Session 8: Flowchart

8.1 Introduction

A flowchart is a pictorial representation of an algorithm. It hides all the details of an algorithm in an attempt to give the big picture; it shows how the algorithm flows from the beginning to the end. The basic objective of a flowchart is to show an algorithm in pictorial form. This allows programmers to be free from the syntax and details of a programming language, therefore giving them to more time to deal with the details of the problem being solved.

8.1.1 Auxiliary Symbols

A flowchart is composed of a combination of symbols. Some symbols are used to enhance the readability or functionality of the flowchart. They are not used directly to show instructions or commands. They show the start and stop points, the order and sequence of actions, and how one part of a flowchart is connected to another. These auxiliary symbols are shown in Figure 8-1.

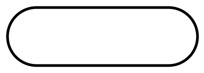
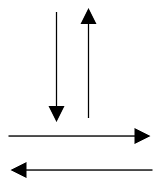
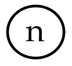
SYMBOL	NAME	DESCRIPTION
	Terminal	Shows the beginning or end of an algorithm
	Flow Lines	Shows the action order in an algorithm
	Connector	Shows the continuity of the algorithm on the next page

Figure 8-1: Auxiliary symbols in flowcharting

A. Start and Stop

An oval is used to show the beginning or end of an algorithm. When is used to show the beginning of an algorithm, write the word START in the oval. When it is used to show the end of an algorithm, write STOP in the oval.

One of the first rules of structured programming is that each program should have only one entry point and one exit. This means that a well-structured flowchart should have one and only one START and one and only one STOP. For example, a flowchart for a program that does nothing is shown in Figure 8-2.

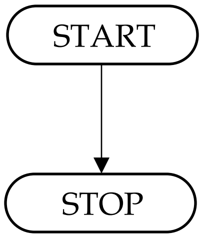


Figure 8-2: Start and stop symbols

B. Flow Lines

Flow lines are used to show the order or sequence of actions in a program. They connect symbols. Usually, a symbol has some entering and some existing lines. The START oval has only one exiting line. The STOP oval has only one entering line. We have already shown the use of flow lines in Figure 8-2.

C. Connectors

You use only one symbol, a circle with a number in it, to show connectivity. It is used when you reach the end of the page, with your flowchart still unfinished. At the bottom of the page, use a connector to show that logic flow is continued at the top of the next page. The number in the connector can be a simple serial number or it can be a combination of a page and symbol in the form *page.number*. Figure 8-3 shows an off-page connector.

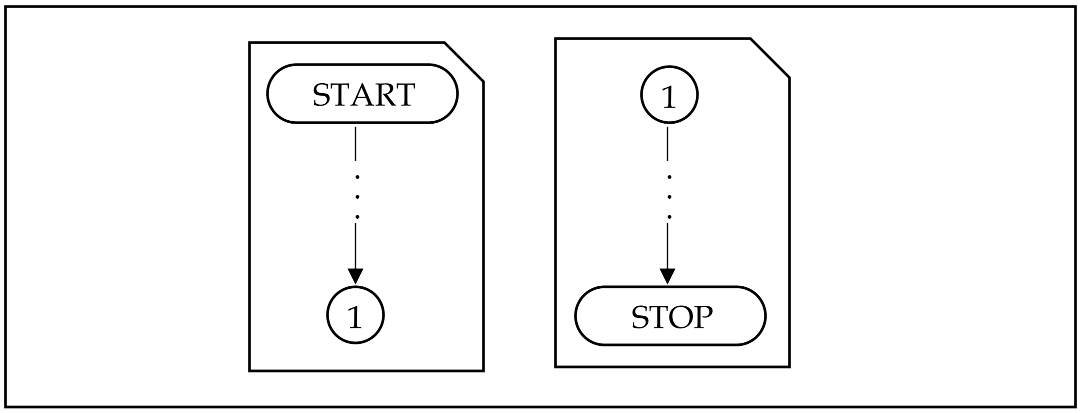


Figure 8-3: Connectors

8.1.2 Main Symbols

Main symbols are used to show the instructions or actions needed to solve the problem presented in the algorithm. With these symbols, it is possible to represent all five structured programming control structures: sequence, decision, repetition.

A. Sequence Statements

Sequence statements simply represent a series of actions that must continue in a linear order. Although the actions represented in the sequence symbol may be very complex, such as input or output operation, the logic flow must enter the symbol at the top and flow out at the bottom. Sequence symbols do not allow any decisions or flow changes within the symbol.

There are five sequence symbols: null, assignment, input/output, module call, and compound statement. The latter four are shown in Figure 8-4.

Null Statement

It is worth noting that do nothing is a valid statement. It is commonly referred to as a null statement. The null statement is considered a sequence statement because it cannot change the flow of execution in a program. There is no symbol for a null statement. It is simply a flow line. Figure 8-2 is an example of a null statement.

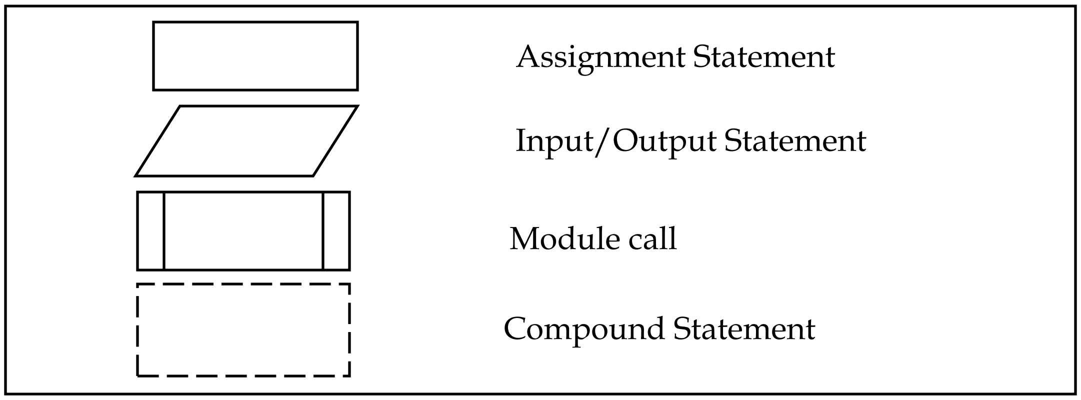


Figure 8-4: Sequence symbols

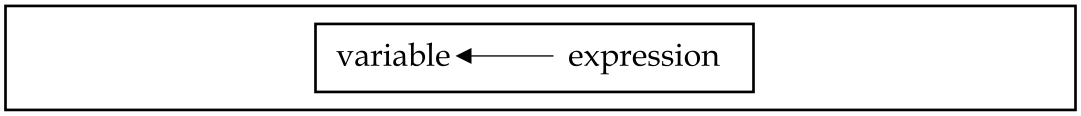


Figure 8-5: Assignment statement

Assignment Statement

The assignment statement is shown using a rectangle. Within the rectangle, the assignment operator is shown as a left – pointing arrow. At the right side of the arrow is an expression whose value must be stored in the variable at the left side. Figure 8-5 shows an assignment statement.

Input/Output Statement

A parallelogram is used to show any input or output, such as reading from a keyboard or writing to the system console.

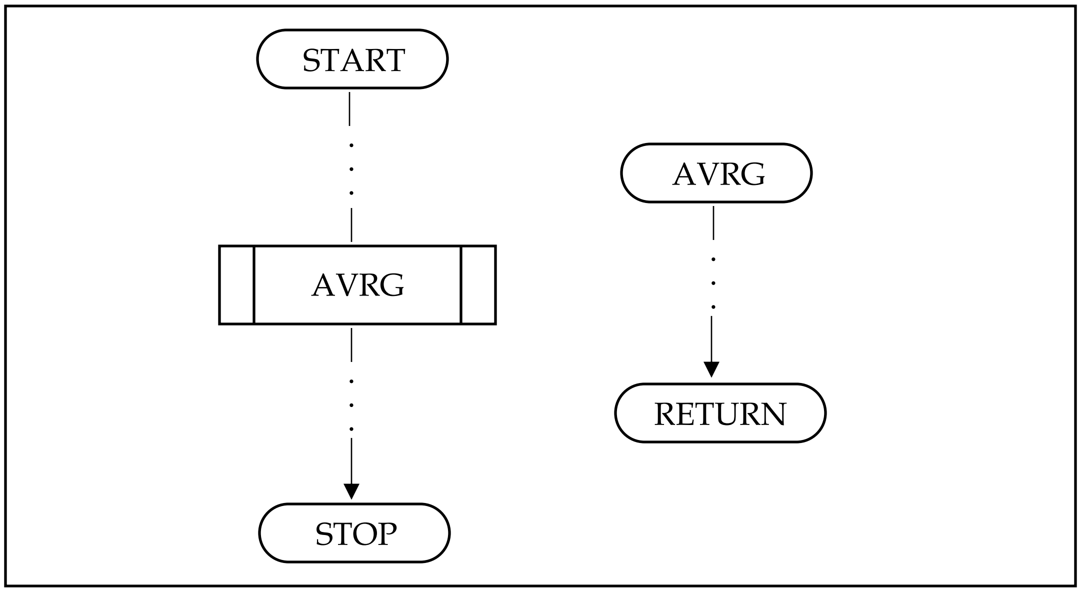


Figure 8-6: Module-call statement

Module-call Statement

The symbol for calling a module is a rectangle with two vertical bars inside. The flowchart for the called module must be somewhere else. In other words, each time you see a module-call statement, look for another flowchart with the module name (Figure 8-6).

Compound Statement

Although there is no actual symbol to show a compound statement, we encapsulate all statements that make a compound statement in a broken-line rectangle.

B. Selection Statements

Unlike the sequence statement, selection statements can cause the flow of execution of the program statements change. They allow the execution of selected statements and the skipping of other statements.

Two-way Selection

The two – way selection symbol is the diamond. When it is used to represent an *if – else* construct, the true condition logic is shown on the right leg of the logic flow nd the false condition, if present, is shown on the left leg of the logic flow.

With the *if – else*, there must always be two logic flows, although often one of them is null. Finally, the construct ends with a connector where the true and false join. In this case, the connector has nothing in it.

Although you will often see decisions with the flow from the bottom of the diamond, this is not good style. Even when one of the flows is null, it still must flow from the left or right side of the diamond. Figure 8-7 shows the use of the decision symbol in the *if – else* construct.

On each branch, you are allowed to have one and only one statement. Of course, the statement in each branch can be a null or compound statement. But only one statement is allowed in each branch; not less, not more.

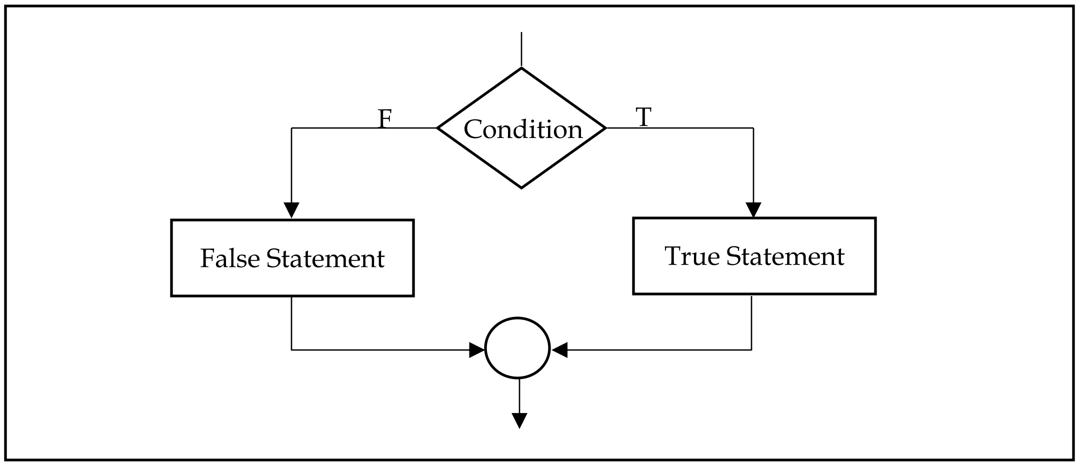


Figure 8-7: Two-way selection

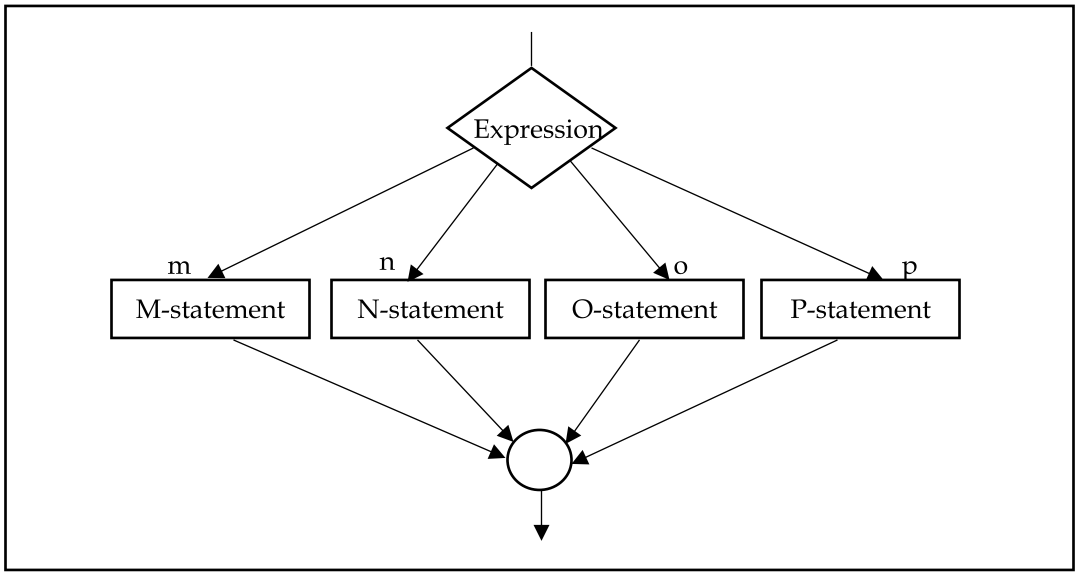


Figure 8-8: Multiway selection

Multiway Selection

The application of the selection symbol used in structured programming is multiway selection (Figure 8-8). As you can see, you can have as many branches as you need. On each branch, you are allowed to have one, and only one, statement. Of course, the statement in each branch can be null or a compound statement. But remember that only one statement in each branch; not less, not more.

C. Looping Statements

There are three looping constructs, namely; *for*, *while*, and *do-while*.

For Construct

The *for* construct is counter – controlled loop structure. It is actually complex construct that has three parts, any of which be null: (1) the loop initialization, which normally sets the loop counter; (2) the limit test; and (3) the end-of-loop action statements, which usually increment a counter. Since the *for* construct is a pretest loop, it is possible that the loop may not be executed. If the terminating condition is true at the start, the body of the *for* construct is skipped.

As is the case with all structured programming constructs, the body of the loop contain one and only one statement. As is the case with other constructs, this one statement can be null or a compound statement. Figure 8-9 shows the *for* construct.

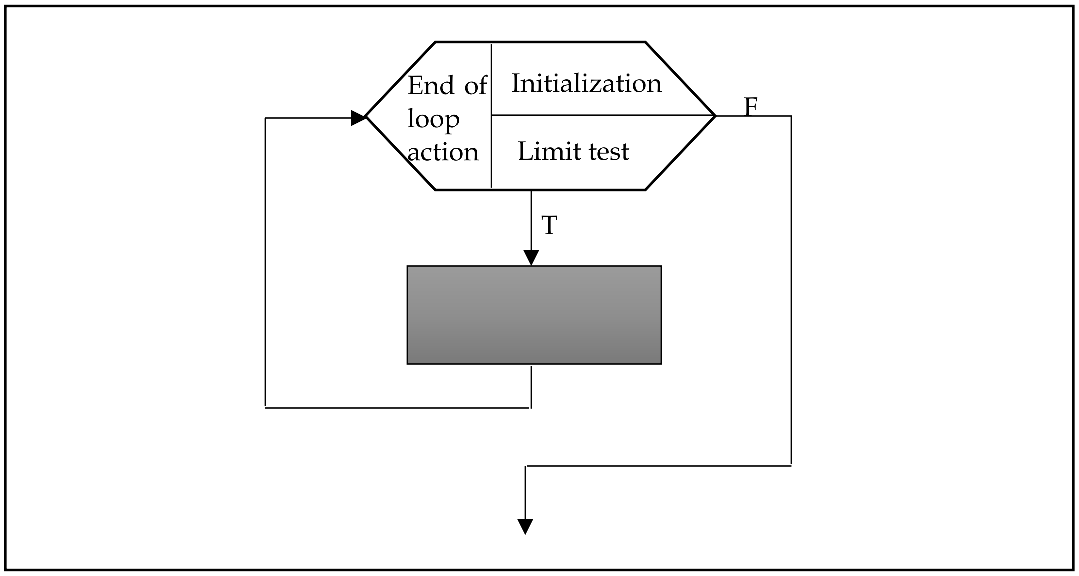


Figure 8-9: *for* construct

while construct

The second looping construct is the *while* construct. The major difference between the *for* and *while* loop constructs is that the *while* construct is not counting loop. Both are pre-test loops; this means that, like the *for*, the body of the *while* construct may never be executed.

You use the same basic symbol for the *while* construct, but because there is only a limit test, the internal divisions are not necessary. Figure 8-10 shows the basic format of the *while* construct.

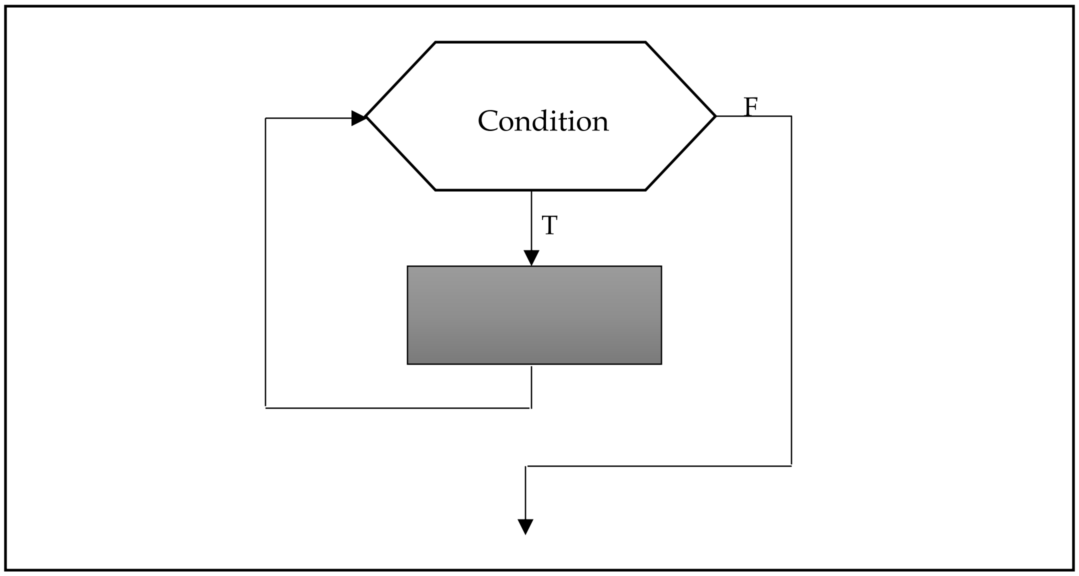


Figure 8-10: *while* construct

***do – while* construct**

The third application of the loop symbol is the *do – while* construct (Figure 8-11). Because of the inherent differences between the *for* and *while* constructs and *do-while* constructs, it is presented differently in a flowchart. There are two major differences between the *while* and the *do-while*:

1. A *while* construct is a pre-test construct. The *do- while* construct is a posttest construct.
2. The body of a while construct may never be executed. The body of a *do-while* construct is executed at least once.

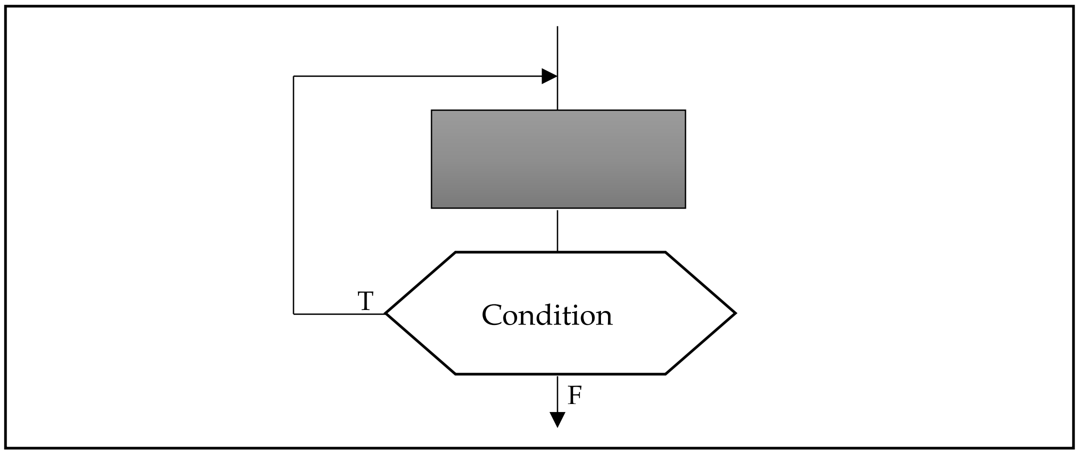


Figure 8-11: *do – while* construct

Exercise

(a.) Draw a flowchart diagram for the example algorithms in Session 7; starting from Algorithm 7-2 to 7-4.

(b.) Draw a flowchart diagram for the exercises in Session 7.