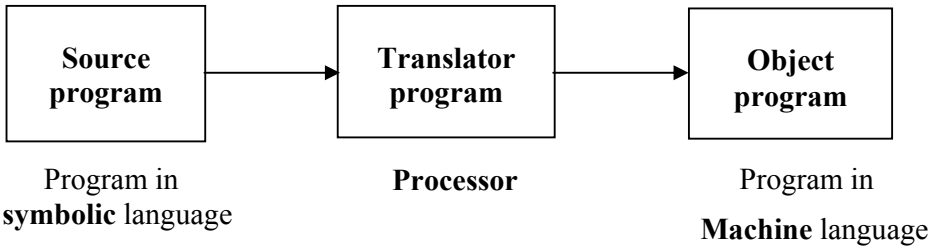
(Calculate salary and tax for 100 employees)



## 4-2-3 Code and Translate the Program

Once the design and the logic are agreed by the programmer, user, and systems analyst, the programmer writes, or codes, the instructions in one of a wide variety of programming languages. Most programs are written **in symbolic languages**, such as *Visual Basic or C++*.

A program written in a *symbolic language* (called **source program**) is not, however, executable in that form. It must translate *into **machine** language* (called object **program**) by translator *program*: See flg. fig.




## 4-2-4 Test the Program Until It Is Fully Debugged

Debugging means finding and correcting all coding **(syntax)** and logic errors, in a program: *typing mistakes such as INPUTT* is *a* **syntax error.**

Note that programs cannot be fully translated into machine language until all syntax errors have been corrected. This is the first step in the debugging process.

The program may be coded perfectly but still generate incorrect output due to a **logic error**. It may occur because there is a mistake in the sequencing of the instructions or because the wrong instructions were used. Mistakes in logic results in run-time errors, which means that the output is incorrect or that the program run was terminated. So, *the main debugging task is to ensure that the program runs and the output is correct*.

Note that some newer languages have **interactive debuggers** that help programmers find errors at certain breakpoints and displaying various program components.

## 4-2-5 Implement (Install) the Program

Before a new program can be implemented, a conversion plan must be carefully developed and monitored to ensure a smooth transition from the old set of procedures to the new one. The staff must also be trained to assist in the conversion process.

## 4-2-6 Maintain the Program

Studies have shown that only about one-quarter of a programmer's time is spent in developing new software. The remaining three-quarters are spent maintaining existing software. Software maintenance falls into two broad categories:

- Correcting errors and making the software easer to use.
- Making modifications to accommodate changing needs.

Software maintenance is the main responsibility of maintenance Programmers.

## 4-2-7 Complete the Documentation for the Program

Documentation, which explains every facet of a program to the user, should be prepared on an ongoing basis as the software is being developed, tested, and implemented. Some documentation is built into the program itself as comments, but most documentation takes the form of printed user manuals. Some parts of the manual explain how to use the program; other parts are written for the technical staff and explain the actual methodology of the program in case it needs modifications.

Before a S/W is considered complete, the documentation must be finalized and distributed to users.

## 4-3 Object-Oriented Programming (OOP) concept

Object oriented programming (OOP) is a process by which a program is organized into objects.

Each object contains both the data and processing operations necessary to perform a task.

Object-oriented programs use objects that are reusable, self-contained components. Programs built with these objects assume that certain functions are the same. For example, tabulating procedure may be suitable for many different programs. There is no need to invent this activity anew every time.

### *Visual programming*

Visual programming is an object-oriented concept that makes use of Windows-type menus, buttons, and other graphics as objects to create programs that are visually appealing, reusable, and relatively easy to code. Visual Basic and Visual C are two examples of visual programming languages.

Visual programming makes application development easier and faster even for non- programmers. 33 % - 80 % the amount of code, to create an application, is reduced when compared with more conventional techniques.

## 4-4 Five Generations of Programming Languages

They are:

- Machine Language: The First Generation.
- Assembly Language: The Second =
- High-Level Languages: The Third =
- Very high-level languages: Fourth-Generation Languages (4GLs).
- Natural languages: Fifth-Generation Languages (5GLs).

Programs are now rarely written in first or second generation. Most are written in a third or Forth-generation language. Fifth-generation languages are more significant in their potential than in their use.

## 4-4-1 Machine Language: The First Generation

Machine language is the computer's internal language, which executes directly without translation.

Initially, in 1940s & early 1950s, all programs had to be coded in machines languages.

Programming in machine language, however, is time consuming and is conducive to making mistakes. Also, machine language is different for each type of computer ***(computer dependent)***, *and* ***nonstandard.***

## 4-4-2 Assembly Language: The Second Generation

In the 1950s, computers were first used commercially, and second generation **(assembly languages)** is started.

In assembly language, the complex operation codes required for execution can be assigned names such as ADD, SUB, and MULT that are easy to remember. Also, the actual storage addresses where data is located can be defined with names such as AMT1 & AMT2 for ease of reference.

Program written in assembly language needs special program called **assembl**er to translate it into machine language.

Assembly language for a given computer depends on its architecture. So, it is different from computer to another. Therefore, assembly languages are ***computer dependent***, and the programs are ***not portable***.

Despite the complexity of assembly languages, they are still used for some applications today. Because they are so similar to machine language, they

result in very efficient code – that is, the program uses relatively little storage.

## 4-4-3 High-level Languages: The Third Generation (3GLs)

**3GLs** are called high level because they are relatively easy to learn (more like the English language).

These languages are ***portable***. But they need very complex translation process. In general, the easier it is to program, the higher the level and the more complex the translation process.

Most translators programs for high-level languages are called ***compilers***, but some are called ***interpreters.***

High – level languages (like their predecessors) are called ***procedural* languages:** *precise set of instructions is needed to accomplish a given task.*

Some of 3GLs that are in use today:

- ***FORTRAN - The first high-level language***

  **FORTRAN** (**For**mula **Tra**nslator): It was developed by IBM & introduced in 1954. It was developed for scientific and engineering applications. It is not well suited for typical business applications that require relatively simple arithmetic operations but a large volume of input/output operations.

  In the past few years, C or C++ has replaced FORTRAN, and very little, if any, new application development is being done in FORTRAN.

- ***COBOL – The language of business***

  **COBOL** (**Co**mmon **B**usiness Oriented **L**anguage) was introduced in 1959. COBOL is very good for processing large, complex data files and for producing well-formatted business reports. COBOL is ***standard*** language, and all compilers include the same basic instruction set, and it is ***portable.***

It remains the most widely used language for business programs developed for mainframes: it has been estimated that 70 to 80 % of all mainframe applications are still coded in COBOL and many micro-based applications are written in COBOL as well.

- ***BASIC***

**BASIC** (**B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode) was introduced in 1965. It is easy to learn and is appropriate for small businesses with a limited programming staff.

The translator program that converts BASIC code into machine language takes very little primary memory.

Although BASIC is used mainly on micros, it is available on mainframes as well.

- ***C: The High-level Alternative to Assembly Language Programming***

It was developed in 1972 (at Bell Lab.). C incorporates the advantages of both assembly language and high-level languages and therefore often referred to as a middle-***level*** *language*. It is structured language that uses high-level instruction formats, but it also allows the programmer to interact directly with the hardware.

C was originally designed to write systems S/W (that require extremely efficient code) but is now considered as general-purpose language.

In addition, instructions written in C are ***portable***. The UNIX operating system was written in C.

Because of its power and wide implementation on microcomputers, it is popular among systems programmers for developing systems S/W and utilities.

Today C is fast being replaced by the enhanced C++ language.

- ***Object-Oriented Programming (OOP) Languages: Visual BASIC, C++, and JAVA***

  In object-oriented programs, objects are used instead of the operation codes and data names or symbolic addresses that are common to most languages.

- ***Visual BASIC*** (often referred as VB) was introduced by Microsoft in 1987 as its first visual development tool. It allow the programmer to easily create complex user interfaces containing standard Windows features such as bottoms, dialog boxes, scroll bars, and menus.

- ***C++*** is OOP that currently dominates the market, but that domination is being challenged by Java. C++ is an enhancement of C language. It includes everything in C and adds support for object-oriented programming. In addition C++ contains many improvements and new features.

  Versions of C++ are available for almost all large system and personal computer platforms.

  *Note*: Visual C is a C++ Microsoft version. It is also designed for Windows application development.

- ***JAVA***

  **JAVA**, developed by Sun Microsystems, is a *"pure"* OO language.

  JAVA can run on many platforms (platform is a combination of H/W & OS ); for that it is relevant to Internet development. It has good start on becoming the universal language of Internet computing.

***Compilers and Interpreters***

Third-generation languages use translators, which are more sophisticated than assembler.

- ***Compilers:*** A compilers translates the entire source program into machine language in one process, thereby creating object program. Most high-level languages use this type of translators**.**
- ***Interpreters:*** *An interpreter* translates a program written in high-level language (like BASIC) one statement at a time as the program actually being run on the computer.

Interpreters do not take up as much space in the computer's primary storage, but they are slower than compilers.

## 4-4-4 Very high-level Languages: Fourth-Generation Languages (4GLs)

Languages belonging to the first 3 generations are *procedural languages.* They consist of instructions that describe the step-by-step procedure to solve the problem.

4GLs are *nonprocedural languages.* In 4GLs program the programmer specifies the desired results, and the language develops the solution.

4GLs are designed to solve specific problems. They are easier to program than 3GLs, and they are widely used by nonprogrammers.

Most experts say that the average productivity improvement factor is about 10; that is, you can be 10 times more productive using 4GL than 3GLs.

Many 4GLs are part of a database management system. 4GLs include query languages and application generators.

- ***Query Languages***

They enable nonprogrammers to use certain easily understood commands to search and generate reports from databases. They are easy to learn. Example of query languages is Structured Query Language (SQL).

- ***Application Generators***

  An application generator is a S/W product that enables you to quickly code new application. An example wizard program, wizards allow you to create such things as fax cover sheets, database reports, and Web pages.

  Many applications generators contain both procedural and nonprocedural components. Examples of these Generators are:

  - *Code Generators:* Code Generators are programs that allow applications to be created by automatically translating them from one language into another. For ex. Code Generators, which are used to translate documents written in Word processing program into HTML-coded Web pages.
  - A wizard program that enables users to create Web pages by responding to prompts.

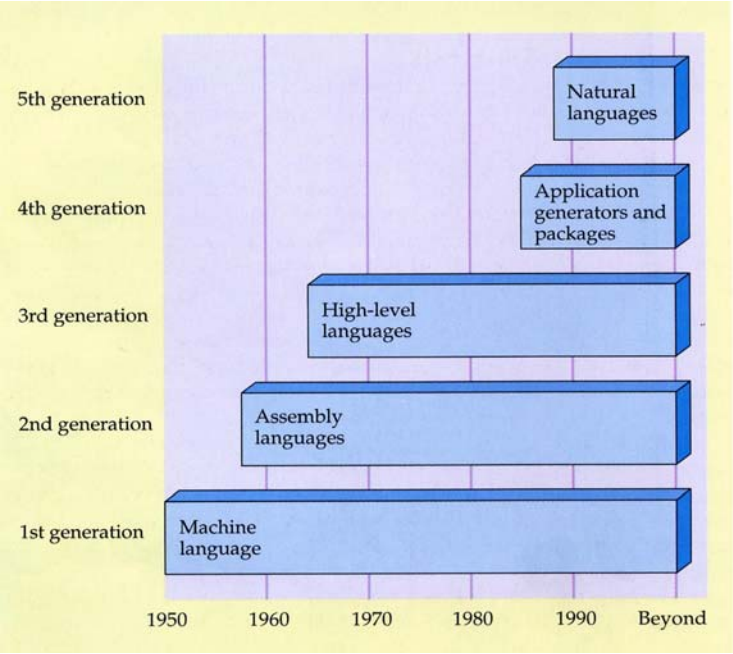
## 4-4-5 Natural Languages: Fifth-Generation Languages

5GLs are most often called *Natural Languages* because of their resemblance to the "natural" spoken or written languages. The natural language translates human instructions into code the computer can execute.

5GLs are also nonprocedural languages. They are most often used to access database or to build expert systems.

Currently, most expert or knowledge-based systems are coded either in LISP or PROLOG although some are written in C or C++ as well. 5GLs are likely to have great impact in the near future.

Following. fig. is a view of the 5 generations of languages.




***CHAPTER SELF-TEST***

1. ***Application programs are _____and_____.***
2. ***Who develops custom programs?***
3. ***A program written in a symbolic language must be _____before it can be run.***
4. ***(T or F) Programs must be coded before the program specifications are determined***
5. ***(T or F) A program written in a high-level language is user-friendlier than one written in an assembly language.***
6. ***The language_____ has the advantages of both assembly languages and high-level languages and is often used for developing operating systems***
7. ***(T or F) Most third-generation languages are portable, which means they are not machine dependent.***
8. ***A language commonly used for business application is called_____.***
9. ***What was the first widely used high-level language for scientific and mathematical applications?***
10. ***(T or F) A compiler translates and executes programs written in a symbolic language one statement at a time.***
11. ***(T or F) Fourth-generation languages such as SQL are often used to query databases.***

***Solutions:***

1. ***Packaged programs and custom designed programs.***
2. ***In-house software develops or outside develops.***
3. ***Translated.***
4. ***F.***
5. ***T.***
6. ***C.***
7. ***T.***
8. ***COBOL.***
9. ***FORTRAN.***
10. ***F.***
11. T.