CHAPTER 7

Composition and Decomposition

Learning Objectives

After learning the concepts in this chapter, the students will be able

- To know the notations used in algorithmic techniques.
- To understand Composition and Decomposition in algorithmic techniques.

In Chapter 1, we saw that algorithms are composed of statements. Statements can be grouped into compound statements, giving rise to a hierarchical structure of algorithms. Decomposition is one of the elementary problem-solving techniques.An algorithm may be broken into parts, expressing only high level details. Then, each part may be refined into smaller parts, expressing finer details, or each part may be abstracted as a function.

7.1 Notations for Algorithms

We need a notation to represent algorithms. There are mainly three different notations for representing algorithms.

- A programming language is a notation for expressing algorithms to be executed by computers.
- Pseudo code is a notation similar to programming languages. Algorithms expressed in pseudo code are not intended to be executed by computers, but for communication among people.
- Flowchart is a diagrammatic notation for representing algorithms. They give a visual intuition of the flow of control, when the algorithm is executed.

7.1.1 Programming language

A programming language is a notation for expressing algorithms so that a computer can execute the algorithm. An algorithm expressed in a programming language is called a program. C, C++ and Python are examples of programming languages. Programming language is formal. Programs must obey the grammar of the programming language exactly. Even punctuation symbols must be exact. They do not allow the informal style of natural languages such as English or Tamil. There is a translator which translates the program into instructions executable by the computer. If our program has grammatical errors, this translator will not be able to do the translation.

BVDDQ3

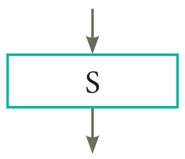
7.1.2 Pseudo-code

Pseudo code is a mix of programming-language-like constructs and plain English. This notation is not formal nor exact. It uses the same building blocks as programs, such as variables and control flow. But, it allows the use of natural English for statements and conditions. An algorithm expressed as pseudo code is not for computers to execute directly, but for human readers to understand. Therefore, there is no need to follow the rules of the grammar of a programming language. However, even pseudo code must be rigorous and correct. Pseudo code is the most widely used notation to represent algorithms.

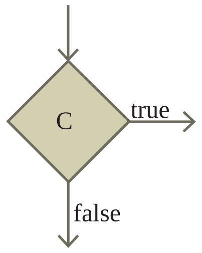
7.1.3 Flowcharts

Flowchart is a diagrammatic notation for representing algorithms. They show the control flow of algorithms using diagrams in a visual manner. In flowcharts, rectangular boxes represent simple statements, diamond-shaped boxes represent conditions, and arrows describe how the control flows during the execution of the algorithm. A flowchart is a collection of boxes containing statements and conditions which are connected by arrows showing the order in which the boxes are to be executed.

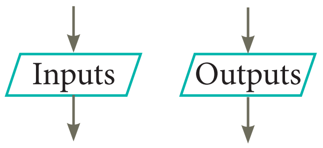
1. A statement is contained in a rectangular box with a single outgoing arrow,which points to the box to be executed next.



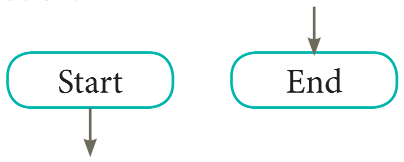
2. A condition is contained in a diamond-shaped box with two outgoing arrows, labeled true and false. The true arrow points to the box to be executed next if the condition is true, and the false arrow points to the box to be executed next if the condition is false.



3. Parallelogram boxes represent inputs given and outputs produced.



4. Special boxes marked Start and the End are used to indicate the start and the end of an execution:



The flowchart of an algorithm to compute the quotient and remainder after dividing an integer A by another integer B is shown in Figure 7.1, illustrating the different boxes such as input, output, condition, and assignment, and the control flow between the boxes. The algorithm is explained in Example 7.4.

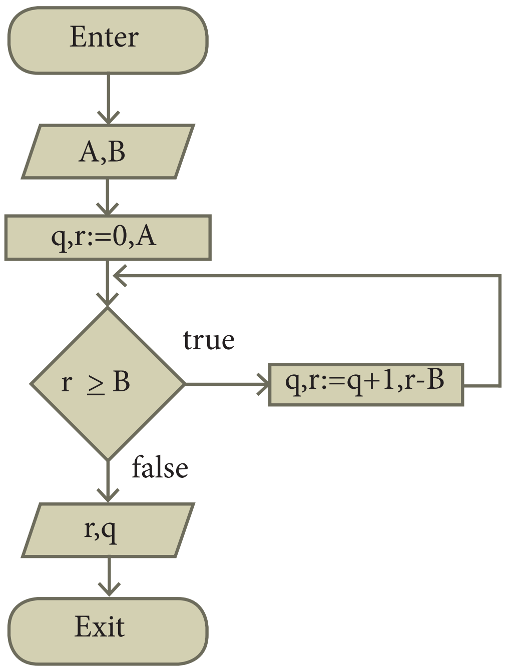


Figure 7.1: Flowchart for integer division

Flowcharts also have disadvantages. (1) Flowcharts are less compact than representation of algorithms in programming language or pseudo code. (2) They obscure the basic hierarchical structure of the algorithms. (3) Alternative statements and loops are disciplined control flow structures. Flowcharts do not restrict us to disciplined control flow structures.

7.2 Composition

A statement is a phrase that commands the computer to do an action. We have already seen assignment statement. It is a simple statement, used to change the values of variables. Statements may be composed of other statements, leading to hierarchical structure of algorithms. Statements composed of other statements are known as compound statements.

Control flow statements are compound statements. They are used to alter the control flow of the process depending on the state of the process. There are three important control flow statements:

- Sequential
- Alternative
- Iterative

When a control flow statement is executed, the state of the process is tested, and depending on the result, a statement is selected for execution.

7.2.1 Sequential statement

A sequential statement is composed of a sequence of statements. The statements in the sequence are executed one after another, in the same order as they are written in the algorithm, and the control flow is said to be sequential. Let S1 and S2 be statements. A sequential statement composed of S1 and S2 is written as

```
S1
S2
```

In order to execute the sequential statement, first do S1 and then do S2.

The sequential statement given above can be represented in a flowchart as shown in in Figure 7.2. The arrow from S1 to S2 indicates that S1 is executed, and after that, S2 is executed.

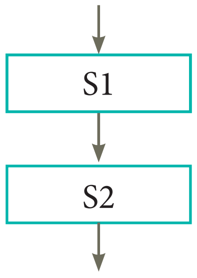


Figure 7.2: Sequential control flow

Let the input property be P, and the input-output relation be Q, for a problem. If statement S solves the problem, it is written as

```
1.  -- P
2.  S
3.  -- Q
```

If we decompose the problem into two components, we need to compose S as a sequence of two statements S1 and S2 such that the input-output relation of S1, say R, is the input property of S2.

```
1.  -- P
2.  S1
3.  -- R
4.  S2
5.  -- Q
```

Example 7.1. Let us solve the Farmer, Goat, Grass, and Wolf problem of Example 6.12. We decided to represent the state of the process by four variables farmer, goat, grass, and wolf, representing the sides of the farmer, goat, grass and wolf, respectively. In the initial state, all four variables have the value L (Left side). In the final state, all four variables should have the value R (Right side). The goal is to construct a statement S so as to move from the initial state to the final state.

```
1. -- farmer, goat, grass, wolf = L, L, L, L
2. S
3. -- farmer , goat , grass , wolf = R, R, R, R
```

We have to compose S as a sequence of assignment statements such that in none of the intermediate states

1. goat and wolf have the same value but farmer has the opposite value, or
2. goat and grass have the same value but farmer has the opposite value.Subject to these constraints, a sequence of assignments and the state after each assignment are shown in Figure 7.3.

```
1.  -- farmer, goat, grass, wolf = L, L, L, L
2.  farmer, goat := R, R
3.  -- farmer , goat , grass , wolf = R, R, L, L
4.  farmer := L
5.   farmer, goat, grass, wolf = L, R, L, L
6.  farmer, grass := R, R
7.  -- farmer , goat , grass , wolf = R, R, R, L
8.  farmer, goat := L, L
9.  -- farmer, goat, grass, wolf = L, L, R, L
10. farmer, wolf := R, R
11. -- farmer , goat , grass , wolf = R, L, R, R
12. farmer : = L
13. -- farmer , goat , grass , wolf = L, L, R, R
14. farmer , goat : = R, R
15. -- farmer , goat , grass , wolf = R, R, R, R
```

Figure 7.3: Sequence of assignments for goat, grass and wolf problem

Other than lines (1) and (15), in line (7), goat and grass have the same value, but farmer also has the same value as they. In line (9), goat and wolf have the same value, but farmer also has the same value as they. Thus, the sequence has achieved the goal state, without violating the constraints.

7.2.2 Alternative statement

A condition is a phrase that describes a test of the state. If C is a condition and both

S1 and S2 are statements, then

```
if  C
    S1
else
    S2
```

is a statement, called an alternative statement, that describes the following action:

1. Test whether C is true or false.
2. If C is true, then do S1; otherwise do S2.

In pseudo code, the two alternatives S1 and S2 are indicated by indenting them from the keywords if and else, respectively. Alternative control flow is depicted in the flowchart of Figure 2.4. Condition C has two outgoing arrows, labeled true and false. The true arrow points to the S1 box. The false arrow points to the S2 box. Out going arrows of S1 and S2 point to the same box, the box after the alternative statement.

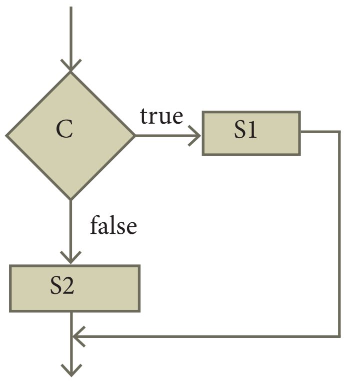


Figure 7.4: Alternative control flow

Conditional statement: Sometimes we need to execute a statement only if a condition is true and do nothing if the condition is false. This is equivalent to the alternative statement in which the else-clause is empty. This variant of alternative statement is called a conditional statement. If C is a condition and S is a statement, then

```
if   C
   S
```

is a statement, called a conditional statement, that describes the following action:

1. Test whether C is true or false.
2. If C is true then do S; otherwise do nothing.

The conditional control flow is depicted in the flowchart of Figure 2.5.

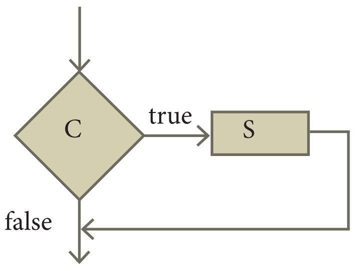


Figure 7.5: Conditional control flow

Example 7.2. Minimum of two numbers: Given two numbers a and b, we want to find the minimum of the two using the alternative statement. Let us store the minimum in a variable named result. Let a ↓ b denote the minimum of a and b (for instance, 4 ↓ 2 = 2, —5 ↓ 6 = -5). Then, the specification of algorithm minimum is

```
minimum(a, b)
-- input s : a , b
-- outputs: result = a ↓ b
```

Algorithm minimum can be defined as

```
1.   minimum(a, b)
2.     -- a, b
3.     if a < b
4.        result : = a
5.     else
6.        result = b
7.     --  result = a ↓ b
```

7.2.3 Case analysis

Alternative statement analyses the problem into two cases. Case analysis statement generalizes it to multiple cases. Case analysis splits the problem into an exhaustive set of disjoint cases. For each case, the problem is solved independently. If C1, C2, and C3 are conditions, and S1, S2, S3 and S4 are statements, a 4-case analysis statement has the form,

```
1. case  C1
2.    S1
3. case  C2
4.    S2
5. case  C3
6.    S3
7. else
8.    S4
```

The conditions C1, C2, and C3 are evaluated in turn. For the first condition that evaluates to true, the corresponding statement is executed, and the case analysis statement ends. If none of the conditions evaluates to true, then the default case S4 is executed.

1. The cases are exhaustive: at least one of the cases is true. If all conditions are false, the default case is true.
2. The cases are disjoint: only one of the cases is true. Though it is possible for

more than one condition to be true, the case analysis always executes only one case, the first one that is true. If the three conditions are disjoint, then the four cases are (1) C1, (2) C2, (3) C3, (4) (not C1) and (not C2) and (not C3).

Example 7.3. We want an algorithm that compares two numbers and produces the result as

$$\text{compare}(a, b) = \begin{cases} 1- & \text{if } a < b \\ 0 & \text{if } a = b \\ 1 & \text{if } a > b \end{cases}$$

We can split the state into an exhaustive set of 3 disjoint cases: a < b, a = b, and a> b. Then we can define compare() using a case analysis.

```
1. compare(a, b)
2.    case a < b
3.       result := -1
4.    case a = b
5.       result := 0
6.    else -- a > b
7.       result : = 1
```

7.2.4 Iterative statement

An iterative process executes the same action repeatedly, subject to a condition C. If C is a condition and S is a statement, then

```
while C
   S
```

is a statement, called an iterative statement, that describes the following action:

1. Test whether C is true or false.
2. If C is true, then do S and go back to step 1; otherwise do nothing.

The iterative statement is commonly known as a loop. These two steps, testing C and executing S, are repeated until C becomes false. When C becomes false, the loop ends, and the control flows to the statement next to the iterative statement. The condition C and the statement S are called the loop condition and the loop body, respectively. Testing the loop condition and executing the loop body once is called an iteration. not C is known as the termination condition.

Iterative control flow is depicted in the flowchart of Figure 7.6. Condition C has two outgoing arrows, true and false. The true arrow points to S box. If C is true, S box is executed and control flows back to C box. The false arrow points to the box after the iterative statement (dotted box). If C is false, the loop ends and the control flows to the next box after the loop.

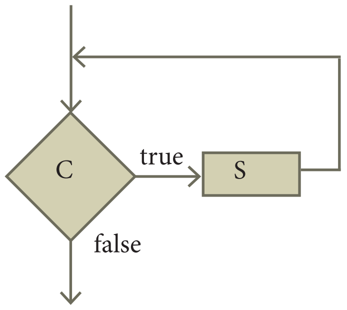


Figure 7.6: Iterative control flow

Example 7.4. Construct an iterative algorithm to compute the quotient and remainder after dividing an integer A by another integer B.

We formulated the specification of the algorithm in Example 6.6 as

```
divide (A , B)
-- inputs:    A is an integer and B ≠ 0
-- outputs :  q and r such that A = q X B
      + r and
--            0 ≤ r < B
```

Now we can construct an iterative algorithm that satisfies the specification.

```
divide (A , B)
-- inputs: A is an integer and B ≠ 0
-- outputs : q and r such that A = q X B + r and
--          0 < r < B
    q, r : = 0, A
    while r ≥ B
        q, r := q + 1, r - B
```

The algorithm is presented as a flowchart in Figure 7.1.

We can execute the algorithm step-by-step for a test input, say, (A, B) = (22, 5). Each row of Table 7.1 shows one iteration — the evaluation of the expressions and the values of the variables at the end of an iteration. Note that the evaluation of the expression uses the values of the variables from the previous row. Output variables q and r change their values in each iteration. Input variables A and B do not change their values. Iteration 0 shows the values just before the loop starts. At the end of iteration 4, condition (r ≥ B) = (2 ≥ 5) is false, and hence the loop ends with (q, r) = (4, 2).

iteration	q	q+1	r	r-B	A	B
0	0		22		22	5
1	1	0+1	17	22-5		
2	2	1 + 1	12	17-5		
3	3	2+1	7	12-5		
4	4	3+1	2	7-5		

Table 7.1: Step by step execution of divide (22, 5)

Example 7.5. In the Chameleons of Chromeland problem of Example 1.3, suppose two types of chameleons are equal in number. Construct an algorithm that arranges meetings between these two types so that they change their color to the third type. In the end, all should display the same color.

Let us represent the number of chameleons of each type by variables a, b and c, and their initial values by A, B and C, respectively. Let a = b be the input property. The input-output relation is a = b = 0 and c = A+B+C. Let us name the algorithm monochromatize. The algorithm can be specified as

```
monochromatize(a, b, c)
-- inputs: a=A, b=B, c=C, a=b
-- outputs : a = b = 0 , c = A+B+C
```

In each iterative step, two chameleons of the two types (equal in number) meet and change their colors to the third one. For example, if A, B, C = 4, 4, 6, then the series of meetings will result in

iteration	a	b	c
0	4	4	6
1	3	3	8
2	2	2	10
3	1	1	12
4	0	0	14

Table 7.2: Series of meetings between two types of chameleons equal in number.

In each meeting, a and b each decreases by 1, and c increases by 2. The solution can be expressed as an iterative algorithm.

```
monochromatize(a, b, c)
    -- inputs: a=A, b=B, c=C, a=b
    -- outputs: a = b = 0, c = A+B+C
        while a > 0
            a, b, c := a-1, b-1, c+2
```

The algorithm is depicted in the flowchart of Figure 7.7.

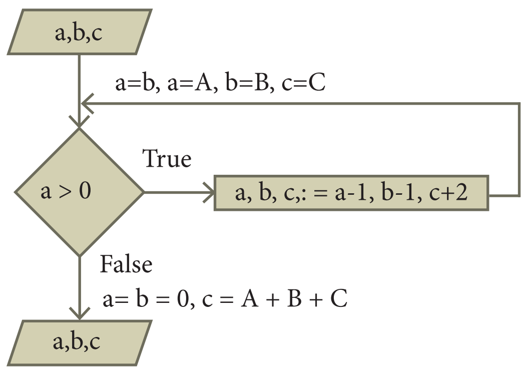


Figure 7.7: Algorithm monochromatize

7.3 Decomposition

Problem decomposition is one of the elementary problem-solving techniques. It involves breaking down a problem into smaller and more manageable problems, and combining the solutions of the smaller problems to solve the original problem. Often, problems have structure. We can exploit the structure of the problem and break it into smaller problems. Then, the smaller problems can be further broken until they become sufficiently small to be solved by other simpler means. Their solutions are then combined together to construct a solution to the original problem.

7.3.1 Refinement

After decomposing a problem into smaller subproblems, the next step is either to refine the subproblem or to abstract the subproblem.

1. Each subproblem can be expanded into more detailed steps. Each step can be further expanded to still finer steps, and so on. This is known as refinement.
2. We can also abstract the subproblem. We specify each subproblem by its input property and the input-output relation. While solving the main problem, we only need to know the specification of the subproblems. We do not need to know how the subproblems are solved.

Example 7.6. Consider a school goer's action in the morning. The action can be written as

1 Get ready for school

We can decompose this action into smaller, more manageable action steps which she takes in sequence:

1 Eat breakfast
2 Put on clothes
3 Leave home

We have refined one action into a detailed sequence of actions. However, each of these actions can be expanded into a sequence of actions at a more detailed level, and this expansion can be repeated. The action "Eat breakfast" can be expanded as

1 -- Eat breakfast
2 Eat idlis
3 Eat eggs
4 Eat bananas

The action "Put on clothes" can be expanded as

1 -- Put on clothes
2 Put on blue dress
3 Put on socks and shoes
4 Wear ID card

and "Leave home" expanded as

1 -- Leave home
2 Take the bicycle out
3 Ride the bicycle away

Thus, the entire action of "Get ready for school" has been refined as

1 -- Eat breakfast
2 Eat idlis
3 Eat eggs
4 Eat bananas

```

-- Put on clothes
Put on blue dress
Put on socks and shoes
Wear ID card

-- Leave home
Take the bicycle out
Ride the bicycle away
```

Refinement is not always a sequence of actions. What the student does may depend upon the environment. How she eats breakfast depends upon how hungry she is and what is on the table; what clothes she puts on depends upon the day of the week. We can refine the behaviour which depends on environment, using conditional and iterative statements.

```
-- Eat breakfast
if hungry and idlis on the table
    Eat idlis
if hungry and eggs on the table
    Eat eggs
if hungry and bananas on the table
    Eat bananas

-- Put on clothes
if Wednesday
    Put on blue dress
else
    Put on white dress
Put on socks and shoes
Wear the ID card

-- Leave home
Take the bicycle out
Ride the bicycle away
```

The action "Eat idlis" can be further refined as an iterative action:

```
-- Eat idlis
Put idlis on the plate
Add chutney
while idlis in plate
    Eat a bite of idli
```

How "Get ready for school" is refined in successive levels is illustrated in Figure 2.8.

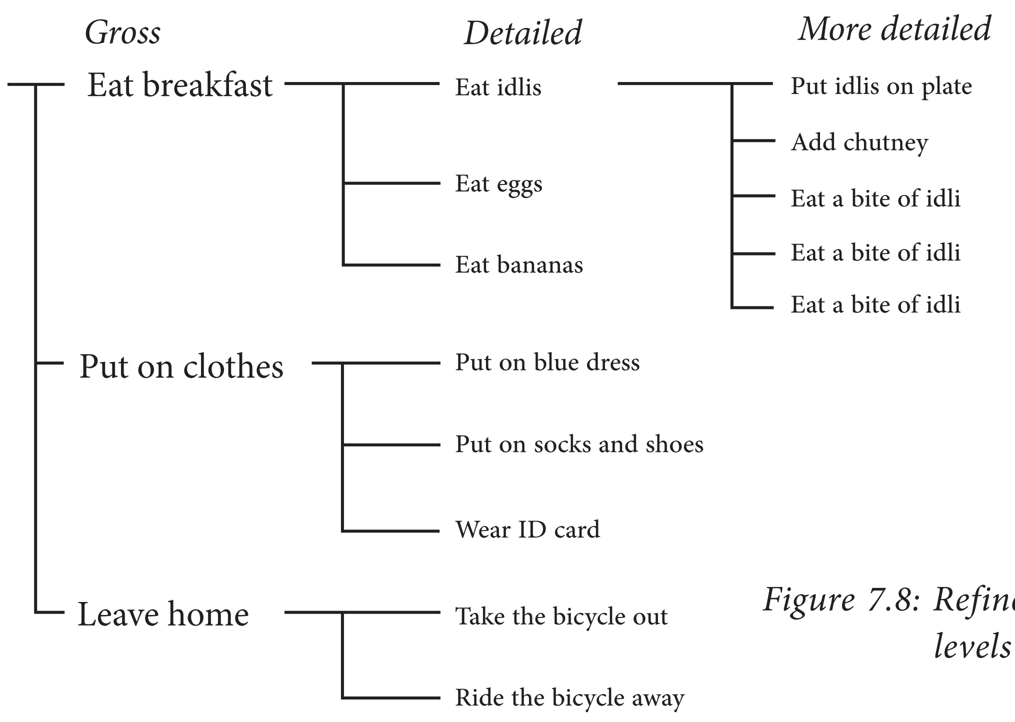


Figure 7.8: Refinement at various levels of details

The action "Eat breakfast" is depicted in a flowchart shown in Figure 2.9.

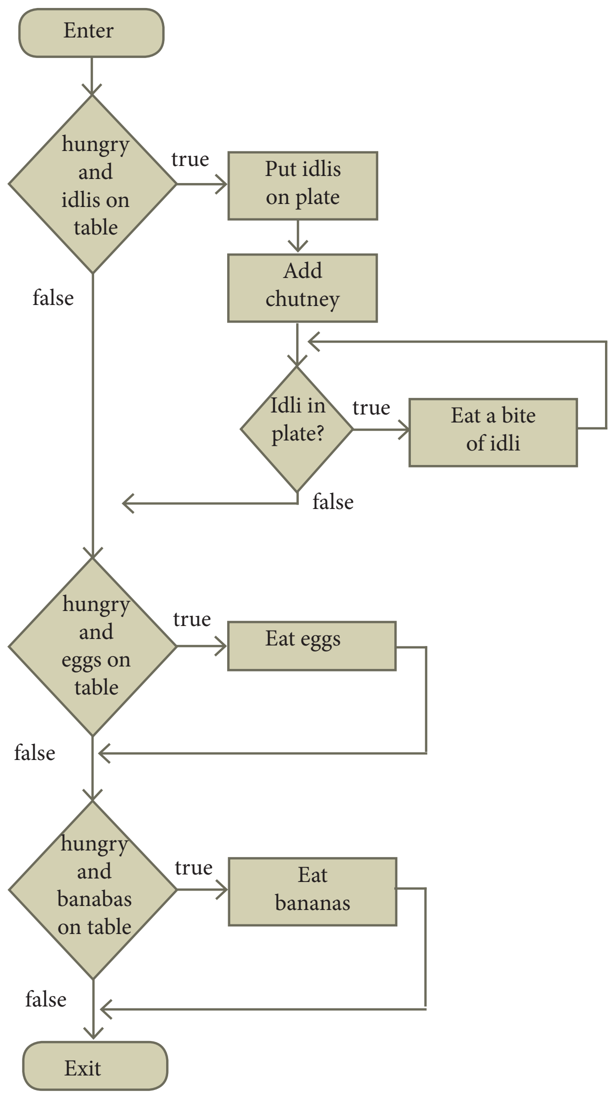


Figure 7.9: Flowchart for Eat breakfast

Note that the flowchart does not show the hierarchical structure of refinement.

7.3.2 Functions

After an algorithmic problem is decomposed into subproblems, we can abstract the subproblems as functions. A function is like a sub-algorithm. Similar to an algorithm, a function is specified by the input property, and the desired input-output relation.

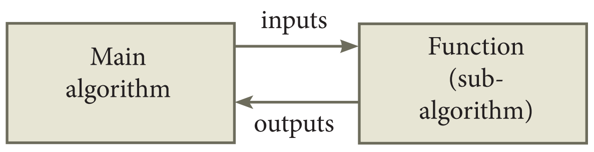


Figure 7.10: Function definition

To use a function in the main algorithm, the user need to know only the specification of the function — the function name, the input property, and the input-output relation. The user must ensure that the inputs passed to the function will satisfy the specified property and can assume that the outputs from the function satisfy the input-output relation. Thus, users of the function need only to know what the function does, and not how it is done by the function. The function can be used a a "black box" in solving other problems.

Ultimately, someone implements the function using an algorithm. However, users of the function need not know about the algorithm used to implement the function. It is hidden from the users. There is no need for the users to know how the function is implemented in order to use it.

An algorithm used to implement a function may maintain its own variables. These variables are local to the function in the sense that they are not visible to the user of the function. Consequently, the user has fewer variables to maintain in the main algorithm, reducing the clutter of the main algorithm.

Example 7.7. Consider the problem of testing whether a triangle is right-angled, given its three sides a, b, c, where c is the longest side. The triangle is right-angled, if

$$c^2 = a^2 + b^2$$

We can identify a subproblem of squaring a number. Suppose we have a function square(), specified as

```
square(y)
-- inputs : y
-- outputs : y²
```

we can use this function three times to test whether a triangle is right-angled. square() is a "black box" — we need not know how the function computes the square. We only need to know its specification.

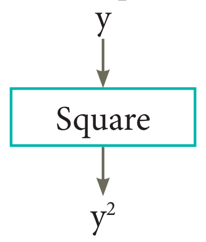


Figure 7.11: square function

```
right_angled(a, b, c)
-- inputs: $c \geq a, c \geq b$
-- outputs: result = true if $c^2 = a^2 + b^2$;
--              result = false , otherwise
if square (c) = square (a) + square (b)
   result := true
else
   result := false
```

Points to Remember

- Compound statements are composed of sequential, alternative and iterative control flow statements.
- The value of a condition is true or false, depending on the values of the variables.
- Alternative statement selects and executes exactly one of the two statements,depending on the value of the condition.
- Conditional statement is executed only if the condition is true. Otherwise, nothing is done.
- Iterative statement repeatedly evaluates a condition and executes a statement as long as the condition is true.
- Programming language, pseudo code, and flowchart are notations for expressing algorithms.
- Decomposition breaks down a problem into smaller subproblems and combine their solutions to solve the original problem.
- A function is an abstraction of a subproblem, and specified by its input property, and its input-output relation.
- Users of function need to know only what the function does, and not how it is done.
- In refinement, starting from high level, each statement is repeatedly expanded into more detailed statements in the subsequent levels.

Evaluation

SECTION – A

Choose the correct answer

1. Suppose u, v = 10 ,5 before the assignment. What are the values of u and v after the sequence of assignments?

   1 u := v

   2 v := u

(a) u, v = 5 ,5 (c) u, v = 10 ,5

(b) u, v = 5 ,10 (d) u, v = 10 ,10

2. Which of the following properties is true after the assignment (at line 3?

```
1 --i, j = 0, 0
2 i, j := i+1, j-1
3 -- ?
```

(a) i+j >0 (b) i+j < 0 (c) i+j =0 (d) i = j

3. If C1 is false and C2 is true, the compound statement

```
1 if C1
2     S1
3 else
4 if C2
5     S2
6 else
7     S3
```

executes

(a) S1 (b) S2 (c) S3 (d) none

C4D5J8

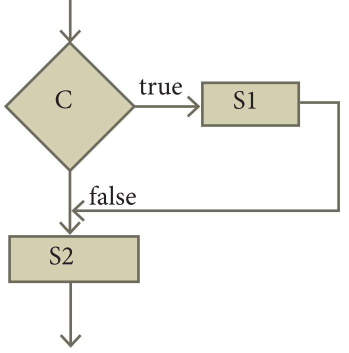
4. If C is false just before the loop, the control flows through

```
1 S1
2 while C
3 S2
4 S3
```

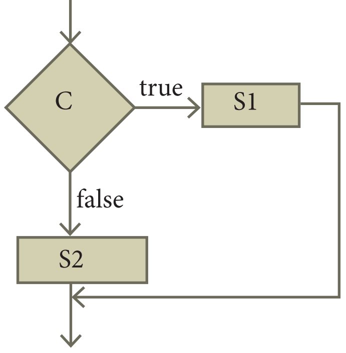
(a) S1 ; S3 (b) S1 ; S2 ; S3

(c)S1 ; S2 ; S2 ; S3 (d) S1 ; S2 ; S2 ; S2 ; S3

5. If C is true, S1 is executed in both the flowcharts, but S2 is executed in



(1)



(2)

(a) (1) only (b) (2) only

(c) both (1) and (2) (d) neither (1) nor (2)

6. How many times the loop is iterated?

```
i := 0
while i ≠ 5
  i := i + 1
```

(a) 4 (b) 5 (c) 6 (d) 0

SECTION-B

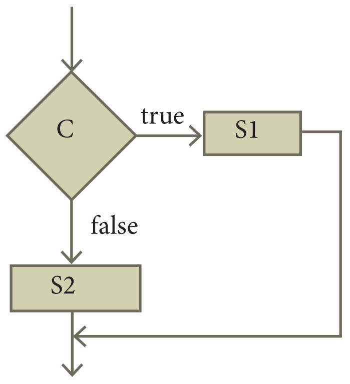
Very Short Answers

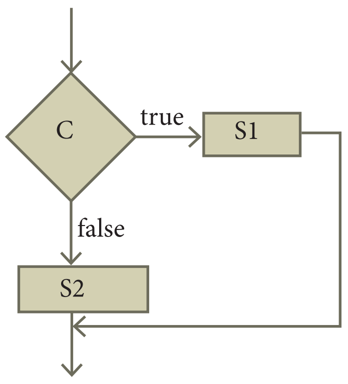
1. Distinguish between a condition and a statement.
2. Draw a flowchart for conditional statement.
3. Both conditional statement and iterative statement have a condition and a statement. How do they differ?
4. What is the difference between an algorithm and a program?
5. Why is function an abstraction?
6. How do we refine a statement?

SECTION-C

Short Answers

1. For the given two flowcharts write the pseudo code.





2. If C is false in line 2, trace the control flow in this algorithm.

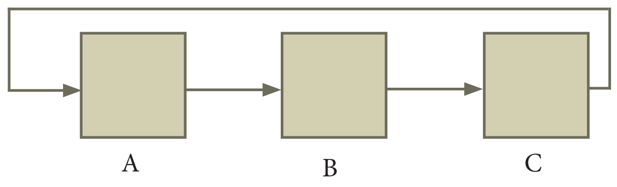
```
1  S1
2  -- C is false
3  if C
4  S2
5  else
6  S3
7  S4
```

3. What is case analysis?
4. Draw a flowchart for -3case analysis using alternative statements.
5. Define a function to double a number in two different ways: (1) n + n, (2) 2 x n

SECTION - D

Explain in detail

1. Exchange the contents: Given two glasses marked A and B. Glass A is full of apple drink and glass B is full of grape drink. Write the specification for exchanging the contents of glasses A and B, and write a sequence of assignments to satisfy the specification.
2. Circulate the contents: Write the specification and construct an algorithm to circulate the contents of the variables A, B and C as shown below: The arrows indicate that B gets the value of A, C gets the value of B and A gets the value of C.



3. Decanting problem. You are given three bottles of capacities 5 ,8, and 3 litres. The 8L bottle is filled with oil, while the other two are empty. Divide the oil in 8L bottle into two equal quantities. Represent the state of the process by appropriate variables. What are the initial and final states of the process? Model the decanting of oil from one bottle to another by assignment. Write a sequence of assignments to achieve the final state.
4. Trace the step-by-step execution of the algorithm for factorial(4).

```
factorial(n)
-- inputs : n is an integer , n ≥ 0
-- outputs : f = n!
   f, i := 1 ,1
   while i ≤ n
     f, i := f × i, i+1
```