CHAPTER II
GENERAL DESCRIPTION OF THE OBJECT OF THE STUDY

II.1. How does a computer work?

II.1.1. The components of a computer.

Computers come in all shapes and sizes, ranging from the multimillion-rupiah supercomputers to the portable computers that fit into a briefcase. But despite this great diversity, all computers function in a similar fashion.

- The ***Central Processing Unit***. The central processing unit (CPU) is the 'brain' of the computer, where all the data processing takes place. The ***arithmetic or logical unit*** is the part of the CPU that performs arithmetic and makes logical decisions. The ***control unit*** interpretes instructions and sequences their execution.
- **Primary Memory**. Primary memory (also called ***main memory, random access memory***, or RAM) allows for very rapid access to its contents. Primary memory holds the program that is currently being executed as well as the

data that the program requires. It typically takes about one millionth of a second to retrieve a single piece of data from primary memory. However, primary memory is erased when the power is turned off and thus is not a suitable medium for long-term storage needs.

- **Secondary Memory**. Secondary memory is a collection of devices that provide long-term storage of data and programs. There are many secondary memory devices available, including floppy disk drives, hard disks, and magnetic tape drives. Secondary memory is much slower than primary memory for data storage and retrieval, but it costs much less for a given amount of storage.
- **Input Devices**. The input devices allow for entering data and programs into a computer. There are many available input devices, including keyboards, floppy disk drives, and hard disk drives.
- **Output Devices**. The output devices provide output from the computer to its users. Such devices include display screens, printers, and plotters.

II.1.2. The operating system

A computer system is a sophisticated collection of hardware. To operate the hardware and organize the flow of data from one system component to another. the system uses a set of programs called the *operating system.* The operating system accepts commands from the system operator, interpretes them, and carries them out. It also controls the storage to and retrieval of data from the secondary storage devices. In addition, the operating system provides programs to manipulate the various system hardware, so that programmers can use the hardware without knowing all the details about how it operates.

A *file* is a collection of related data stored in a computer system. A file can contain accounting information, a company report, a stored graphics image, or a program. The operating system provides commands for performing a variety of file manipulations. such as copying the contents of one file to another, erasing a file, and renaming a file.

II.1.3. How a program is executed ?

A program is a sequence of instructions for the computer to carry out. The program may contain data that are to be manipulated or it may use data that are contained in one or more data files in secondary storage.

The program is stored as a file in secondary storage. When we wish to run the program, we give command to the computer. On a microcomputer, this is usually done by typing the command from the keyboard. In response to the command, the computer first reads the program from secondary storage and places it into primary storage. The CPU contains an *instruction register*, which identifies the next instruction to be executed in terms of the instruction's location in primary memory. When the program is first placed into primary storage, the instruction register is set to indicate the first instruction of the program.

Here are the steps:

1. The CPU reads the instructions specified in the instruction register.
2. The instruction is decoded in a special decoding portion of the CPU.
3. The instruction is carried out.
4. The instruction register is updated by copying into it the location of the next instruction.
5. If there are further instructions in the program, the computer will repeat the procedure, beginning with Step 1.

II.2. PASCAL LANGUAGE

Pascal is a programming language designed by the Swiss computer scientist Niklaus Wirth in the late 1960s. Pascal emphasizes *structured programming*, a style of programming in which a program is written as a sequence of steps to be executed one after the other. The steps may contain substeps, each of which is executed one after the other (Joel and Goldstein,1987:25)

Pascal is a block-structured language and a Pascal program is composed of an arbitrary number of possibly nested blocks, subprograms, called procedures and functions. There is a single main program that may contain within it one or more definitions for procedures or functions. Each of these may, in turn, contain nested blocks. Programs, procedures, and functions are all coded in the same manner (with the exception that a function's name must have a declared type). Each block contains (optionally) its own local definitions and declarations and a sequence of executable statements delimited by the keyword *begin* and *end*. The block header contains:

- a keyword indicating the kind of block (*program*, *procedure*, *function*);
- the name of the block;
- and any parameters to be associated with arguments at the point of call (Friedman,1991: 306)

QUICK REFERENCE TO PASCAL (DALE AND WEEMS,1991)

Syntax Element	Example
Heading	PROGRAM Reference (Input, Output, DataFile);
Declarations	
CONSTants	CONTS
real	TaxRate = 0.27;
integer	StrSize = 20;
string	Blanks = '
TYPEs	TYPE
set	Letters = SET OF 'A'..'Z':
enumerated	Jobclass = (PtTime, Unit, NonUnit, Super);
subrange	Range = 1..StrSize;
string	String = PACKED ARRAY [Range] OF Char;
array	Table = ARRAY [JobClass] OF Integer;
record	Person = RECORD Name: String; Pay: Real END; (* Record *)
file	People = FILE OF Person;
pointer	Ptr = ^Person;
VARiables	VAR
integer	Number: (*An integer variable*) Integer;
real	Overtime, (*A real variable *) Regular: (* Another real *) Real;
character	InChar: (* One character *) Char;
Boolean	Flag: (* Boolean flag *)

	Boolean;
text file	DataFile: (* External Data File *) Text;
array	ClassCount: (* Holds four integers *) Table;
record	Individual: (* A single record *) Person;
PROCEDURE	PROCEDURE GetTable(VAR InFile: (* Receives *) Text; VAR InTable: (* Returns *) Table;
local variables	VAR Counter: JobClass;
body	BEGIN (* GetTable *) FOR Counter := PtTime TO Super DO Readln(InTable[Counter]) END; (* GetTable *)
FUNCTION	FUNCTION NetPay (GrossPay: (* Receives *) Real): Real; (* Returns *)
body	BEGIN (* NetPay *) NetPay := GrossPay - GrossPay * TaxRate END; (* NetPay *)
Program Body	BEGIN (* Reference *)
Comment	(* These code segments are examples of proper syntax *) (* and style only. This isn't a working program. *)
Assignment	
integer +,* MOD,-,DIV	Number := (Number + 1) * 100 Number := Number MOD 10 - Number DIV 100;
real *,-,/,+ boolean,=,>,<	Overtime := Overtime * 15.0 - Overtime / 20.0 + 0.5; Flag := (Number = 1) OR (Number > 4) OR (Number <0);
NOT,AND,OR	Flag := NOT EOF AND NOT EOLN OR Flag;
Input/Output	
Read DataFile	Read(DataFile, Overtime, Regular);
Readln Input	Readln(InChar, Number);
Write DataFile	Write(DataFile, 'Overtime = ', Overtime:8:2);
Writeln Output	Writeln(Individual.Name:30, ' ':5, Number:1);
IF-THEN-ELSE	
condition	IF (Number <> 0) AND (0.001 >= Abs(Overtime))
true branch	THEN BEGIN Writeln('Regular work week.'); Individual.Pay := 400.0

```
                    END
    false branch    ELSE
                      Writeln('Special work week.');

IF-THEN
    condition       IF Flag
    true branch       THEN
                        Number := 0;

WHILE Loop          Number := 1;
    test            WHILE (Number <= 20) AND NOT EOLN DO
                      BEGIN
                        Read(Individual.Name[Number]);
                        Number := Number + 1
                      END;

REPEAT Loop         REPEAT
                      Writeln('Enter Y or N.');
                      Readln(InChar)
    test            UNTIL InChar IN ['Y', 'N', 'y', 'n'];

FOR Loop            FOR Number := 1 TO 10 DO
                      Writeln(Number:10, Sqr(Number):10);

CASE
    selector        CASE InChar OF
    table list        'P', 'N': BEGIN
                                  Table[PtTime] := Table[PtTime] + 1;
                                  Table[NonUnit] := Table[NonUnit] + 1;
                                END;
                      'U'     : Table[Unit] := Table[Unit] + 1;
                      'S'     : Table[Super] := Table[Super] + 1
                    END;  (* Case *)

WITH                WITH Individual DO
                      Writeln(Name, ' ':5, Pay:8:2)
End of Program  END.        (* Reference *)
```

II.2.1. Naming program elements: identifiers

Identifiers are used in Pascal to name things. Some are defined in the language and are reserved for specific uses. Others are defined by the programmer. Identifiers are made up of *letters* (A-Z, a-z) and *digits* (0-9), but must begin with a letter. Here are some examples of valid identifiers:

Hello G7 Room17w LinguaPoetica Count Abc3d4

And here are some examples of invalid identifiers and the reasons why:

Identifier	*Explanation*
40Hours	Identifiers must begin with a letter.
Get Data	Blanks are not allowed in identifiers.
Box-22	The hyphen (-) is a math symbol (minus) in Pascal.
SizeIn$	Special symbols are not allowed.
Program	The word program may only appear in the program heading.

II.2.2. Data and data types

A computer program operates on data (stored internally in memory, stored externally on disk or tape, or input from keyboard, text scanner, or electrical sensor) and produces output. In Pascal each piece of data must be a specific type. The data type determines how the data is represented in the computer and how the kinds of processing can be performed by the computer. Some types

of data are used so frequently that Pascal has define them. But some must be defined by the programmer. Pascal defines four simple types of data, they are: integer numbers, real numbers, characters, and Boolean.

II.2.2.1. Integer

Integers are positive or negative numbers; they have no fractional part. They are made of an optional sign and one or more digits:

+22 -17 1 -2890 0 1997

When there is no sign, we must assume that the number is positive. Commas are not allowed. Theoretically, integers can have any numbers of digits; practically, the computer limits their size. Pascal has a predefined identifier, **MaxInt**, whose value is set to be the largest integer number that can be represented in a computer. If MaxInt is 32767, then the range of integers allowed is -MaxInt through MaxInt or -32767 through 32767. MaxInt is different for every computer.

II.2.2.2. Real

Real numbers are decimal numbers, having an integer part and a fractional part, with a decimal point in between. They can also have an exponent, as in scientific notation. (In scientific notation, a number is written as

a decimal value multiplied by a power of 10 to indicate the actual position of the decimal point). Instead of writing 7.901×10^{12} , however we can write 7.901E+12. Here are some examples of real numbers: 18.0 123.45 7911E2 1234567.98765 3456E-13 3.14E2

II.2.2.3. Char

Data type Char describes data consisting of one alphanumeric character-a letter, a digit, or a special symbol: 'A' 'a' '8' '2' '+' '-' '$' '?' '*' ' '

Each machine has a character set, the set of alphanumeric characters it can represent. Each character is enclosed in single quotation mark.

II.2.2.4. Boolean

Boolean is a type of data with just two values: True and False. We use it to test conditions in a program, to help the computer make decisions.

To ask a question in Pascal, we make an assertion. If the assertion we make is true, the answer to the question is yes. If the statement is not true, the answer to the question is no. For example, if we want to ask ,''Are we going to the movie tonight?'' we would say,''We are going to the movie tonight.'' If the assertion is true, the answer to the question is yes. If not, the

answer is no. So asking questions in Pascal means making an assertion that is either true or false. The computer evaluates the assertion, checking it against some internal condition (the values stored in certain variables, for instance) and seeing whether it is true or false.

In Pascal, assertions take form of Boolean expressions. Just as arithmetic expression is made up of numeric values and operations, a Boolean expression is made up of Boolean values and operations. A Boolean expression can be:

- a Boolean variable or constant,
- an expression followed by a relational operator followed by an expression,
- a Boolean expression followed by a Boolean operator followed by a Boolean expression.

Data type Boolean has just two literal constant:True and False. A Boolean variable is a variable declared to be of type Boolean, which means that its contents can be either True or False. For example, if **DataOK** is a Boolean variable, then **DataOK:= True** is a valid assignment.

We can also assign values to Boolean variables by setting them equal to the result of comparing two expressions with a relational operator. Relational

operators tests a relationship between two values. This is an example, Test is a Boolean variable and A and B are Integer variables:

```
BEGIN
    Read(A,B);
    Test:= A<B;(*Compares A and B with the "less than" relational *)
              (*operator and assigns the Boolean result to Test *)
```

If the value of A is 5 and the value of B is 8 then the assertion is True; if they are in another way around it is False. These are relationships that we can test for in Pascal:

- = Equal to
- <> Not equal to
- > Greater than
- < Less than
- >= Greater than or equal to
- <= Less than or equal to

II.2.3. Declarations

Identifiers can be used to name both constants and variables. In other words, an identifier can be the name of a memory location whose contents never change or the name of a memory location whose contents do change.

We can tell the computer what an identifier represents with a *declaration*, a statement that associates a name (an identifier) with a description of an element

in a Pascal program (just as a dictionary definition associates a name with a description of the thing being named). In a declaration we name an identifier and what it represents.

As a general rule, all identifiers in Pascal must be declared or defined before they are used. This is why all declarations are grouped together at the beginning of a program, in the declaration section. Within the declaration section, the declarations appear in a standard order. The first identifiers declared are the constants.

II.2.3.1. Constants

All numbers, integer and real, are constants. So are single characters and series of characters or strings: 16 32.3 'A' 'English Department' , notice that character literals and string literals are in single quotations mark. This is to differentiate between strings and identifiers. 'Arabic' (in single quotation marks) is the character string made up of the letters A,r,a,b,i,and c in that order. Arabic (without the quotation marks) is an identifier, the name of a place in memory. This is a valid constant section:

```
CONST
    Stars=          '********';
    Blank=          ' ';
    Pi=               3.14159;
    InterestRate=     0.15;
    TaxRate=          0.001;
    Max=              35;
    Message=        'Syntax error';
```

II.2.3.2. Variables

A program operates on data. Data is stored in memory. While a program is executing, a different value may be stored in the same memory location at different times. This kind of memory is called a ***variable***, and its contents are the ***variable value***. The symbolic name that we assign to a variable memory location is the ***variable name*** or ***variable identifier***. In practice we simply refer to the variable name as the ***variable***.

Declaring a variable means specifying both its name and its data type. This tells the compiler to associate a name with a memory location whose contents will be of a specific type (for example, Integer, Real, Char, and Boolean). Pascal is strongly typed language. This means that a variable can only contain data values of the type specified in its declaration (Goldstein 1987:26).

II.2.4. Taking action: executable statements

We have already looked at ways of defining objects in a program. Now we will look to ways of acting, or performing operations, on those objects.

II.2.4.1. Assignment

The value of a variable is changed through an ***assignment statement***. For example, A:= 10; means assigns the value 10 to the variable A (puts the value 10 into the memory location called A). This is the form of an assignment statement ***variable***:= ***expression***; the meaning of the assignment operator (:=) is `becomes'; the value of the variable **becomes** the value of the expression.

Only one variable can be on the left-hand side of an assignment statement. An assignment statement is not like a math equation (x + y = z + 4); the expression (what is on the right-hand side of the assignment operator) is evaluated, and the value is stored in the single variable on the left of the assignment operator. The value assigned to a variable must be the same type as the variable. Given the declarations

```
VAR
  Num,
  ID:
     Integer;
  Rate:
     Real;
  Test:
     Boolean;
  Ch:
     Char;
```

The following are valid assignments:

Variable	Expression
ID	:= 2856
Rate	:= 0.36
Test	:= True
Ch	:= 'B'
Num	:= ID

These are not valid assignments:

Assignment Statement		Explanation
ID	:= 2.5	ID is Integer; 2.5 is Real.
Ch	:= 3	Ch is Char; 3 is an Integer constant.
Test	:= 'A'	Test is Boolean; 'A' is a Char constant.
Ch	:= 'Hello'	Ch can hold only one character.
Test	:= 'True'	Test is Boolean; 'True' is a string.
Num	:= Rate	Num is integer; Rate is Real.

Variables keep their assigned values until they are changed by another assignment statement. Expressions are made up of constants, variables, and operators. These are all valid expressions: ID + 2 Rate - 6.0 4 - ID

The operators allowed in an expression depend on the data type of the constants and variables in the expression. The *arithmetic operators* are:

- + Addition
- - Subtraction
- * Multiplication
- / Division
- DIV Integer Division (no fractional part)
- MOD Modulus (remainder from integer division)

II.2.4.2. Output

Maybe we often asked someone, ``Do you know what time it is?'' and the person only smile and say,''Yes,I do.'' This is like the situation that currently exists between us and the computer. We already know enough how to tell the computer to perform simple calculations, but the computer won't give us the answer until we tell it to write them out.

In Pascal we use **Writeln** statement to write out the results of calculations. **Writeln** is short for *write line*, which is how it's pronounced. The writeln statement takes the following form: **Writeln** (*parameter,parameter* ...); . A parameter can be a literal, a constant or variable identifier, or an expression. Here is an example of a

Writeln statement: **Writeln('The answer is', Result);** The writeln statement is a standard procedure.

II.2.4.3 Procedure

A ***procedure*** is a subprogram. A standard procedure is a subprogram built into the programming language. We call (use) a procedure by writing its name, often followed by a list of parameters. The ***parameter list*** is a way for a main program to communicate with a subprogram. In the case of Writeln, the parameter list is simply a list of what we want to print. It can contain one or more ***expression*** separated by commas. Writeln takes the data we give, prints it out on one line, then goes to the start of the next line. In the last example, there are two parameters. The first is a literal string,**'The answer is'**. The second is a variable, **Result**, whose value will be printed. Here is an example of a parameter list with four parameters: **('Rate is ', Rate, ' and tax is ', Rate * Total);**

When we use a Writeln statement, the computer temporarily holds the program, starts the Writeln subprogram running, and gives it the data from the parameter list. When Writeln has finished printing the

data, the computer goes back to the program and runs where it left off.

The parameter list makes it possible for the same subprogram to work on many different sets of data. For example we can have Writeln print out something different each time we use it simply by change its parameters.

Writeln prints the current value of named constants and variables. It prints string constants exactly as they appear in the parameter list. We must use single quotation marks to enclose the string if we want to tell the computer that we want to print a string constant, not a named constant or variable. If we want to print a string that includes a single quotation mark (an apostrophe), we must type two single quotation marks, with no space between them, in the string. For example to print the word *don't*, the Writeln looks like this:

```
Writeln('don''t');
```

In addition to the Writeln statement, Pascal has a second output procedure, **Write** . Write is like Writeln, except that when it finishes printing its parameter list, it does not go to the next line. So, the output from a series of Write statements appears on one line.

II.2.5. Adding comments to a program

All we need to create a working program is the correct combination of declarations and executable statements. The computer ignores comments, but they are important to anyone who must read the program. Comments can appear anywhere in a program. They are delimited by the (* *). A comment can be more than one line. When this happens, make sure the comments doesn't include any program statements. For example, the declaration:

```
VAR
 FuelLoad:      (* The amount of fuel, entered in pounds.
   Real;           Jet-A fuel weighs 6.7 pounds per gallon *)
```

looks reasonable, but it produces an error message because the comment encloses the data type, Real. Here is one way to write the declaration correctly:

```
VAR
 FuelLoad:      (* The amount of fuel, entered in pounds  *).
   Real;        (* Jet-A fuel weighs 6.7 pounds per gallon *)
```

Here is another:

```
VAR
 FuelLoad:      (* The amount of fuel, entered in pounds.
                   Jet-A fuel weighs 6.7 pounds per gallon *)
   Real;
```

Both of these declarations place the data outside the comment.

A comment should appear at the beginning of a program to explain what the program does. Each constant

definition and variable declaration should have a comment that explains how the identifier is used. In addition, comments should introduce each major step in a long program and should explain anything that is unusual or difficult to read (for example, a long formula). It is important to make the comments concise, and to arrange them in the program so that they are easy to see an it is clear what they refer to. If comments are too long or crowd the statements of the program, they make the program more difficult to read.

CHAPTER III

DATA PRESENTATION AND ANALYSIS