TOPIC 3 : ALGORITHM DESIGN FOR SEQUENCE CONTROL STRUCTURE

Data Type, Data & Information

- Data are raw materials entered into a computer to be processed in order to produce meaningful information.

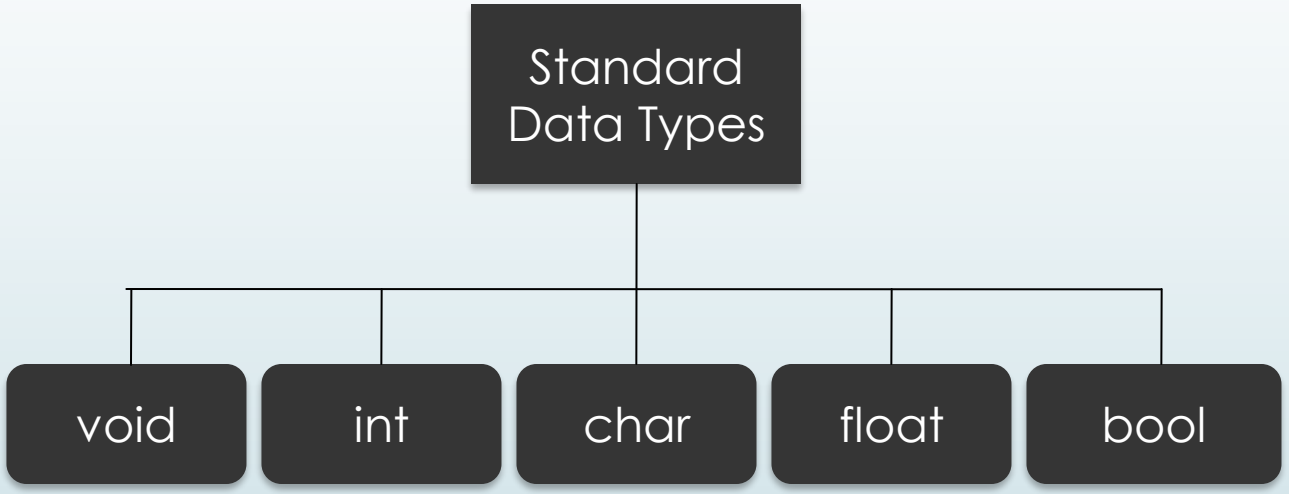
Data is a collection of unprocessed items, which can include text, numbers, images, audio, and video.

Information conveys meaning and is useful to people.

Data type

- Textual data type consists of number or numeric, letter (small or capital letter) and special symbols or special characters.
- Textual data can be further divided into two types:
  - **Simple or primitive data type**
    - Data that has a single value and cannot be further broken into small values.
  - **Complex or composite data type**
    - Can be broken down further to a few primitive data. It means that a complex or composite data are actually a collection of primitive data.

Standard data types



… Standard data types

Data type	Description
void	▪ The *void* type has no values and no operations. ▪ In other words, both the set of values and the set of operations are empty.
Integer (int)	▪ An *integer* type is a number without a fraction part. ▪ It is also known as an integral number. ▪ C++ supports three different sizes of the integer data type: **short int, int** and **long int** (defines these data types so that they can be organized from the smallest to the largest)
Character (char)	▪ A character is any value that can be represented in the computer's alphabet. ▪ Most computers use ASCII alphabet. ▪ Most of the personal, mini-, and mainframe computers use one byte to store the *char* data types. (Note: A byte is 8 bits. With 8 bits, there are 256 different values in the *char* set)
float	▪ A *floating-point* type is a number with a fractional part.

… Standard data types

Data type	Description
float	▪ A *floating-point* type is a number with a fractional part. ▪ The C++ language support three different sizes of floating-point data types: **float, double** and **long double** (so that can be organized from smallest to largest). ▪ Although the physical size of floating-point type is machine dependent, many computer support the sizes **float (4 bytes), double (8 bytes)** and **long double (10 bytes)**.
Boolean (bool)	▪ *Logical* or *Boolean* data consists of only two values: **true** and **false**. ▪ True (1) and false (0).

A final point to remember about all the data types that have been discussed is that each type has its own internal format. Therefore, when the same number value is stored in different types, the internal bit configuration will be different. For example: the ASCII character plus (+) has a value of 43, but its bit configuration is different from the integer 43, which is also different from the float 43.0.

*** One of the your jobs as a programmer is to use these different types **consistently** in your programs.

Operators and identifiers

Identifier

- One feature present in all computer languages is the **identifier** – allow us to name data and other objects in the program.
- **Identifier** is used to define names to represent **variables**, **constant** and name of **function**.
- Each piece of data in the computer is stored at a unique address.
- If we didn't have identifiers that we could use to symbolically represent data locations, we would have to know and use data addresses to manipulate them.
- Instead, we simply give data identifier names and let the compiler keep track of where they are physically located.
- Different programming language use different rules to form identifier.
- In C++, consists of letters (capital letter A through Z, the lowercase letter a to z), digits (0 through 9), and the underscore (no other symbols are permitted to form a identifier.
- Besides, the 1st character cannot be a digit - must be begin with a letter or underscore.
- Good identifier names are **descriptive** (by combining two or more words) but **short**.

Variables

- **Variables** are memory locations, whose contents can vary or differ over time.
- Names that refer to memory locations, that can holds values
- It is the ability of memory variable to change in value that makes computers and programming worthwhile.
- Because one memory location can be used over and over again with different values, you can write program instructions once and then use them for thousands of separate calculations.
- Every computer programming language has its own set of rules for naming variables.

- Most language allow both **letters** and **digits** within variable names.
- Different languages put different limits on the **length** of variable names, although in general, newer languages allow longer names.
- Many modern languages, such as C++, C# and Java allow more than 200 characters in a variable name. Variable names in these languages usually consist of **lowercase letters**, **don't allow hyphens**, but do **allow underscores**.
- Naming variables is **case sensitive**, so HOURLYWAGE, hourlywage, and hourlyWage are considered three separate variable names.
- Most programmers who use the more modern language employ the format in which multiple-word variable names are run together, and each new word within the variable name begins with an uppercase letter (**camel casing**).

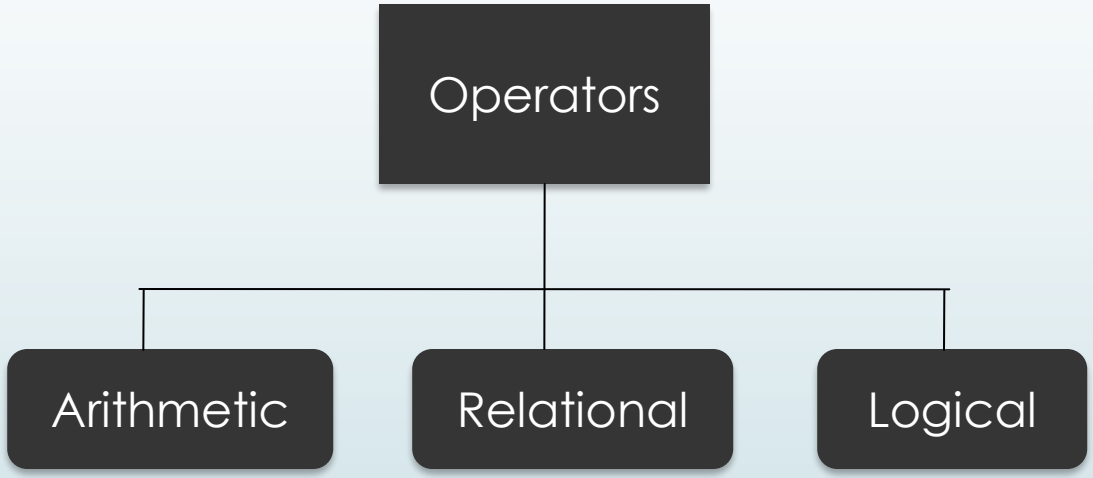
Constant

- **Constant** are memory location, whose content is not allowed to change during program execution.
- Holds data that **remains** the **same** as the program runs.
- Allow us to give a name to a value that is used several times in a program.
- Once the variable is declared constant, the **value cannot be changed** and the variable cannot be assigned to another value.

Rules of naming identifier, variable and constant

- Even though every language has its own rules for naming variables, when designing the logic of a computer program, you should not concern yourself with the specific syntax of any particular computer language
  - The logic, after all, works with any language.
- The names follow only two rules:
  1. *Names must be one word.*
     - The name can contain letters, digits, underscores, or other with the exception of *space*.
  2. *Names should have some appropriate meaning*.
     - This is not a rule of any programming language.
     - As long as the correct numeric result is placed in the variable, its actual name doesn't really matter.
     - However, it's much easier to follow the logic of a program if you use appropriate meaning of variable name.
     - You might think that you will remember how you intended to use a cryptic variable name within a program, but six years later when a program requires changes, you and other programmers working with you, will appreciate clear as well as descriptive variable names.

Operators



Arithmetic operator

- One of the most important uses of a computer is its ability to calculate.
- To perform arithmetic operations (manipulate integral and floating-point data types).
- The arithmetic operators:

Operator	Operation	
+	Addition	Integral data type
-	Subtraction	
*	Multiplication	
/	Division	
%	Modulus (Remainder)	Floating-point data type
--	Decrement by 1	Integral data type
++	Increment by 1	

Relational operator

- To compare the values of two operands.
- The evaluation result is either **TRUE (1)** or **FALSE (0).**
- The relational operators:

Operator	Operation
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Logical operator

- When there is more than one relational expression at a time, logical operator are used to perform the evaluation.
- The evaluation result is either **TRUE (1)** or **FALSE (0).**
- The logical operators:

Operator	Operation
&&	AND
||	OR
!	NOT

OPERATOR PRECEDENCE

- When more than one arithmetic operator is used in an expression, C++ uses the operator precedence rules to evaluate the expression.

Operator Category	Operator (evaluated from left to right)
Parentheses	( )
Multiply, Division, Modulus	* / %
Add, Subtract	+ -
Relational Operators	< <= > >=
Equality Operators	== !=
Logical AND	&&
Logical OR	||

Highest

Lowest

The multiplication, division, and modulus operators are evaluated before addition and subtraction operators. Operators having the same level of precedence are evaluated from left to right. Grouping is allowed for clarity.

Assignment Statements

Assignment Statements

- A statement is a step in algorithm.
- It is an instructions that tells the computer what to do.
  - Assignment statement

Assignment Statement

- **Purpose**: To assign data or value to a variable.
- It directly assigned a value to the variable or the result of a calculation in the program.
- We can give value to the variable that stores a numeric value using three ways:
  i. Assign a fixed value to a variable
  ii. Copy the value of a variable to another variable
  iii. A result of a calculation using a formula

Assignment Statement (cont.)

- Words that represent assignment statement: ***set*** or ***initialize***
- Symbols that represent assignment statement: **=** or
- For example:

  i. ***Initialize count to zero***
  ii. ***count 0***
  iii. ***count = 0***
  iv. ***Set count to zero***
  v. ***Set grade with 'A'***
  vi. ***Set name to "Hawa Adam"***

Assignment Statement (cont.)

- We can copy the value of a variable to another variable by writing these statements:

  i. ***Set number2 with number1***

  ii. ***number2 ← number1***

  It will cause the content at variable *number1* to be copied to location *number2*

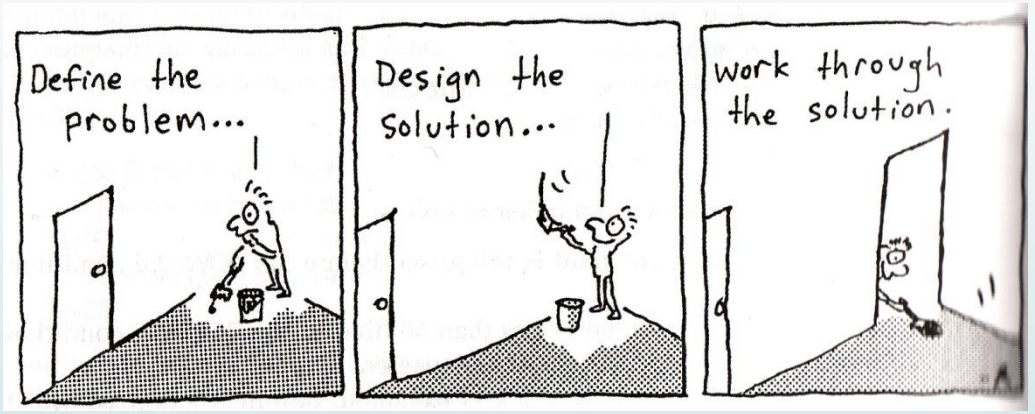
  i. ***number1 = number2***

  It will cause the content at variable *number2* to be copied to variable *number1*

Assignment Statement (cont.)

- Examples of the assignment statement involving calculation:
  i. ***area = length x width***
  ii. ***Multiply variable length with width and store the result into variable area***
  iii. ***Add 5 to variable cnt***
  iv. ***cnt = cnt + 5***
  v. ***Divide sum with 5 and set the result to average***
  vi. ***y =***

Analysis of Simple Problem



Defining the problem

- Involves carefully reading and rereading the problem until you understand completely what is required.
- Additional information will need to be sought to help resolve any ambiguities or deficiencies in the problem specifications.
- The problem should be divided into three separate components:
  1. ***Input*** : a list of the source data provided to the problem
  2. ***Output*** : a list of the outputs required
  3. ***Processing*** : a list of actions needed to produce the required outputs

When reading the problem statement, the input and output components are easily identified, because they are descriptive words such as nouns and adjectives. The **processing** component is also identified easily. The problem statement usually describes the processing steps as actions, using **verbs and adverbs**.

Designing a solution algorithm

- Once the problem has been properly defined, you usually begin with a rough sketch of the steps required to solve the problem.
- The 1st attempt at designing a particular algorithm usually does not result in a finished product…
  - Steps may be left out, or some that are included may later be altered or deleted.
  - Do not hesitate to alter algorithms, or even to discard one and start again, if you are not completely satisfied with it.

Note: If the algorithm is **NOT** correct, the program will never be

Checking the solution algorithm

- After a solution algorithm has been established, it must be tested for correctness.
- This step is necessary because most logic errors occur during the development of the algorithm, and if not detected, these errors can be passed on the program.
- It is much EASIER to detect errors in pseudocode than in the corresponding program code.

Algorithm Development for Sequence Control Structure

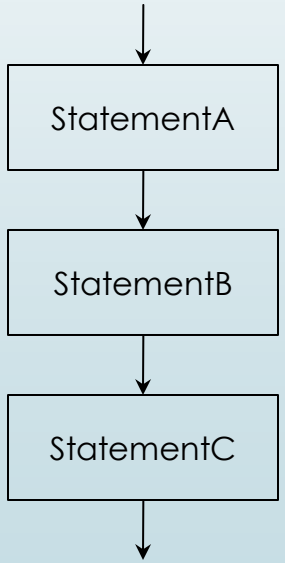
What is the control structure…?

In simple term, control structure is the type of algorithm. The structure could be **sequence**, **selection** and **repetition**. Each of these is needed for a different purpose within a program.

Sequence control structure

- The most common form of control structure – a basic as well as a default program.
- Each step is carried out in order of their position and is only done once.
- A sequence is basically where the code just follows one linear path until it reaches the end.

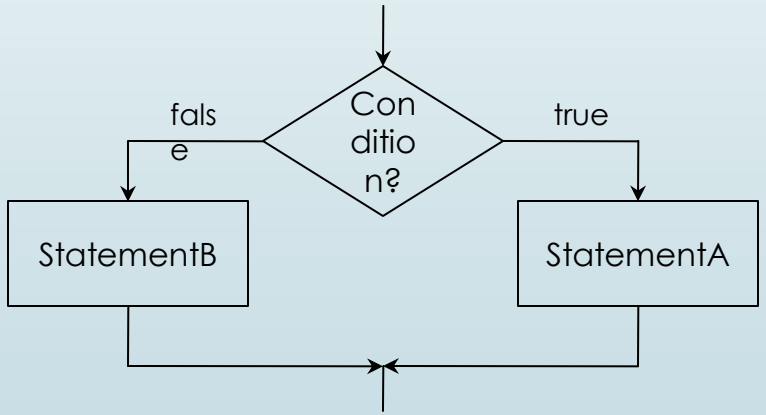
```
Start
  Statement A
  StatementB
  StatementC
End
```



Selection control structure

- This control structure allows the program to choose between one of two or more alternative paths of instructions.
- Decides a particular answer from a set of variable answers and carries out the steps that proceeds it.
  - With this structure, you ask a question, and depending on the answer, you take one of two courses of action.

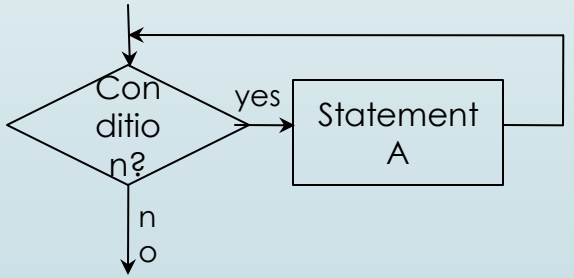
```
Start
  IF condition is true THEN
      Statement A
  ELSE
      StatementB
  ENDIF
End
```



Repetition control structure

- This is basically where a program will repeat a set of instructions until it reaches a certain point where an event has occurred (the condition is met)
- In a loop, you ask a question; if the answer requires an action, you perform the action and ask the original question again. If the question's answer requires that the action be taken again, you take the action and then ask the original question again.

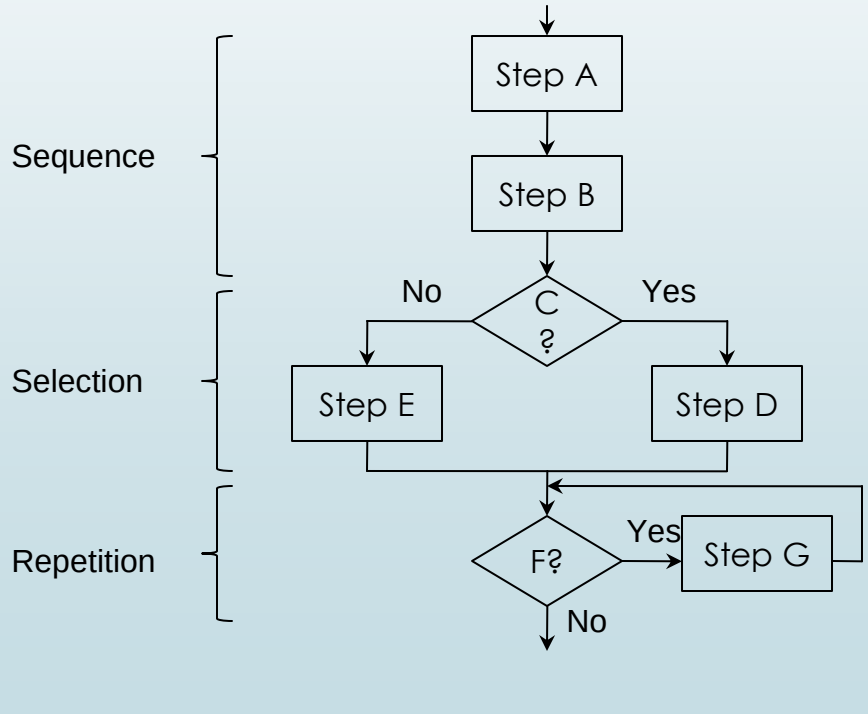
```
Start
   WHILE condition is true
        StatementA
   ENDWHILE
End
```



Combination of three structures

- All logic program can be solved using only these three structures (sequence, selection and repetition).
- The three structures, can be combined in an infinite number of ways. For example: there can be a sequence of steps followed by a selection or repetition followed by sequence.

```
do step A
do step B
 if condition C is true then
    do step D
 else
    do step E
 endif
while condition F is true then
    do step G
```



How to Write a Detailed Algorithm Using Pseudocode?

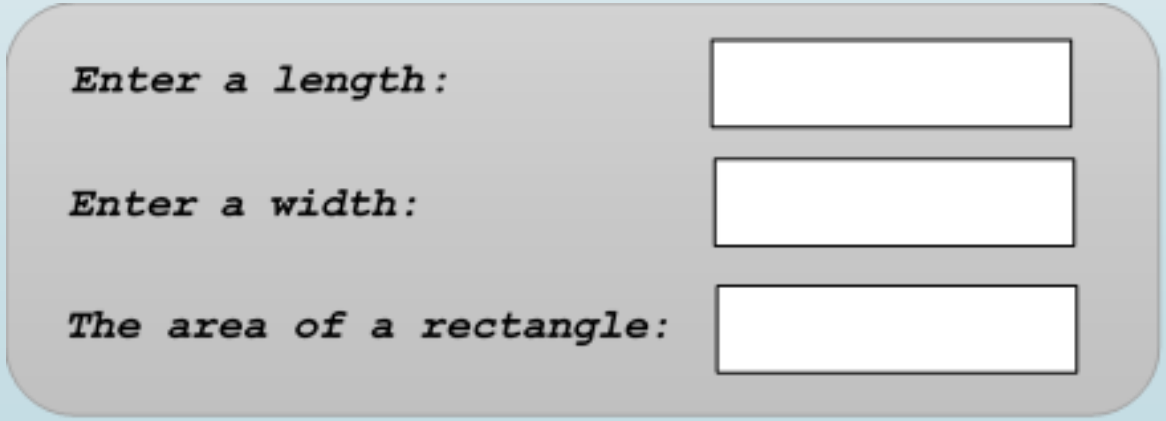
- Described based on the expected screen and a basic algorithm.
- The algorithm usually contains processing term such as the word *calculate*.
- ***detailed algorithm = basic algorithm + expected screen + calculation process***

How to Write a Detailed Algorithm Using Pseudocode? (cont.)

- Example:
  - ***Basic pseudocode:***

```
Start
        Read length, width
        Calculate area of a rectangle
        Display area of a rectangle
End
```

  - ***Expected screen:***

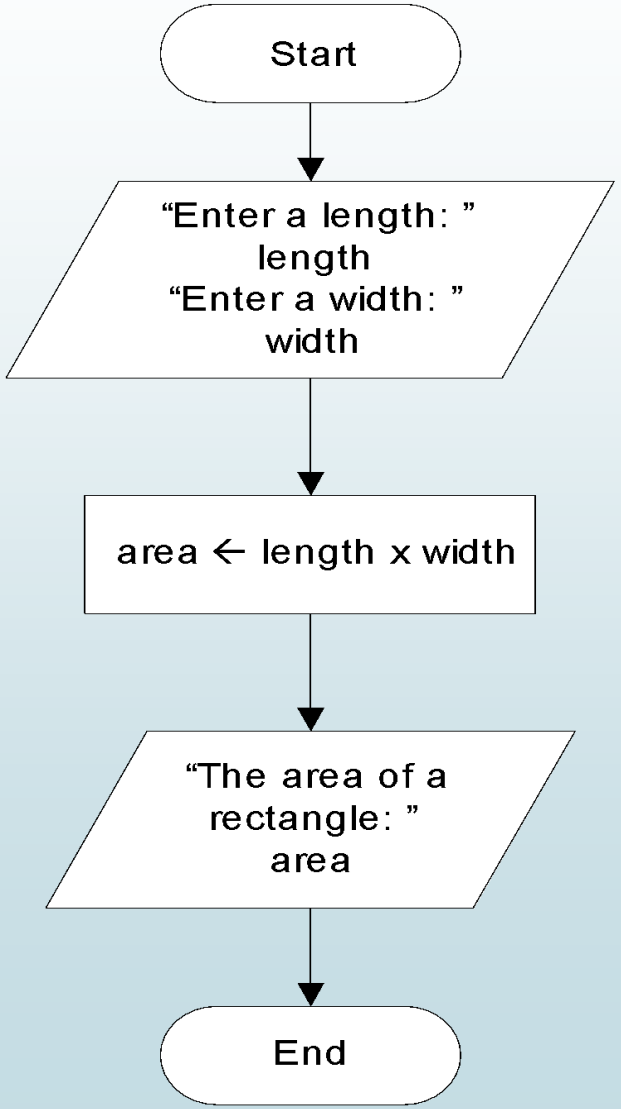


How to Write a Detailed Algorithm Using Pseudocode? (cont.)

- ***Detailed pseudocode:***

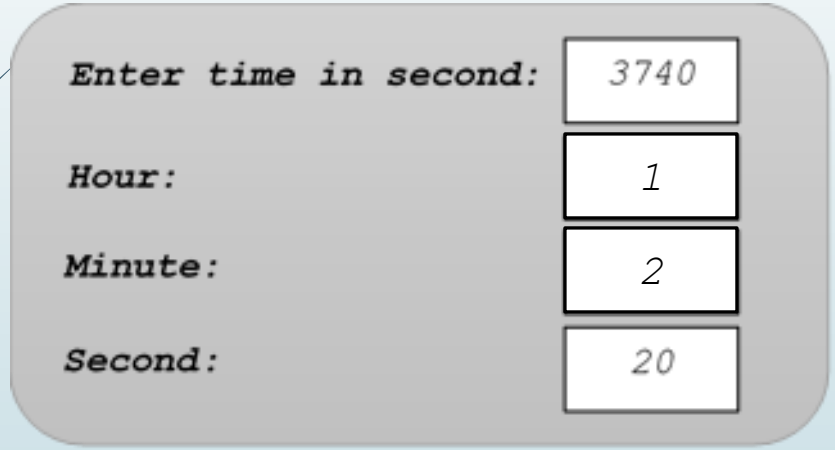
```
Start
Input:
        Display "Enter length: "
        Read length
        Display "Enter width: "
        Read width
Process:
        area = length x width
Output:
        Display "The area of a rectangle: ", area
End
```

How to Draw a Detailed Algorithm Using Flowchart?



Exercise 1

1) Ask the user to enter time in unit second. Convert the value into hour, minute and second. Example of expected screen is:



2) Ask the user to enter an amount of money in ringgit. Convert how many notes in RM1.00, RM5.00, RM10.00, Rm50.00 and RM100.00. Example of expected screen is: