CHAPTER 5 BASIC PROGRAMING ALGORITHM

Figure 5.1 shows a Russian stamp illustrating Muhammad ibn Mūsā al-Khwārizmī. For those who are active in the computer world should know him. He is an Islamic scientist many contribution in the field of mathematics, astronomy, astrology and geography and many becoming the foundation of modern science. From his name the term that will be studied in this chapter is emerged. From Al-Khawarizmi becoming **algorithm** in English and translated into **algorithm** in Indonesian.

Standard competence in basic programming is based on four (4) basic competences. In this book, each basic competence will be elaborated with exercises. The summary will be in the end of each chapter.
(source: www.wikipedia.org).

Figure 5.1. Muhammad ibn Mūsā al-Khwārizmī in a Russian stamp

The basic competence in this chapter is to explain the variable, the constant and type of data, create an algorithm / programming logic flow, applying array management, and file operation. Before proceeding please review the operation system, the principle of problem solving, and supporting materials such as mathematics.

At the end the chapter, some exercises will included from easy problems into difficult problems. This exercise is used to measure capacity in the basic competence. Thus, after studying the basic competence on your own or under teacher's guidance, you may measure your own capacity by doing the exercise.

OBJECTIVES

After studying this chapter, the reader should be able to:

- Explain the variables, the constants and the data types.
- Create algorithm / flow of programming logic.
- Apply management array.
- File operation.

5.1. VARIABLES, CONSTANTS, AND DATA TYPES.

Variables, constants and data types are three (3) parameters that always will be found during programming processes. Any programming language from the simplest to the most complex requires us to understand these three (3) matters.

5.1.1. Variables

A **variable** is a place that we could fill in a value, empty it , and recall it as needed. Every variable will have a **name (identifier)** and **value**. Please see the following example:

Example 5.1. variable name and value

```
username = “joni”
Name = “Al-Khawarizmi”
Price = 2500
Total Price = 34000
```

Shown in Figure 5.1, username, Name, Price and Total Price are the variable names. While, “joni”, “Al-Khawarizmi”, 2500 and 34000 are the value of each variable, respectively. These values will be kept in the name of their respective variable as long as no change is made.

In most programming languages, a variable must be initially declared to facilitate the *compiler t*o work on it. If the variable is not declared then every time compiler met the new variable in the source code there will be a delay as the compiler create a new variable. This will slow down the compiler process. Moreover in several programming languages, compiler refuses to continue compiling process.

The variable name must follow the naming convention of the programming language. In general, there is a common convention among these programming languages. These conventions are:

- Variable name must be preceded with letter
- No space in the variable name. One may use underscore (_) to represent a space.
- Variable name should not contain special characters, such as: .,+, -, *,/, <, >, &, (,) etc.
- Variable name should not use key words of the programming language.

Example 5.2. Example of variable name.

Correct variable name	Wrong variable name
namasiswa	Nama siswa --- (space usage)
XY12	12X --- (starts with number)
harga_total	Harga.total --- (use . character as space)
JenisMotor	Jenis Motor --- (space usage)
alamatRumah	For --- (use programming language keyword)

5.1.2. Constants

A constant is a variable with fixed value and cannot be changed. Thus, a constant is also a variable but different in the value A variable should be considered as a constant if no change in its value. In a source code, the value of a constant is normally declared in constant declaration. While a variable is usually determined by its name and data type without any value in it. Naming convention of variable as well as data type convention are also used for contants.

For example, if we write a program to do mathematical calculation that uses pi value (3,14159) that might be used in many places in the program, it would be much easier to use pi as a constant. The use of pi as a constant is much easier to write rather than repeatedly use 3,14159 in the source code.

5.1.3. Data Type

Data type is the type of data that may be used by the computer to satisfy in computer programming. Each variable or constant in source code should use a definite data type. A correct data type of variable or constant will save computer resources, especially memory. One of the important task of a programmer is to select the correct data type to produce an efficient and high performance program.

There are many available data types depending on the programming language. However, in general, data types may be grouped as shown in Figure 5.2.

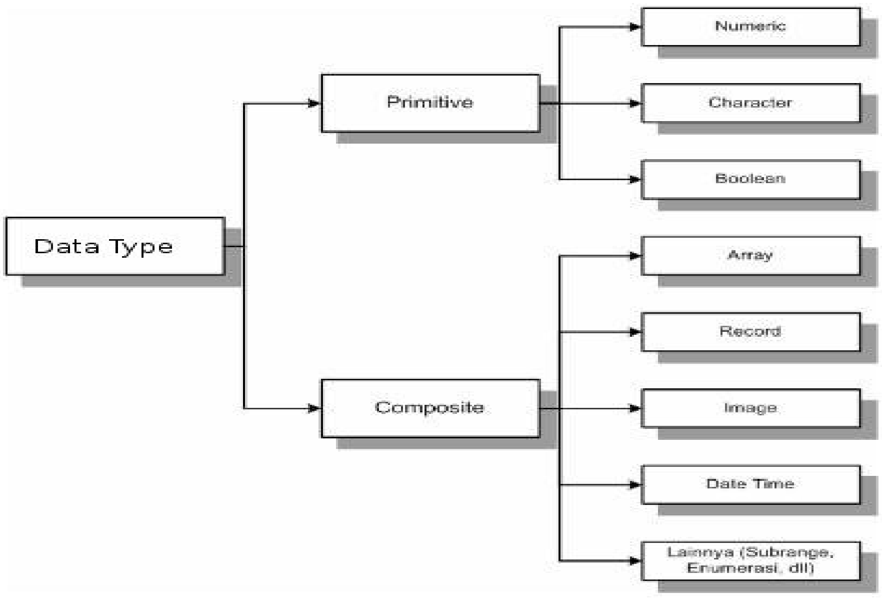


Figure 5.2. Data type grouping.

Data type *primitive i*s the basic data type that directly available in the programming language. Whereas *composite* data type is data type that formed by two or three primitive data type.

Numeric data type

numeric data type is used in variable of constant to store a numerical value. All programming language should have a numerical data type, some differences in the numerical type it may accommodate.

Numeric data type is including integer (the integer) and float (fraction). In addition to type, some programming language adds precision into the data type, such as, single for limited precision of data type and double for more accurate data type. In the upcoming chapters discussing the application of programming language this section will be explained further.

The selection of data type for variable / constant should be carefully done. Manual or guide of respective programming language in data type section must be thoroughly read. Please see the following example.

Example 5.3. Numeric data type usage.

```
Program A                                   Result A
#include <iostream>                    X =12
using namespace std;                    Y =2.15
int main() {                      Z =25
        int x, z;
        float y;
        x = 12;
        y = 2.15;
        z = x * y;
        cout << "X =" << x << endl;
        cout << "Y =" << y << endl;
        cout << "Z =" << z << endl;
        return 0; }
```

```
Program B                                   Result B
#include <iostream>                    X =12
using namespace std;                    Y =2.15
int main() {                      Z =25.8
        int x;
        float y, z;
        x = 12.8;
        y = 2.15;
        z = x * y;
        cout << "X =" << x << endl;
        cout << "Y =" << y << endl;
        cout << "Z =" << z << endl;
        return 0; }
```

```
Program C                                   Result C
#include <iostream>                    X =12
using namespace std;                    Y =2.15
int main() {                      Z =25.8
        int x;
        float y, z;
        x = 12;
        y = 2.15;
        z = x * y;
        cout << "X =" << x << endl;
        cout << "Y =" << y << endl;
        cout << "Z =" << z << endl;
        return 0; }
```

The above three (3) source codes (A, B and C) are written in C++. It it fairly similar, the only difference is in the data type. In program A, variable x and z declares as data int (integer) while y is floating point. The result is not correct. Z as X multiply by Y should be 25,8 (results of 12 x 2.15). However, since Z declares as data int, the result is 25.

Among the three (3) source code, only source code C provides the correct result. Why source code B is wrong? Please pay attention to the bold printed in the source code and pin point the error.

Character

Together with numeric data type, character data type is the frequently used data type. Character data type is often known as char or string. String data type is used to store text or anything between quotation marks (“... ”) or reaped single (‘... ’). Pay attention to the following example.

Example 5.4. Character data type usage.

Source code

```
#include <iostream>

using namespace std;

int main() {
      int x;
      x = 5;
      char huruf = 'A';
      char* kata = "Java";
      cout << "X = " << x << endl;
      cout << "Isi variabel huruf = "
      << huruf << endl;
      cout << "Isi variabel kata = " <<
      kata << endl;
      return 0; }
```

Result

```
X = 5
Isi variabel huruf
= A
Isi variabel kata =
Java
```

In this example, we declare the variable x as int (integer), whereas the variable *huruf* and *kata* uses char data type (character). Note the execution result.

Boolean

Boolean data type is used to keep True / False value. Most language uses non-zero value as true and zero (0) as false. Boolean data type is often used to set a decision, such as on branching like IF... THEN or IF... THEN... ELSE.

Array

Array is a simple structured data. Array will store data with the same (homogeneous) data type in a variable. Each data location is indexed that used as the address of the data. Mode detailed explanation will be described in the chapter.

Record or Struct

As in Array, Record or Struct is a composite data structure. Record is used in Pascal / Delphi whereas Struct is used in C++. Different from array, record data type uses heterogeneous data type. For example, array may store a lot of single type of data, such as integer. Whereas in record, it can accommodate many data with different type of data in a single record, for example, one section may be integer, some section may be character, and other part may be boolean. Record usually accommodates data of an object. For example, a student may have name, address, age, birthplace, and date of birth. Name will use data string, address uses data string, age uses data type single (numeric), birthplace uses data string and date of birth uses data type date. The following is the example of record declaration in Delphi.

Example 5.5. Data type record declaration in Delphi.

```
Type TRecord_Siswa = Record
          Nama_Siswa   : String[30]
          Alamat       : String[50]
          Usia                : Real
      EndRecord
```

Image

Image or picture uses data type graphics. For example, graphics of the development of SMK students, family photo, trip video etc. In modern programming languages especially visual based programming language will likely have a good image support.

Date Time

Date and time is stored in specific format. Variable or constant declares with date data type may keep both date and time. This data type is categorized as composite data type as it forms by several data types.The following is an example of data type written in Visual Basic.

Example 5.6. Usage example of date time data type in Visual Basic.

```
Dim WaktuLahir As Date
WaktuLahir = “01/01/1997”
WaktuLahir = “13:03:05 AM”
WaktuLahir = “02/23/1998 13:13:40 AM”
WaktuLahir = #02/23/1998 13:13:40 AM#
```

Other data type
Subrange

Data type subrange has a range of value sets by the programmer. Such data type has a limited minimum and maximum value. Such data type is support by Delphi. The following is an example of data type subrange in Delphi.

Example 5.7. Data type subrange declaration in Delphi.

```
Type
    BatasIndeks = 1..20
    RentangTahun = 1950..2030
Var
    Indeks          : BatasIndeks
    Tahun           : RentangTahun
```

Enumeration

The following data type has sequential integer element. Each element is mapped to a name variable written in bracket. Such data type may be found in Delphi and declarative programming language such as SQL. The following is an example of data type enumeration in Delphi.

Example 5.8. Usage of data type enumeration.

```
Type
    Hari_dlm_Minggu = (Nol, Senin, Selasa, Rabu,
                      Kamis, Jumat, Sabtu,
                      Minggu)
    Nama_Bulan = (Nol, Januari, Pebruari, Maret,
                   April, Mei, Juni, Juli,
                   Agustus,
                   September, Oktober, Nopember,
                   Desember)
Var
    No_Hari          : Hari_dlm_Minggu
    No_Bulan           : Nama_Bulan
```

In the above example, data type Hari dlm Minggu includes the enumeration ranging from zero, Monday up to Sunday mapped to value 0, 1, to 7. Whereas data type Nama_Bulan includes the enumeration ranging from zero, January up to December mapped to value 0, 1, to 12.

Object

Data type object is used to store value related to objects provided by Visual Basic, Delphi and other GUI based programming language. For example, we have a form with Command button named Command1, we may declare the variable as follows.

Example 5.9. Data type object usage example.

```
Dim A As Command Button
Set A = Command1
A.Caption = “HEY!!!”
A.FontBold = True
```

In this example, variable A is declared as data type object as Command Button. Next, we may set variable A as control Command button on the form (Command1). Thus, we can access all *property, method* and *event* of Command1 object using variable A.

Variant

This type of data type is only in Visual Basic. This is the most flexible data type as it may accommodate all type of data type.

5. PROGRAMING ALGORITHM STRUCTURE

5.2.1. Algorithm Concept

Algorithm is a sequential logical stages to systematically solve a problem. Problem may be anything, but noted that all problems must have an initial condition prior to work on its algorithm. Algorithm concept may be similar to recipe. A recipe usually has a list of needed materials or spices, and sequence and how to build or to cook it. If the material is not available or not calibrated then the recipe may not be built / cook. Likewise if cooking stages is not sequential, a correct result will not be received.

Different algorithms may be applied to a problem with same condition. Algorithm complexity level may be measured by the amount of computation to solve a problem. In general, algorithm with short execution time to solve a problem would have low complexity, while algorithm with long execution time may have much higher complexity. Please note pn the following simple algorithm.

Example 5.10. Algorithm to calculate area of a triangle.

1. Start
2. Read data on base length and height.
3. Area is base multiply by half of the height.
4. Show the area.

5. Stop

The above algorithm is a simple algorithm with only five steps. In this algorithm, there is no repetition or selection processes. All the steps is carried out by only one pass.

For a glance, the algorithm looks right. However, as we look closer the algorithm contains a big fundamental mistake, namely, no limit on value of base and height. What would be happens of value of base length and hight is zero (0) or negative? The result would not be correct. To make sure all input data conforming the requirement, if input of base length and height less than 0 then the program will be terminated. Thus, the following algorithm will be used.

Example 5.11 Refinement of the algorithm to calculate the area of a triangle.

1. Start
2. Read data on base length and height.
3. Check base length and height, if value of base length and height more than zero (0) then continue to step 4 else stop.
4. Area is base multiply by half of the height
5. Show the area
6. Stop

From the above explanation, the main summary of an algorithm are, firstly, **an algorithm must be correct.** Secondly, **an algorithm must stop,** and after stop, an **algorithm must provide the correct result.**

5.2.2. How to write Algorithm

There are three (3) methods to write an algorithm, namely,

- ***Structured English (SE)***

  SE is a good enough tool to depict an algorithm. SE is basically an English, but we can modify it using Indonesian and call it Structured Indonesian (SI). The algorithm such as in Example 5.10 and 5.11 are written in SE. Since it uses daily language, SE is suitable to be used in communicating algorithm to software users.

- ***Pseudocode***

  Pseudocode resembles SE. Due to its resemblance, SE and Pseudocode are considered to be the same. *Pseudo* means imitation or resembled, whereas *code* refer to program code. Thus, *pseudocode* means code that resembles the instruction of the program code. Pseudocode is based on the the actual programming language, such as, BASIC, FORTRAN or PASCAL. PASCAL based *pseudocode* is often used. Sometimes, *pseudocode* refers as *PASCAL-LIKE* algorithm. If Example 5.10 is written in pseudocode based on BASIC, it appears as follows.

Example 5.12. Pseudocode.

```
1. Start
2. READ alas, tinggi
3. Luas = 0.5 * alas * tinggi
4. PRINT Luas
5. Stop
```

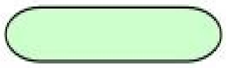
In Example 5.12, the algorithm resembles BASIC. READ and PRINT statements are the *keywords* in BASIC. By using pseudocode, translation process from algorithm to source code can be made easier.

- ***Flowchart***

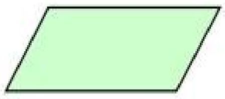
Flowchart is scheme / *chart* that shows the program *flow* in logical manner. Flowchart is a tool to show algorithm using certain notations / symbols. The following section will discuss it in a more detail.

There are several important symbols used to show an algorithm as shown in Figure 5.3.

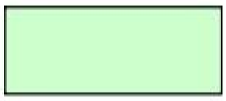
This notation is known as Terminator to show the beginning and the end of an algorithm



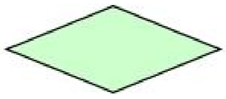
This notation is known as Data to represent the data input or output or to represent data entry operation or printing the result.



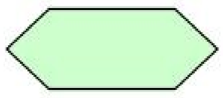
This notation is known as Process to represent a process.



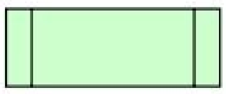
This notation is known as Decision to represent condition selection in a program



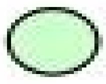
This notation is known as Preparation to set starting value, end value, the increase / decrease value of the counter.



This notation is known as Predefined Process to show the process is done by other subprocess, such as, procedure, sub-procedure, function.



This notation is known as Connector to show continuation of flowchart from pne page to another page.



This notation is known as Arrow to show the data flow from one process to another process.

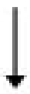
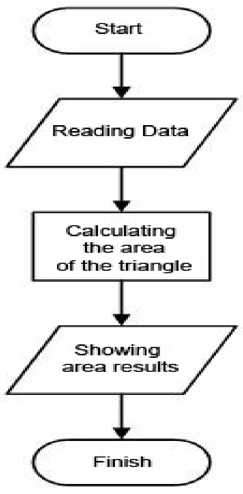
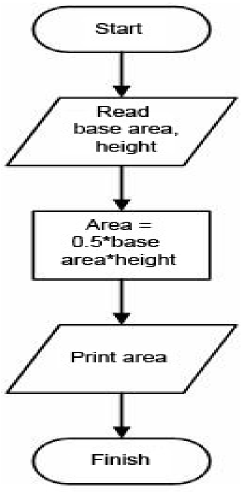


Figure 5.3. Symbols used in flowchart.

There are two (2) types of flowchart, namely, program logic flowchart and detailed computer program flowchart. The program logic flowchart is used to show each step of the computer program in a logical manner and usually is prepared by an system analyst. Whereas the detailed computer program flowchart is used to show the detail instruction- of the computer program and usually prepared by a programmer. If the Example 5.10 is made into program flowchart, it is shown in Figure 5.4.



Program logic flowchart



Detailed Computer Program Flowchart

Figure 5.4. Program flowchart.

5.2.3. Sequential Algorithm.

There are three (3) foundation in algorithm structure, namely, sequencing, branching and looping. An algorithm will usually uses the three (3) structures to solve a problem.

In this section, we will firstly discuss sequential algorithm structure. The sequential structure is similar to a car run in a straight road and no turn or intersection as shown in Figure 5.5. The car will pass kilometers of road until it reaches its destination.

Figure 5.5. Car travels in straight road.

The sequential structure consists of one or more instruction. Each instruction is sequentially executed based on its sequence in source code, and executed after one instruction is completed. Instruction sequence sets the final stage of the algorithm. If the sequence is changed, the final result may be changed. According to Goldshlager and Lister (1988) the sequential structure follows the following rules:

- Each instruction is executed one by one.
- Each instruction is carried out very precise, none is repeated.
- Instruction sequence in the process is the same is action sequence in the algorithm.
- At the end of the final instruction is the end of the algorithm.

Example 5.13. Flowchart to calculate an area.

Create a flowchart to calculate:

1. Volume of a block.
2. Area of a circle.

Solution:

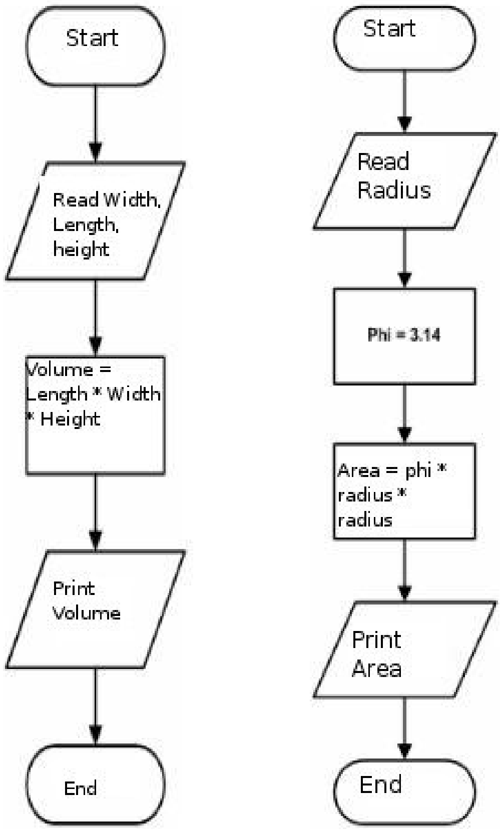
This exercise is a problem for sequential algorithm as no branching and looping is necessary. To calculate the volume of a block, we need to decide on the needed input and output variable. To calculate the volume of a block we need to know the length, width, and height of the block. Volume of a block will be the output. On area of a circle, radius is the input and area will be the output. To calculate the area of a circle, we will need the phi constant. Flowchart of these two (2) problem is shown in Figure 5.6.

Example 5.14. Flowchart to convert temperature.

Create a flowchart to calculate the temperature conversion from Fahrenheit to Celsius using the following formula °C = 5/9 x (°F -32).

Solution:

This exercise uses sequential algorithm. The input variable is F and the output variable is C. Flowchart of this exercise may be seen in Figure 5.7.


Start
Read Width, Length, height
Volume = Length * Width * Height
Print Volume
End
Start
Read Radius
Phi = 3.14
Area = phi * radius * radius
Print Area
End

a. flowchart to calculate a block's volume

b. flowchart to calculate circle's area

Figure 5.6. Flowchart to calculate block's volume and circle's area

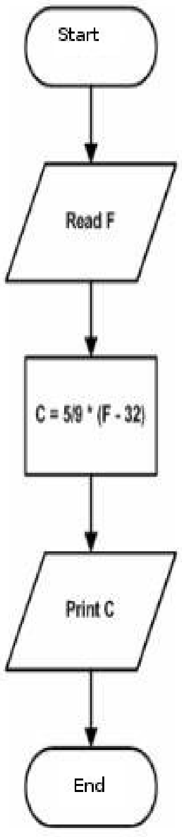


Figure 5.7 Flowchart to calculate temperature conversion

5.2.4. Branching Algorithm

A program will not always follow a sequential structure, sometimes we must change program sequence and jump to certain line in the program. This is known as branching or decision process. This occurs when the car reaches an intersection as shown in Figure 5.7. The driver must decide to turn left or right.

In the branching structure, a program will branched as the required condition is met. In such process, decision symbol in the flowchart must be used. Decision symbol is meant to test a condition. The result will determine which branch is used.

Example 5.15 Branching structure for age limitation problem.

Figure 5.8. The car at an intersection

A rule to watch a movie is as follows, if the age of the spectator is more than 17 years then the spectator is permitted and if less than 17 years then the spectator is not permitted to watch. Create a flowchart for the problem.

Solution:

The above problem is a typical problem for branching structure. The problem uses the statement *if ..... then ....*

The flowchart to solve the above problem is shown in Figure 5.9. In Figure 5.9, it uses the Decision symbol. At Decision symbol the required condition is inspected, whether the age is more than 17 years or not. If the answer yes then the program will produce a text output of “Silahkan Menonton”, whereas if input the age less than 17 years then the program will produce text output “Anda Tidak Boleh Menonton”.

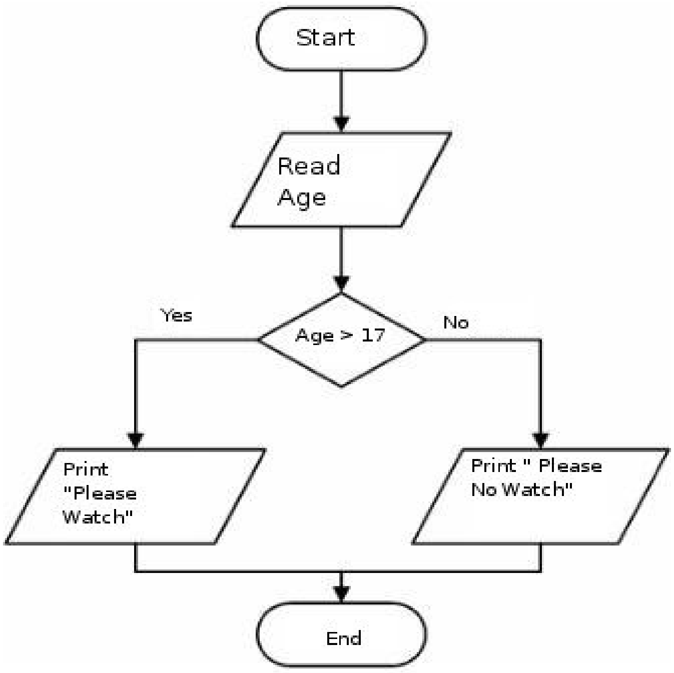


Figure 5.9. Flowchart to solve movie spectator problem.

Example 5.16. Branching structure for the calculation of two numbers.

In a calculation, P = X + Y. If P positive, then Q = X * Y, whereas if P negative then Q = X/Y. Create a flowchart to calculate P and Q

The solution:

> In this example the needed input is X and Y, at the condition inspection is carried out on P value whether positive (including 0) or negative. Please see the resulting flowchart in Figure 5.10.

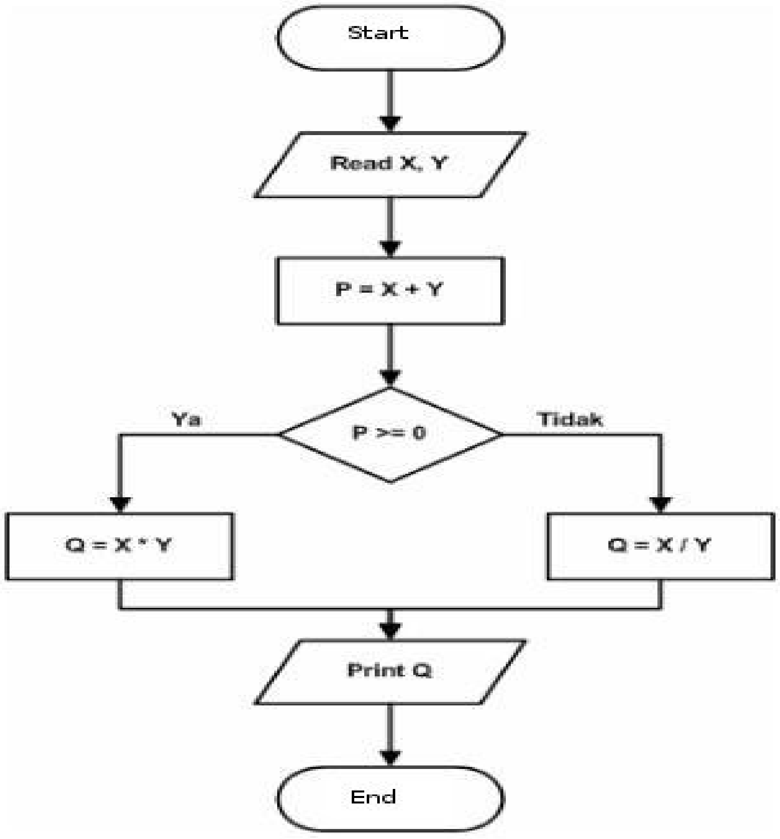


Figure 5.10. Flowchart to solve the calculation of two numbers.

The two examples (5.15 and 5.16) show a simple branching involving one branching. In more complicated problems, we may encounter more branching. We may also encounter a branching structure within a branching structure, or normally called as nested branching. Please see the following example.

Example 5.17. Nested branching for a photocopy problem.

A copy service uses the following rule:

- Subscriber price would be Rp. 75.- / page.
- Non-subscriber would be Rp. 100,- / page if less then 100 pages. If more than 100 pages it would be Rp. 85,- / page.

Create a flowchart to calculate the total cost for someone to copy X number of pages.

Solution:

This example looks complicated. There are two (2) branches. The first branch will look at whether he / she is a subscribed customer or not. The second branch, if he / she is not a subscriber then check if the number of pages more than 100 pages or not.

This is a nested branching problem. Please look carefully at the second requirement.

> ***If*** *he / she is not a subscribed customer,* ***then if*** *he / she make less than 100 sheets* ***then*** *the price is Rp. 100 / page.*

The second if is located within the first if.

Input needed in this problem would be the status of the customer and the number of sheet / page to be copied. Thus, the input variable would be:

- Status – the status of the customer.
- JLF – the number of sheets / pages to be copied.

In addition, there are several other variable namely HPP to store the price per page and TH to store the total cost. Please note that Status using data type character, and this the data should uses “”.

Flowchart to solve the above problem is shown in Figure 5.11.

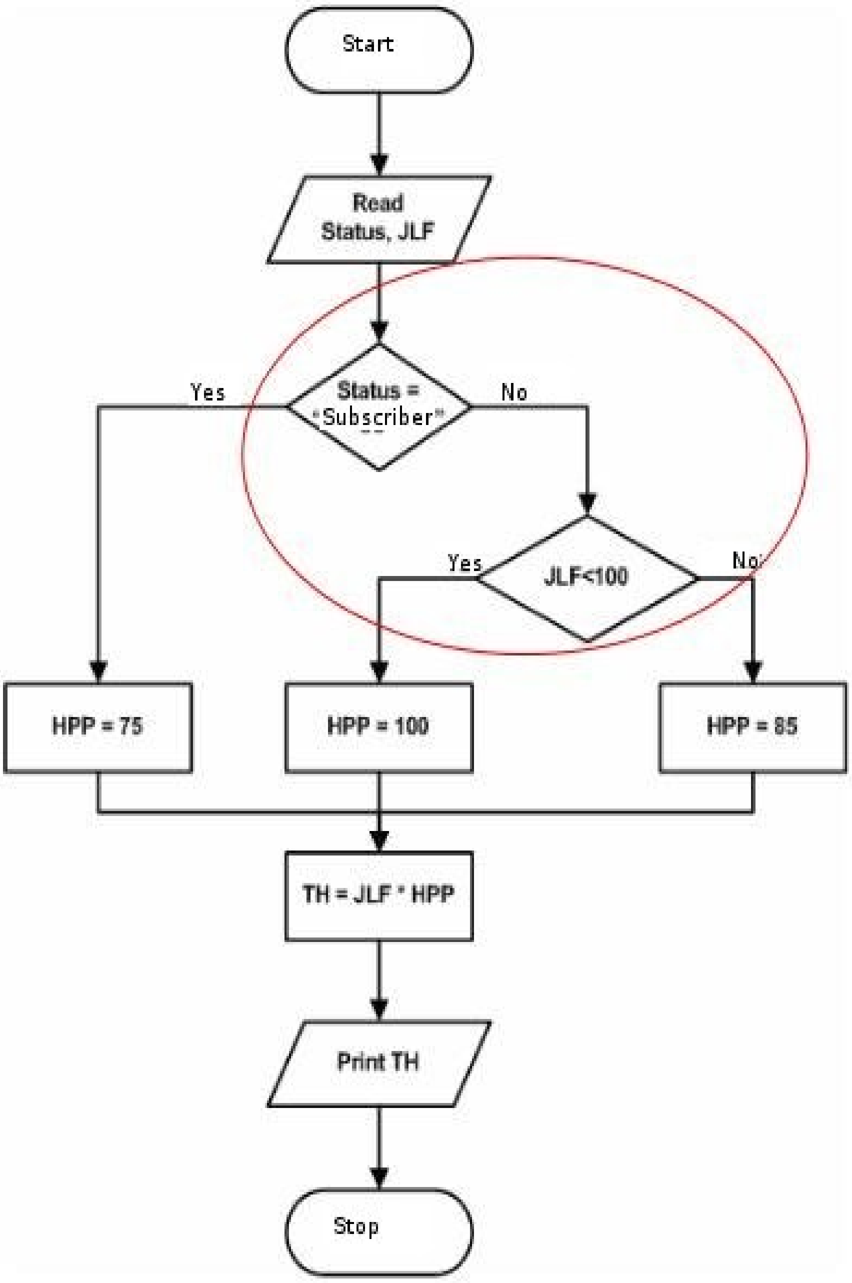


Figure 5.11. Flowchart to solve photocopy pricing problem.

Example 5.18. Nested branching in student's passing grade.

The passing rule of student who take web programming course is as follows.

- If mid test mark is more than 70 then student is final mark would be the same as mid test mask.
- If mid test mask is less than or equal to 70 then student will only be passed if the final mark is equal or more than 60 where the final mark = (mid term mastk x 40%) + (final mark x 60%).

Create a flowchart to solve the problem with output student number, student name, and passing status.

Solution:

In this example, there are two branching. The first branch is to check whether the mid test mark is more than 70. If the mid test mark is less than 70, then check whether the final mark is more than 60.

The needed variable input is the student number (NIM), student name, mid test (NUTS) and final exam mark (NUAS). Whereas the variable output consisted of NA that was used to keep the value and the Status of the end to keep the status of the passing. The output variable is NA to store the final mark and Status to store the passing status.

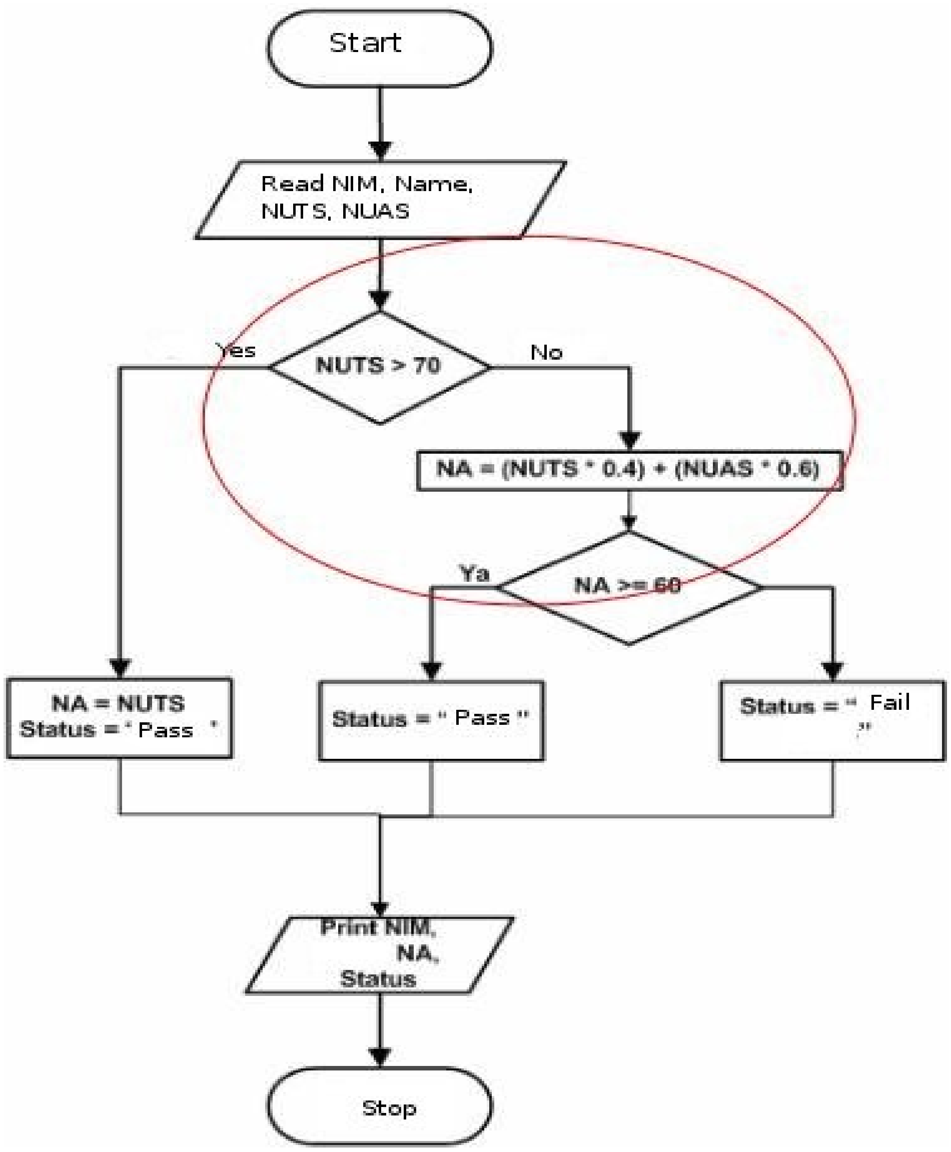


Figure 5.12. Flowchart to solve student passing grade problem.

5.2.5. Looping Algorithm

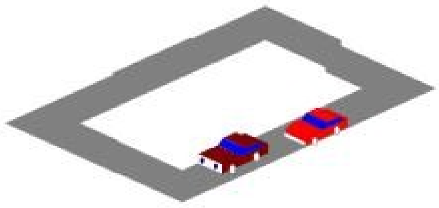


Figure 5.13. Car racing in a circuit.

In many cases, we encounters job that must be repeatedly done. One of the simplest example is a car racing as shown in Figure 5.13. In car racing, participant must follow the circuit tracks as set by the race rule. Whoever the fastest in reaching the finish line, he wins.

In creating a computer program. We sometimes must repeat one or a group of instruction to archive the required result. In computer, a repetition process is easy to carried out. This is due to the advantage of computer as compared to human to do task or repeated instruction without tired, bored, or lazy. Compared to the race car driver, at one time he must be feeling tired driving his race car in circles.

The looping algorithm consists of two parts, namely,

1. **Repetition condition,** the requirement to be met to carry out the repetition. This condition is usually stated in Boolean expression that must be tested whether being true or false.
2. **The repetition body (the loop body),** that is one or more instructions that will be repeated.

The repetition structure is usually started by an initialization section and termination section. **Initialization** is the instructions that are carried out prior to repetition. Initialization is usually used to provide initial value to the variables. Whereas **termination** instruction is carried out after the completion of the repetition.

There are several forms of the repetition, each with its own requirement and characteristics. Several forms may be used for the same case, but some of the form may suitable for a certain case only. The selection of the form may influence the correctness of the algorithm. The correct repetition form depends on the problem.

- **Repetition structure using For.**

Repetition using *For* is the oldest form of repetition in programming language. Almost all programming languages provide this method, despite may be different in syntax. In *For* structure, we need to know how many times the the loop bodies will be looped. In this structure we normally use a variable, namely, loop counter, to take note the number of loop has been done. The value of loop counter may be increasing or decreasing during repetition process. General *flowchart* using *For* structure is shown in Figure 5.14. Please note the usage of symbol *preparation* in the *flowchart.*

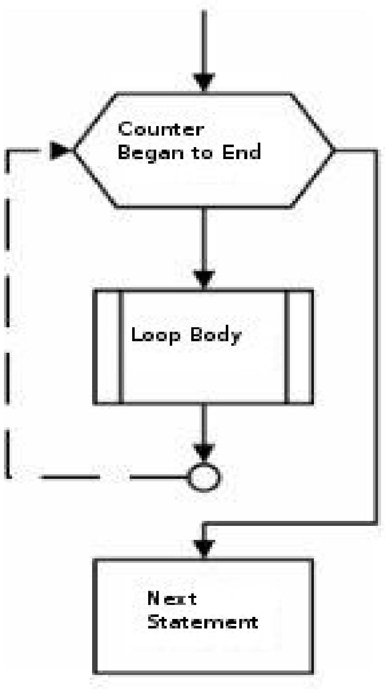


Figure 5.14. Repetition algorithm using For.

In execution of repetition using for, the following steps must be followed:

1. Initialized counter value.
2. Checked whether counter is larger than its maximum / final value. If true then exit the repetition process. If the counter is decreasing, check whether the counter is lower than its minimal / final value. If true then exit the repetition process.
3. Execute the statements / instructions in the loop body.
4. Increase / decrease the *counter* in accordance with the *increment* value. If no increment value, use 1 as default.
5. Repeat step no 2.

It is import to initialized *counter* at the beginning of the loop. If we try to change final value in the *loop* body, it would give not much impact into how many repetition to be carried out.

Example 5.19 Algorithm to print a statement 100 times.

One upon a time, during our elementary school days done an unforgiving mistake, the teacher told us to write something 100 times as a punishment. An example of the statement would be “I will will not repeat such behavior again”. How can an algorithm help in this case?

The solution:

In this example, we need the variable counter, for example, I. Initial value of I is 1 and the final value is 100. Whereas the increment at each repetition is 1. Instruction to print will be repeated until *counter* reaches 100. Flowchart to do the task may be seen in Figure 5.15.

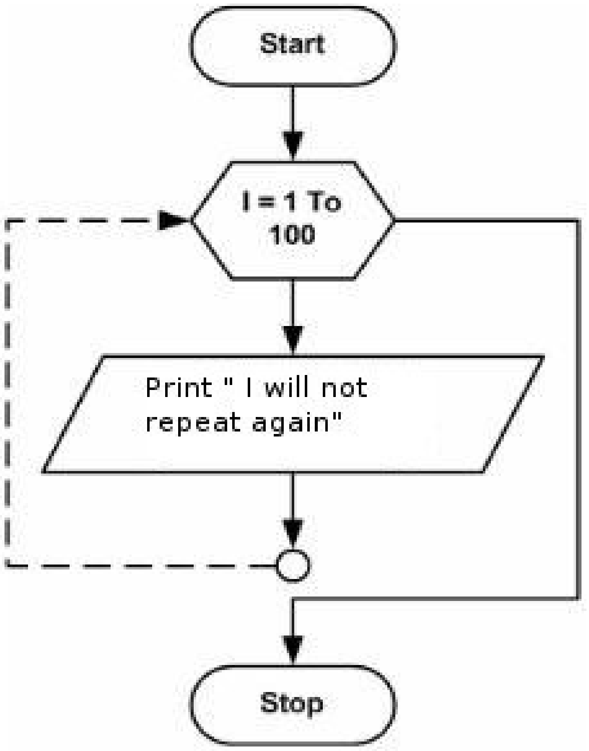


Figure 5.15. Flowchart to write a statement 100 times.

Please note how easy to carry out the repetition. In Figure 5.15 the *increment* is not included, as mentioned earlier if increment is not included then *increment* default to one.

Example 5.20. Flowchart to print the member of an association.

An A association with member 1, 3, 5,.., 19. Create the flowchart to print the member of this association.

The solution:

In this example, we need variable counter, say A (similar to the association name). Its initial value is 1 and the final value is 19. From the pattern it is clear the increment is 2 (1 to 3, 3 to 5, etc). Thus, in every repetition, A increased by 2 The flowchart to solve the problem is shown in Figure 5.16.

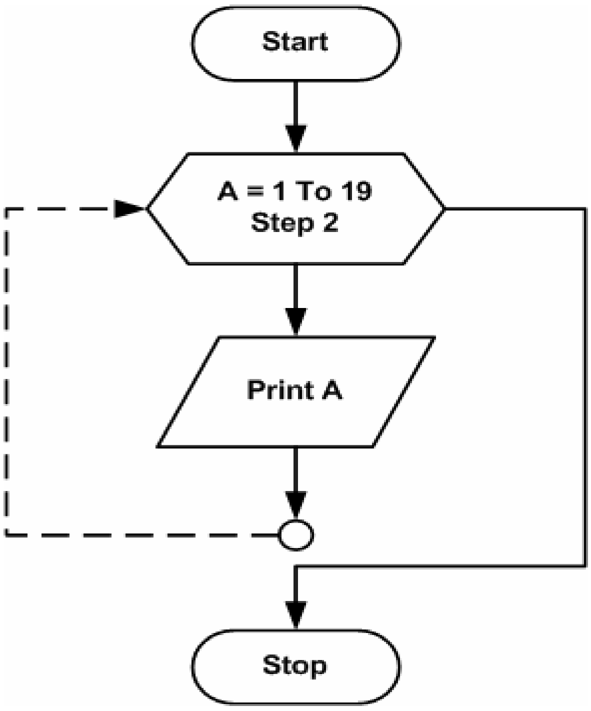


Figure 5.16. Flowchart to print the member of association A.

In Figure 5.16, note the symbol preparation. There is an additional statement, namely, **step 2**. This is the *increment.* For each repetition, *counter* A increases by 2 so that it will be printed as 1, 3, 5,.., 19.

Example 5.21. Determine the results from a repetition *flowchart*

Look to *flowchart i*n Figure 5.17. Determine the result of the *flowchart.*

The solution:

In this example, we tried to determine result of a *flowchart.* What do you think? Let's analyze the *flowchart.*

In the *flowchart,* after start, a process contains a statement A = 1. This is known as **initialization.** It provides the initial value for A = 1. The variable *counter* X is initialized to 1 and end value is 10, without *increment* (or uses the *default increment* to 1). Entering the loop body for the first time, variable A will be immediately printed. Value of variable A is 1. Next process is the statement A = A + 2. This is possibly a rather strange statement, it is needed in a programming. It means substitute the old A value with the new one that has been increased by 2. So A will be 3. Afterwards carry out the second repetition. At this stage, A value is 3, will be printed as 3. After that, A is replaced by A + 2. The new value is 5. And so on. Thus, the output of this flowchart is 1, 3, 5, 7, .., 19.

As we can see, Figure 5.16 and 5.17 provide the same output. Between both flowchart, Figure 5.17 is the recommended flowchart. It is longer but more structured. In addition, not all programming languages provides increment facility.

Look carefully at flowchart in Figure 5.17, especially the position Print A statement. Please try to change its position so that it is placed after A = A + 2 statement. How is the result?

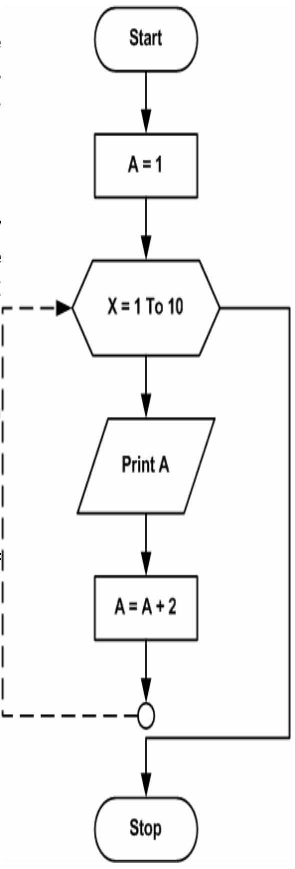


Figure 5.17 Flowchart to print certain number

Like branching structure, we'll find nested repetition structure. Thus, a repetition inside another repetition structure. Please see the following Example 5.22

Example 5.22. Determined results of a repetition flowchart.

Please see flowchart in Figure 5.18. Determine results of the flowchart.

The solution:

In this example, we try to determine the result of nested repetition *flowchart.* What do you think? Let's analyze the *flowchart.*

In Figure 5.18, there is two (2) preparation symbols. The first is to initialize variable *counter X,* and the second is to initialize variable *counter Y.* Variable *counter Y* is located after variable *counter X.* Thus, it means repetition using variable counter Y within repetition using variable *counter X*. It is known as nested repetition.

In this kind of repetition, the flow of program execution is as follows.

- Variabel X will be initialized with *counter* value of 1.
- Variable Y will be initialized with *counter* value of 0.
- Calculate Z as X multiply by Y. Since X = 1 and Y = 0, thus, Z = 0.
- Print X, Y, and Z on screen.
- Repeat the flow, increase Y by 1 while X is 1, as X is not repeated yet.
- Now Z = 1 as X = 1 and Y = 1.
- Print X, Y, and Z on screen again.
- Repeat the flow, increase Y by 1 so that Y = 2, thus, Z = 2.
- Exit Y rotation as final counter of Y = 2 is reached.
- Increase X by 1 so that X = 2, and do the same process with Y again.
- Repeat the process again to reach the counter X = 3. Thus, the final result of the flowchart is as follows.

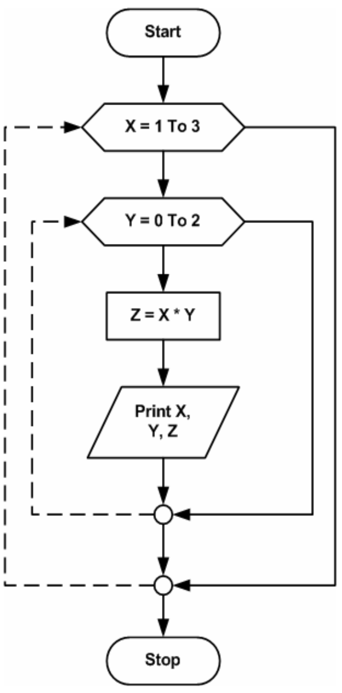


Figure 5.18. Flowchart with nested repetition.

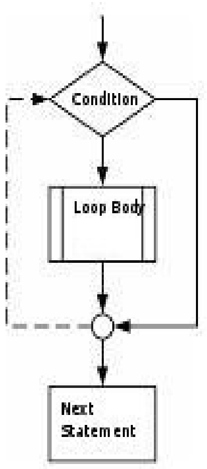
X	Y	Z
1	0	0
1	1	1
1	2	2
2	0	0
2	1	2
2	2	4
3	0	0
3	1	3
3	2	6

From Example 5.22, there are rules that must be satisfied in nested repetition, namely.

- Each repetition (the loop body) must have respective variable counter.
- Repetition must not overlap.
- Repetition structure using While.

In repetition with *For*, the number of repetition is known as we set the start and stop value in the beginning of the repetition. What if we don't know how many repetition should be done? Repetition using *While* is the answer to such problem. Similar to For, repetition using *While* structure is supported by most programming language with different syntaxes.

While structure will repeat the loop body as long as the *While* condition is hold true. Thus, there is no need to know the number of repetition. General flowchart for *While* structure is shown in Figure 5.19.



Picture 5.19. *Flowchart* umum *While.*

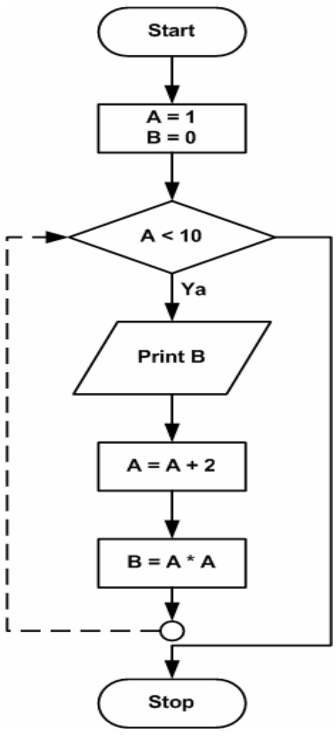
In Figure 5.19, preparation symbol is no longer used. However, decision symbol is used to control the repetition.

In While repetition, we normally need to initialize variables.

Example 5.22 Repetition with *While* to print the variable.

Examine *flowchart* in Figure 5.20. How is the *output* of the *flowchart*?

The solution:



Examine Figure 5.20. Can you describe the output of the flowchart? Take a closer look at the flowchart execution stages as follows.

1. In the flowchart, there are two variables A and B. Initialized the variables so that A = 1 and B = 0 before doing the loop. Variable A is the *counter.*

2. In decision symbol, the A value is checked whether met the condition (< 10). If True then the next statements is executed, if not then the loop will be stopped. At the beginning of the execution this condition is met as A = 1.

3. Do Print B.

4. A value is increase by 2. Thus, the new A will be 3. Whereas the B = 9 as a result of A * A with A = 3.

5. The loop proceeds and check whether A is less then 10. At this stage, A = 3, thus, the condition is met. Steps repeated to step no. 3, until variable A is no longer met the requirement (< 10). Thus, the output of this flowchart is 0, 9, 25, 49, 81.

As in repetition using For, repetition with While may be used for nested repetition. The rules and methods is the same as repetition with For.

Figure 5.20Flowchart repetition with while to print certain numbers.

Several programming languages provides repetition by means of *Do... Loop* and *Repeat.. Until.* These two (2) methods resembles While, the difference is located in the condition. In While structure, the condition is check before the loop body. Whereas Do... Loop and Repeat... Until, the condition inspection is carried out after the loop body.

5.3. ARRAY MANAGEMENT

Variable array has been briefly touched. In this section, the array will be described in a more detail.

5.3.1. Array Concept

Thus far, the used variables are common variable with a name and point to a certain value in numeric or string. New value may be given to the variable by removing the old value. Would it be possible to store several values into the variable and keep them all? The solution is using index in the name of the variable. Such method is called array.

Array is a data structure that stores a collection of elements with same data type. Each elements may be accessed through its index. Index of array must be a sequential data type, such as, integer or string. Array may be pictured as locker with series of storage with sequential number as shown in Figure 5.21.To store and to retrieve from a certain box, we need to know the number of the box.

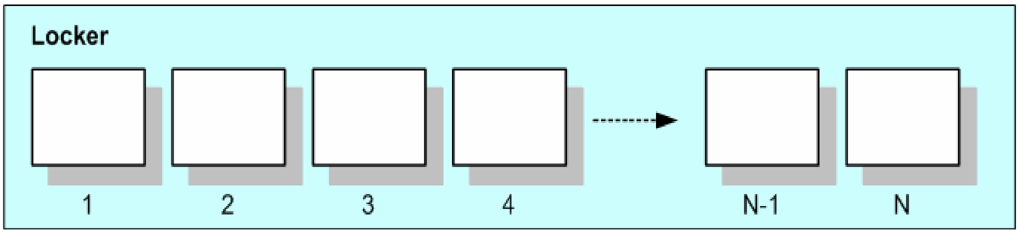
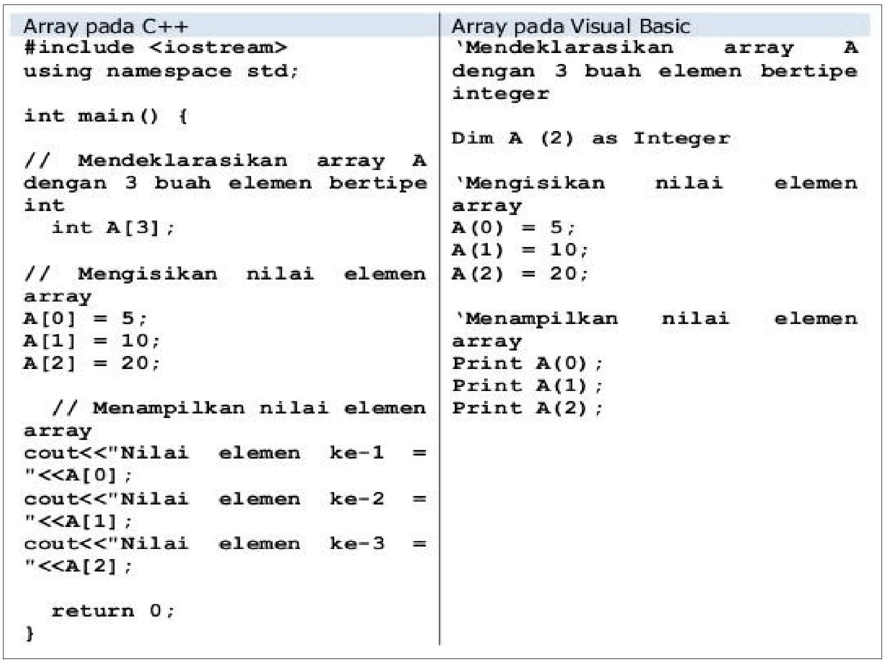


Figure 5.21. Locker with many indexed storage box.

In variable array, we are not only set the data type, but also the number of elements in the array or the upper index limit of the array. In many programming languages, such as, C++, Visual Basic, and several others, the index in an array starts at 0 not 1. Method to write variable array may be different from one programming language to another. However, in variable array, the data type and index limit must be set. To fill the data into the array, we need to specify the index number of the array to which the data will be stored or retrieved.

The declaration example, store and retrieve data in an array as follows.

Example 5.23. Using an array in C++ and Visual Basic.



Array pada C++

```
#include <iostream>
using namespace std;

int main() {

// Mendeklarasikan array A
dengan 3 buah elemen bertipe
int
  int A[3];

// Mengisikan nilai elemen
array
A[0] = 5;
A[1] = 10;
A[2] = 20;

  // Menampilkan nilai elemen
array
cout<<"Nilai elemen ke-1 =
"<<A[0];
cout<<"Nilai elemen ke-2 =
"<<A[1];
cout<<"Nilai elemen ke-3 =
"<<A[2];

  return 0;
}
```

Array pada Visual Basic

```
'Mendeklarasikan array A
dengan 3 buah elemen bertipe
integer

Dim A (2) as Integer

'Mengisikan nilai elemen
array
A(0) = 5;
A(1) = 10;
A(2) = 20;

'Menampilkan nilai elemen
array
Print A(0);
Print A(1);
Print A(2);
```

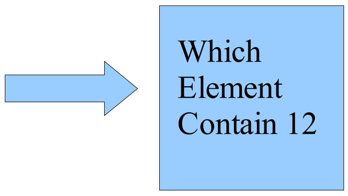
Please examine the two (2) above codes. In the variable declaration the maximum index number of the array is 2 but the number of elements is 3 as index starts from 0 nor 1.

5.3.2. Looking for data in an Array

One of the common problem in an array is how to look for certain element in an array. For example, in the above locker in Figure 5.21 available 100 box. Then, we need to find locker belongs to one of the student named “Rudy”. Another example, in a school with lots of students, we are asked to find data of a student based on certain name or certain address. Please examine the following example.

Example 5.24. Searching on an array.

A[0] = 23
A[1] = 22
A[2] = 45
A[3] = 12
A[4] = 10
A[5] = 34



In this example, we are asked to find an element containing the number 12 from an array. There are six (6) element in the array. How do you think to solve the above algorithm?

The general method and the easiest is by using linear search. In this past, this method is consider inefficient as it requires significant amount of time. However, as computer becoming more powerful, execution time of linear search method is no longer a problem. Linear search compares the element of the array with the value we are looking for, One by one starting from the first element.

Applying the method in Example 5.24, program flow will take the following steps.

- Set the number we are looking for, namely 12.
- Retrieve the first element A[0], compare its content (in this case 23) with the one we are looking for. If it is the same then stop.
- If false then continue with the next element A[1], compare the content with the one we are looking for. If it is the same then stop.
- If false then continue to the next element until we find the element.

Example 5.24 will give the result that element A[3] is 12. Describe the flow in flowchart is shown in Figure 5.22.

- I is the variable counter and index.
- N is the index upper limit.
- Bil[I] is the name variable array containing numbers.
- A is the variable that we are looking for.

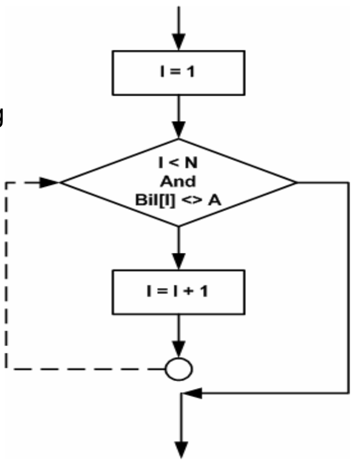


Figure 5.22. Flowchart to search on number in an array.

Flowchart in Figure 5.22 uses While structure to do the repetition. Two (2) conditions must be met, namely, I < N and Bil[I] <> A. Meaning that I must be less than the upper limit of the index N and the content of Bil[I] is not the same of the one we are looking for, namely A, and, thus, the searching process proceeds to the next index. As long as the condition is met, the searching continues. Note that we uses “and” that means both condition must be met. Searching will be terminated if one of the conditions or both conditions no longer met. Thus, if Bil[i] is the same as A, searching process will be terminated as While condition is no longer hold true.

5.3.3. Ordering data in an array.

another common problem in an array is to order or to sort the data in an array. Examine the example 5.24. In the example, it is clear that the element in not sorted. How can we sort the element from the large number to smaller ones or the reserve?

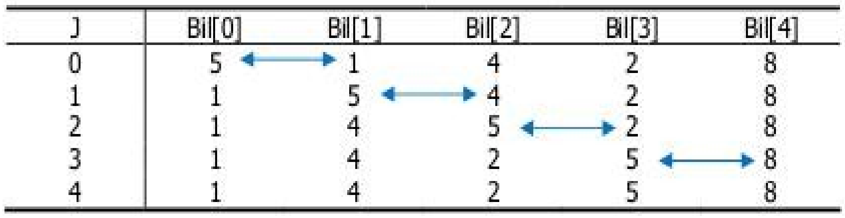
There are several algorithm can be used to sort a collection of number, such as, bubble sort, selection sort, shell sort, quick sort, etc. In this book, we will discuss bubble sort. Although the performance may not as good as other, it is easy to understand and commonly used. Examine the following example.

Example 5.24. Sorting using *bubble sort.*

For example a variable array Bil with 5 elements, I.e., "5 1 4 2 8". Sort from the smallest value to the highest value.

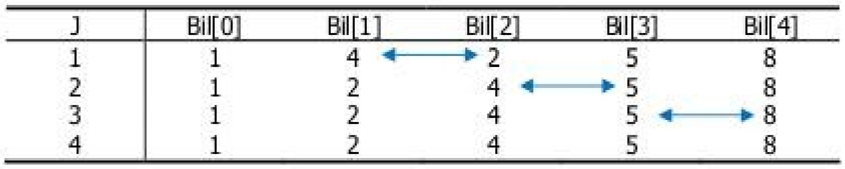
The solution:

> We will use bubble sort to sort an array. Bubble sort is carried out by comparing two number in consecutive location. If the order is correct, then continue to the next two numbers. If wrong order, then switch the place of the two numbers.
>
> Let's apply the algorithm. Examine the following table array. The initial condition of J =0. First, compare Bil [0] and Bil [1]. Bil [0] = 5 whereas Bil [1] = 1. Use bubble sort rule, Bil [0] is not in a correct location as it is bigger than Bil [1]. So we must switch the contents of these two element in the array so that Bil [0] = 1 and Bil [1] = 5 (examine the line of J = 1). Next step is to compare Bil [1] with Bil [2]. Bil [1] = 5 and Bil [2] = 4, so we must switch the contents from this element (examine the line of J = 2). Continues until we compare Bil [3] with Bil [4].



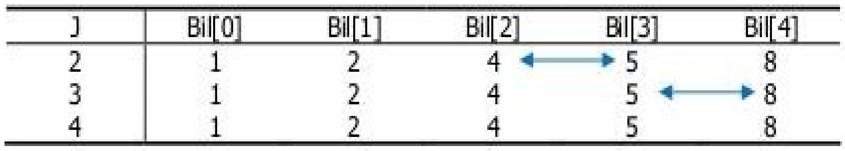
J	Bil[0]	Bil[1]	Bil[2]	Bil[3]	Bil[4]
0	5 ↔	1	4	2	8
1	1	5 ↔	4	2	8
2	1	4	5 ↔	2	8
3	1	4	2	5 ↔	8
4	1	4	2	5	8

As shown, the last position of the table is not fully sorted. As only one pass is performed starting with Bil [0]. We will carry out the comparison again, but starts at Bil [1] since Bil [0] should be the smallest number now. So that our table will be as follows.

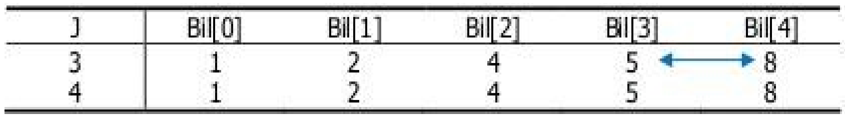


J	Bil[0]	Bil[1]	Bil[2]	Bil[3]	Bil[4]
1	1	4 ↔	2	5	8
2	1	2	4 ↔	5	8
3	1	2	4	5 ↔	8
4	1	2	4	5	8

In the table above, the sequence is correct. However, the algorithm is not completed as it must check the next rounds. It needs to check two (2) more round to compare Bil [2] and Bil [3]. The two (2) tables are as follows.



J	Bil[0]	Bil[1]	Bil[2]	Bil[3]	Bil[4]
2	1	2	4 ↔	5	8
3	1	2	4	5 ↔	8
4	1	2	4	5	8



J	Bil[0]	Bil[1]	Bil[2]	Bil[3]	Bil[4]
3	1	2	4	5 ↔	8
4	1	2	4	5	8

Shown in Figure 5.23 is the flowchart.

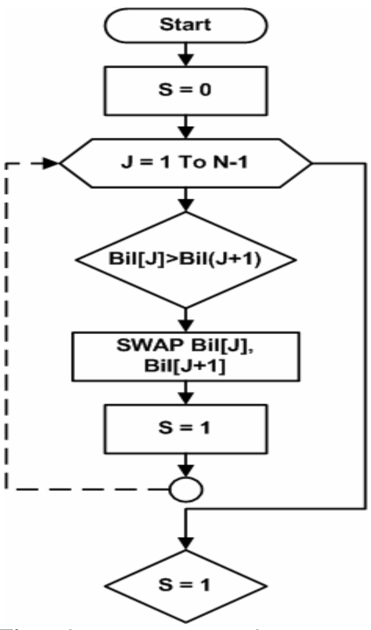


Figure 5.23. Flowchart to sort numbers.

5.4. FILE OPERATION

File is often used to store data to be more permanent. Normally, data and outputs will be lost after the program terminated, so file is used to keep the data. There are two (2) kind of files, namely, program file and data file. Program file contains program code, whereas data file contains data. There two (2) kinds of data file, namely, sequential data file and random access data file. The main difference between the two is shown in the following table.

Sequential data file	**Random access data file**
Record / data must be sequentially read from the first record.	Record may be random.
Field length of each record may be different.	Field length of each record must be the same.
Modify and adding a record is more difficult.	Modify and add a record is easy.

5.4.1. Algorithm to write data to a file

The algorithm to write data for sequential data file and random access data file is the similar, the only difference is the mode used. The followings is the algorithm to write data in Structured English (SE).

```
Open "mode", <buffer number>, "file name"
Write <record number>, field 1, field 2, .. field n
Close buffer number
```

Mode O tells that the file is opened for writing.

Example 5.25. Example to implement algorithm to write data.

For example, we have a data file named “siswa.dat” with field name of the student, the address, the telephone number. To write the data would be as follows.

```
Open "O", #1, "siswa.dat"
Write #1, <nama>, <alamat>, <no.telepon>
Close #1
```

Notation #1 shows that siswa.dat is placed in buffer no 1. This notation must be the same and used throughout the program. It means that we placed a file with buffer #1 then when open, write, read and close must use this notation. Likewise when we placed it in buffer no #2.

.

5.4.2. Algorithm to read data file

The algorithm to read data is similar to write data, but uses mode I instead of O. I means input so that data file is open to be read. The followings is the algorithm to write data in Structured English (SE).

```
Open "modus", <buffer number>, "nama file data"
While not EOF:
    Input <record number>, field 1, field 2, .. field n
    Print field 1, field 2, .. field n
End while
Close buffer number
```

The While Not EOF statement is used to check whether it reaches the last line of data or end of file (EOF). If false it will read all data and print until it reaches last line. Input statement is used to read data from file into the program. While statement print is used to print onto screen.

The example 5.26. Example of algorithm to print data.

Data file named “siswa.dat” similar to example 5.25 with field student's name, the address, the telephone number. To read the data is as follows.

```
Open "I", #2, "siswa.dat"
```

```
While not EOF:
        Input #2, <nama>, <alamat>, <no.telepon>
        Print <nama>, <alamat>, <no.telepon>
End while
Close buffer number
```

5.5. SUMMARY

- Variable is the location to store or to remove value and to retrieve it as needed. Every variable has a name / identifier and value.
- Constant is a variable with fixed value and cannot be changed.
- Data type is the type of data that can be process by the computer that satisfy the requirement in computer programming.
- Data type could be grouped into primitive data type and composite data type. The primitive data type includes numeric, character, and boolean. Whereas the composite data type includes array, record / struct, image, date time, subrange, enumeration, object and variant.
- An algorithm is logical sequence to solve a problem that systematically build. An algorithm must be correct and stop when completed. After completion, an algorithm should provide a correct result.
- An algorithm could be written in Structured English, Pseudocode And Flowchart.
- Sequential structure consists of one or more instruction. Every instruction is executed based on the sequence.
- In branching structure, program will change its sequence if the required condition is met.
- Repetition structure consists of repetition condition and loop body and can be done using For and While.
- Array is data structure to store a collection of elements with the same type, each element may be directly accessed through its index. Searching method in an array may uses linear search whereas ordering it may use bubble sort.
- Data file may be sequential or random access. Method in reading and writing is distinguished by the mode.

5.6. EXERCISE.

1. Determine whether the variable name is right or wrong. If it is wrong, provides the reasons.

   a. nama.guru
   b. NamaGuru
   c. 2x
   d. harga/buku
   e. hargaPerBuku

2. Determine match data type for the following variable (not variable name) and provide the reasons.

   a. total student
   b. Body weight
   c. Body height
   d. Student name
   e. Birthplace
   f. Birth date

3. Examine the Example 5.3.

   a. If in program A all variables (x, y and z) are declared as data type int, how much is z?
   b. If in program A all variables (x, y and z) are declared as data type float, how much is z?
   c. If in program B, variable x is declared as float, how much is z?

4. There are two glasses A and B, glass A contains red color solution, glass B contains yellow color solution. Mixed the content of these two glasses, so that glass A contains yellow solution and glass B contains red solution. Create the algorithm in Structured English (SE).

5. Varible A = 6, Variable B = 10. Create a flowchart to switch the value of A and B so that variable of A = 10 and variable B = 6.

6. PT. Sandang Nyaman plan to use computer to calculate weekly payment of its employees. The needed data is the employee name and the working hours of a week. The hourly payment rate is Rp. 4500,-. Create the flowchart for this problem with the output the employee name, their working hours and the received payment.

7. Same as problem no. 6, but if the working hours exceeds 25 hours per the week then it is considered overtime. The hourly overtime payment is one and a half of the hourly payment in normal condition. How is the flowchart?

8. A seller of the primary school textbook is trying to attract buyers with the following provisions:

   - If the total book is less then or equal to 100 copies, then the buyer does not receive any discount.
   - If the total book more than 100 copies but less or equal to 200 copies, then the first 100 copies received 5% discount, and the rest receive 15% discount,
   - If the total book more than 200 copies, then the first 100 copies receive 7% discount, for the second 100 copies receive 17% discount and the rest receive 27% discount.