Algorithm Writing Technique

Identifying Number

Each algorithm is assigned an identifying number as follows: Algorithm 4.3 refers to the third algorithm in Chapter 4; Algorithm P5.3 refers to the algorithm in Solved Problem 5.3 in Chapter 5. Note that the letter "P" indicates that the algorithm appears in a problem.

Steps, Control, Exit

The steps of the algorithm are executed one after the other, beginning with Step 1, unless indicated otherwise. Control may be transferred to Step *n* of the algorithm by the statement "Go to Step *n*." For example, Step 5 transfers control back to Step 2 in Algorithm 2.1. Generally speaking, these Go to statements may be practically eliminated by using certain control structures discussed in the next section.

If several statements appear in the same step, e.g.,

Set K := 1, LOC := 1 and MAX := DATA[1].

then they are executed from left to right.

The algorithm is completed when the statement

Exit.

is encountered. This statement is similar to the STOP statement used in FORTRAN and in flowcharts.

Comments

Each step may contain a comment in brackets which indicates the main purpose of the step. The comment will usually appear at the beginning or the end of the step.

Variable Names

Variable names will use capital letters, as in MAX and DATA. Single-letter names of variables used as counters or subscripts will also be capitalized in the algorithms (K and N, for example), even though lowercase may be used for these same variables (k and n) in the accompanying mathematical description and analysis. (Recall the discussion of italic and lowercase symbols in Sec. 1.3 of Chapter 1, under "Arrays.")

Assignment Statement

Our assignment statements will use the dots-equal notation := that is used in Pascal. For example,

Max := DATA[1]

assigns the value in DATA[1] to MAX. In C language, we use the equal sign = for this operation.

Input and Output

Data may be input and assigned to variables by means of a Read stateme. with the following form:

Read: Variables names.

Similarly, messages, placed in quotation marks, and data in variables may be output by means of a Write or Print statement with the following form:

Write: Messages and/or variable names.

Procedures

The term "procedure" will be used for an independent algorithmic module which solves a particular problem. The use of the word "procedure" or "module" rather than "algorithm" for a given problem is simply a matter of taste. Generally speaking, the word "algorithm" will be reserved for the solution of general problems. The term "procedure" will also be used to describe a certain type of subalgorithm which is discussed in Sec. 2.6.

2.4 CONTROL STRUCTURES

Algorithms and their equivalent computer programs are more easily understood if they mainly use self-contained modules and three types of logic, or flow of control, called

1. Sequence logic, or sequential flow
2. Selection logic, or conditional flow
3. Iteration logic, or repetitive flow

These three types of logic are discussed below, and in each case we show the equivalent flowchart.

Sequence Logic (Sequential Flow)

Sequence logic has already been discussed. Unless instructions are given to the contrary, the modules are executed in the obvious sequence. The sequence may be presented explicitly, by means of numbered steps, or implicitly, by the order in which the modules are written. (See Fig. 2.3.) Most processing, even of complex problems, will generally follow this elementary flow pattern.

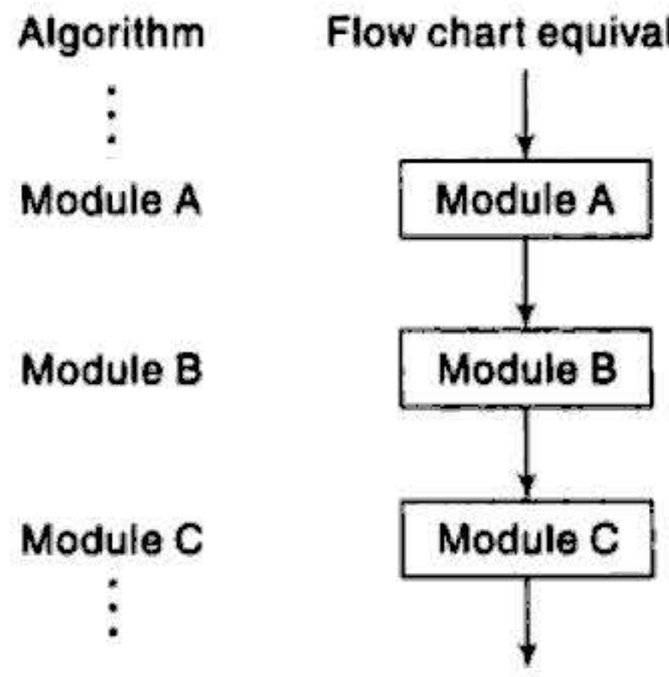


Fig. 2.3 Sequence Logic

Selection Logic (Conditional Flow)

Selection logic employs a number of conditions which lead to a selection of one out of several alternative modules. The structures which implement this logic are called conditional structures or If structures. For clarity, we will frequently indicate the end of such a structure by the statement

[End of If structure.]

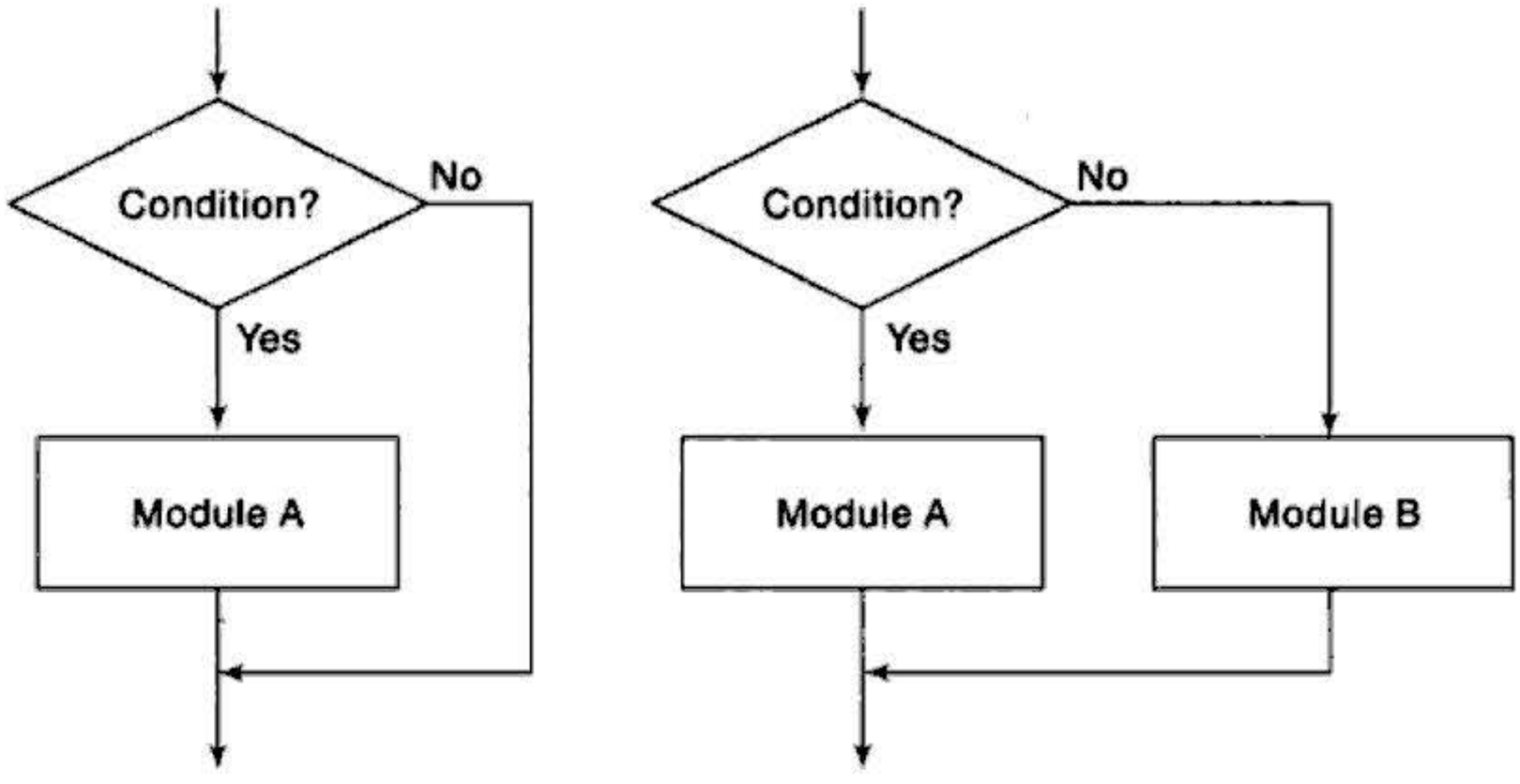
or some equivalent.

These conditional structures fall into three types, which are discussed separately.

1. **Single Alternative.** This structure has the form

If condition, then:
[Module A]
[End of If structure.]

The logic of this structure is pictured in Fig. 2.4(a). If the condition holds, then Module A, which may consist of one or more statements, is executed; otherwise Module A is skipped and control transfers to the next step of the algorithm.



(a) Single alternative. (b) Double alternative.

Fig. 2.4

2. **Double Alternative.** This structure has the form

If condition, then:
[Module A]
Else:
[Module B]
[End of If structure.]

The logic of this structure is pictured in Fig. 2.4(b). As indicated by the flow chart, if the condition holds, then Module A is executed; otherwise Module B is executed.

3. **Multiple Alternatives.** This structure has the form

If condition(1), then:
[Module A_1]
Else if condition(2), then:
[Module A_2]
⋮
Else if condition(M), then:
[Module A_M]
Else:
[Module B]
[End of If structure.]

The logic of this structure allows only one of the modules to be executed. Specifically, either the module which follows the first condition which holds is executed, or the module which follows the final Else statement is executed. In practice, there will rarely be more than three alternatives.

Example 2.5

The solutions of the quadratic equation

$$ax^2 + bx + c = 0$$

where $a \neq 0$, are given by the quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The quantity $D = b^2 - 4ac$ is called the *discriminant* of the equation. If D is negative, then there are no real solutions. If $D = 0$, then there is only one (double) real solution, $x = -b/2a$. If D is positive, the formula gives the two distinct real solutions. The following algorithm finds the solutions of a quadratic equation.

Algorithm 2.2: (Quadratic Equation) This algorithm inputs the coefficients A, B, C of a quadratic equation and outputs the real solutions, if any.

Step 1. Read: A, B, C.
Step 2. Set D: = B^2 - 4AC.
Step 3. If D > 0, then:
(a) Set X1 := (-B + $\sqrt{D}$)/2A and
X2 := (-B - $\sqrt{D}$)/2A.

```
            (b) Write: X1, X2.
        Else if D = 0, then:
            (a) Set X := -B/2A.
            (b) Write: 'UNIQUE SOLUTION', X.
        Else:
            Write: 'NO REAL SOLUTIONS'
        [End of If structure.]
Step 4. Exit.
```

Remark: Observe that there are three mutually exclusive conditions in Step 3 of Algorithm 2.2 that depend on whether D is positive, zero or negative. In such a situation, we may alternatively list the different cases as follows:

Step 3.
1. If D > 0, then:
   .....
2. If D = 0, then:
   .....
3. If D < 0, then:
   .....

Program 2.2

```
/* C implementation of Algorithm 2.2*/
#include <stdio.h>
#include <conio.h>
#include <math.h>

void main()
{
int A,B,C,D;
float X,X1,X2;
clrscr();

printf("Enter the values of A, B and C: ");
scanf("%d %d %d",&A,&B,&C);
D=B*B-4*A*C;

if(D>0)
{
 X1=((-1)*B+sqrt(D))/2*A;
 X2=((-1)*B-sqrt(D))/2*A;
 printf("X1= %.2f, X2= %.2f", X1,X2);
}
else if(D==0)
{
 X=(-1)*B/2*A;
 printf("UNIQUE SOLUTION X=%.2f",X);
}
else
printf("NO REAL SOLUTIONS");
```

```
getch();
}
```

Output:

```
Enter the values of A, B and C: 1
-3
-4
X1= 4.00, X2= -1.00
```

Iteration Logic (Repetitive Flow)

The third kind of logic refers to either of two types of structures involving loops. Each type begins with a Repeat statement and is followed by a module, called the *body of the loop*. For clarity, we will indicate the end of the structure by the statement

[End of loop.]

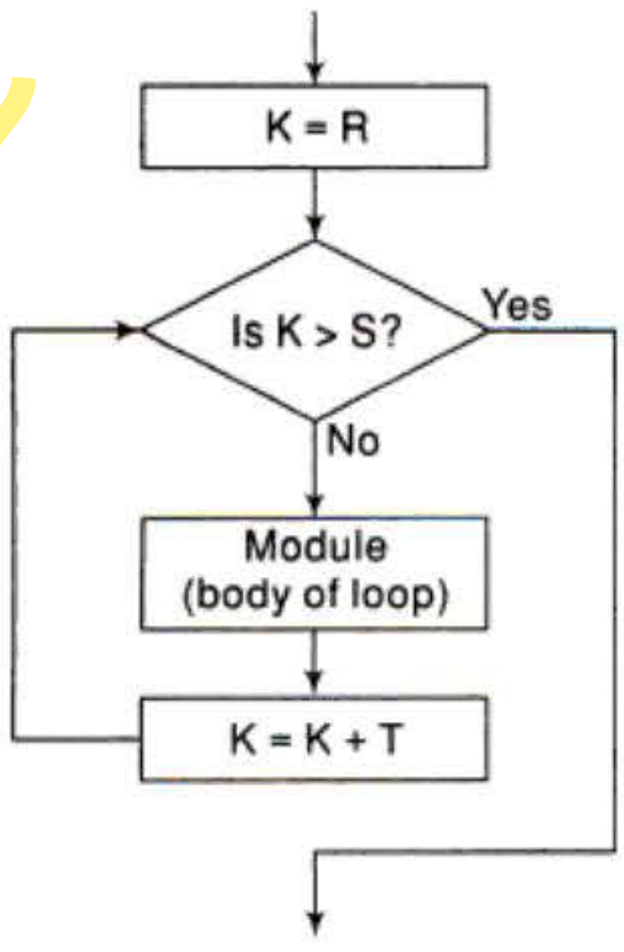
or some equivalent.

Each type of loop structure is discussed separately.

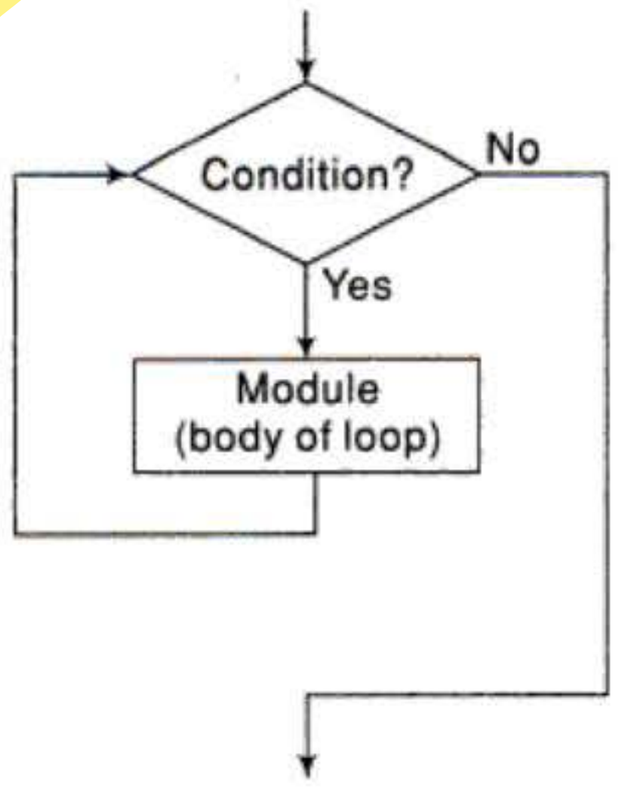
The *repeat-for loop* uses an index variable, such as K, to control the loop. The loop will usually have the form:

Repeat for K = R to S by T:
[Module]
[End of loop.]

The logic of this structure is pictured in Fig. 2.5(a). Here R is called the *initial value*, S the *end value* or *test value*, and T the *increment*. Observe that the body of the loop is executed fitst with K = R, then with K = R + T, then with K = R + 2T, and so on. The cycling ends when K > S. The flow chart assumes that the increment T is positive; if T is negative, so that K decreases in value, then the cycling ends when K < S.



(a) Repeat-For structure.



(a) Repeat-While structure.

Fig. 2.5

The *repeat-while loop* uses a condition to control the loop. The loop will usually have the form

Repeat while condition:
[Module]
[End of loop.]

The logic of this structure is pictured in Fig. 2.5(b). Observe that the cycling continues until the condition is false. We emphasize that there must be a statement before the structure that initializes the condition controlling the loop, and in order that the looping may eventually cease, there must be a statement in the body of the loop that changes the condition.

Example 2.6

Algorithm 2.1 is rewritten using a repeat-while loop rather than a Go to statement:

Algorithm 2.3: (Largest Element in Array) Given a nonempty array DATA with N numerical values, this algorithm finds the location LOC and the value MAX of the largest element of DATA.

1. [Initialize.] Set K := 1, LOC := 1 and MAX := DATA[1].
2. Repeat Steps 3 and 4 while K ≤ N:
3. If MAX < DATA[K], then:
   Set LOC := K and MAX := DATA[K].
   [End of If structure.]
4. Set K := K + 1.
   [End of Step 2 loop.]
5. Write: LOC, MAX.
6. Exit.

Program 2.3

```
/* C implementation of Algorithm 2.3*/
#include <stdio.h>
#include <conio.h>

void main()
{
int  DATA[10]={22,65,1,99,32,17,74,49,33,2};
int N, LOC, MAX, K;
N=10;
K=0;
LOC=0;
MAX=DATA[0];
clrscr();

while(K<N)
{
  if(MAX<DATA[K])
```

```
  {
    LOC=K;
    MAX=DATA[K];
  }
  K=K+1;
}

printf("LOC= %d, MAX= %d",LOC,MAX);
getch();
}

Output:
LOC = 3, MAX= 99
```

Algorithm 2.3 indicates some other properties of our algorithms. Usually we will omit the word "Step." We will try to use repeat structures instead of Go to statements. The repeat statement may explicitly indicate the steps that form the body of the loop. The "End of loop" statement may explicitly indicate the step where the loop begins. The modules contained in our logic structures will normally be indented for easier reading. This conforms to the usual format in structured programming.

Any other new notation or convention either will be self-explanatory or will be explained when it occurs.