# Code Design

Instructor << >>

## Agenda

- Algorithmic Problem Solving
  - What is an Algorithm?
  - Pseudo Code
  - Flowchart
  - Naming Convention
  - Variables
  - The Structure Theorem
- Communication between modules

## Algorithmic Problem Solving
### What is an algorithm?

- Lists the steps involved in accomplishing a task (like a recipe)
- Defined in programming terms as ‘a set of detailed and ordered instructions developed to describe the processes necessary to produce the desired output from a given input’
- An algorithm must:
  - Be lucid, precise and unambiguous
  - Give the correct solution in all cases
  - Eventually end

## Algorithmic Problem Solving
### Pseudo Code: What is Pseudo Code?

- Detailed and readable description (in structured English) of what a computer program or algorithm must do (states the steps to the problem solution)
  - Statements are written in simple English
  - Each instruction/step is written on a separate line
  - Keywords and indentation are used to signify particular control structures
  - Each set of instructions is written from top to bottom, with only one entry and one exit
- Used as a detailed step in the process of developing a program. It allows designers or lead programmers to express the design in great detail and provides programmers a detailed template for the next step of writing code in a specific programming language

## Algorithmic Problem Solving
### Pseudo Code: Guidelines 1/3

There are six basic computer operations:

1. A computer can receive information
2. A computer can put out information
3. A computer can perform arithmetic
4. A computer can assign a value to a variable or memory location
5. A computer can compare two variables and select one of two alternate actions
6. A computer can repeat a group of actions

1. A computer can receive information
   - Read (information from a file)
   - Get (information from the keyboard)

2. A computer can put out information
   - Write (information to a file)
   - Display (information to the screen)

3. A computer can perform arithmetic
   - Use actual mathematical symbols or the words for the symbols
     - Add number to total
     - Total = total + number
     - +, -, *, /
   - Calculate, Compute also used

4. A computer can assign a value to a piece of data
   - 3 cases
     - To give data an initial value
       - Initialize, Set
     - To assign a value as a result of some processing

## Algorithmic Problem Solving
### Pseudo Code: Guidelines 2/3

- The keywords used for pseudocode in this document are:
  1. for start and finish
     BEGIN MAINPROGRAM, END MAINPROGRAM
  2. for initialisation
     INITIALISATION, END INITIALISATION
  3. for subprogram
     BEGIN SUBPROGRAM, END SUBPROGRAM
  4. for selection
     IF, THEN, ELSE, ENDIF
  5. for multi-way selection
     CASEWHERE, OTHERWISE, ENDCASE
  6. for pre-test repetition
     WHILE, ENDWHILE
  7. for post-test repetition
     REPEAT, UNTIL

## Algorithmic Problem Solving
### Pseudo Code: Guidelines 3/3

- **Keywords** are written in **CAPITALS**.
- **Structural elements** come in **pairs**, e.g.
  - *for every BEGIN there is an END*
  - *for every IF there is an ENDIF, etc.*
- **Indenting** is used to **show structure** in the algorithm.
- The names of **subprograms** are **underlined**.

  *This means that when refining the solution to a problem, a word in an algorithm can be underlined and a subprogram developed.*

- This feature enables the use of the **'top-down' development** concept, where details for a particular process need only be considered within the relevant sub-routine.

## Algorithmic Problem Solving
### Flowchart - Overview

- A diagrammatic representation that illustrates the sequence of operations to be performed to get the solution of a problem.
- Generally drawn in the early stages of formulating computer solutions.
- Facilitate communication between programmers and business people.
- Play a vital role in the programming of a problem and are quite helpful in understanding the logic of complicated and lengthy problems. Once the flowchart is drawn, it becomes easy to write the program in any high level language. Often we see how flowcharts are helpful in explaining the program to others. Hence, it is correct to say that a flowchart is a must for the better documentation of a complex program.

# Algorithmic Problem Solving
## Flowchart - Basic Flowchart Symbols

Rounded box - use it to represent an event which occurs automatically. Such an event will trigger a subsequent action, for example `receive telephone call', or describe a new state of affairs.

Rectangle or box - use it to represent an event which is controlled within the process. Typically this will be a step or action which is taken. In most flowcharts this will be the most frequently used symbol.

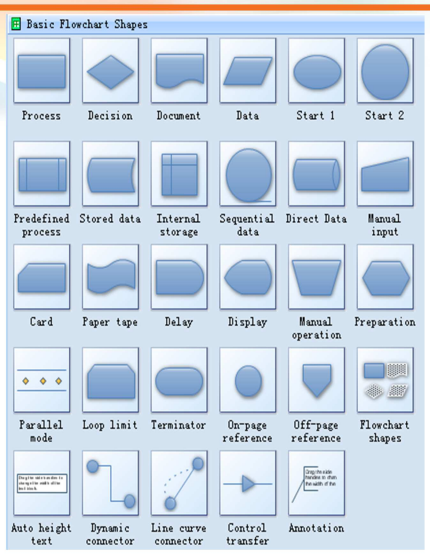
Diamond - use it to represent a decision point in the process. Typically, the statement in the symbol will require a `yes' or `no' response and branch to different parts of the flowchart accordingly.

Circle - use it to represent a point at which the flowchart connects with another process. The name or reference for the other process should appear within the symbol.

# Algorithmic Problem Solving

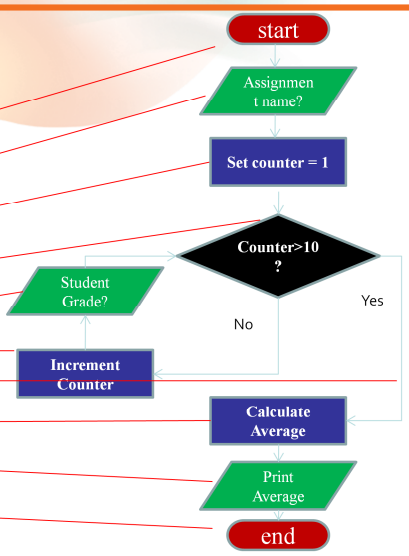
## Flowchart - Guides for drawing flowcharts

Flowcharts are usually drawn using some standard symbols; however, some special symbols can also be developed when required.



## Algorithmic Problem Solving
### Flowchart - Sample

- Pseudocode
  - # Start Program
    - # Prompt for assignment
    - # Set a counter
    - # Loop through grades
      - # Get student grade
      - # Increment counter
    - # End loop
    - # Calculate Average
    - # Print Average
  - # End Program



# Algorithmic Problem Solving

## Flow Chart - Practice Time...

- Practice 1

  Draw a flowchart to find the largest of three numbers A,B, and C
- Practice 2

  Draw a flowchart to find the sum of first 50 natural numbers

## Algorithmic Problem Solving
### Naming Conventions 1/2

- When designing an algorithm, a programmer must introduce some unique names which represents variables or objects in the problem.
- Names should be meaningful.
- Names should be transparent to adequately describe variables (Number1, number2, etc.).

## Algorithmic Problem Solving
### Naming Convention 2/2

- Underscore is used when using more than one word (sales_tax or word_count).
- Most programming language does not tolerate a space in a variable as space would signal the end of a variable name.
- Another method is to use capital letters as a word separator (salesTax or wordCount).

## Algorithmic Problem Solving
### Variables: Using & Naming 1/2

- **Variable**: a memory location whose contents can vary; also called an **identifier**
- Each programming language has it own rules for naming identifiers, including:
  - Legal characters
  - Maximum length
  - Use of upper or lower case
- Variable name must be a single word, but can be formed from several words
  - rate, interestRate, interest_rate

## Algorithmic Problem Solving
### Variables: Using & Naming 2/2

- Choose meaningful names for variables
  - Improves the readability & maintainability of code

**TABLE 1-1:** VALID AND INVALID VARIABLE NAMES FOR AN EMPLOYEE'S LAST NAME

| Suggested variable names for employee's last name | Comments |
|---|---|
| `employeeLastName` | Good |
| `employeeLast` | Good—most people would interpret `Last` as meaning `Last Name` |
| `empLast` | Good—`emp` is short for employee |
| `emlstnam` | Legal—but cryptic |
| `lastNameOfTheEmployeeInQuestion` | Legal—but awkward |
| `last name` | Not legal—embedded space |
| `employeelastname` | Legal—but hard to read without camel casing |

## Algorithmic Problem Solving
### Variables: Assigning Values to Variables

- **Assignment statement**:
  - Assigns a value to a variable
  - Variable must appear on the left side, value on the right side of the assignment operator
  - Right side may be an expression that will be evaluated before storing the value in the variable
- **Assignment operator**: the equal sign (=) in most languages
- **Variable**:
  - Memory location: has an address and a value
  - Value (contents) is used for various operations

## Algorithmic Problem Solving
### Variables: Data Types 1/3

- Two basic data types:
  - Text
  - Numeric
- **Numeric data** stored by numeric variables
- **Text data** stored by string, text, or character variables
- **Constants**:
  - Values that do not change while the program is running
  - Have identifiers, and can be used like variables for calculations, but cannot be assigned new values

## Algorithmic Problem Solving
### Variables: Data Types 2/3

- Some programming languages implement several numeric data types, such as:
  - **Integer**: whole numbers only
  - **Floating-point**: fractional numeric values with decimal points
- Character or string data is represented as characters enclosed in quotation marks
  - “x”, “color”
- Data types must be used appropriately

# Algorithmic Problem Solving
## Variables: Data Types 3/3

**TABLE 1-2:** SOME EXAMPLES OF LEGAL AND ILLEGAL ASSIGNMENTS

*Assume* `lastName` *and* `firstName` *are character variables.*

*Assume* `quizScore` *and* `homeworkScore` *are numeric variables.*

| Examples of valid assignments | Examples of invalid assignments | Explanation of invalid examples |
|---|---|---|
| `lastName = "Parker"` | `lastName = Parker` | If `Parker` is the last name, it requires quotes. If `Parker` is a named string variable, this assignment would be allowed. |
| `firstName = "Laura"` | `"Parker" = lastName` | Value on left must be a variable name, not a constant |
| `lastName = firstName` | `lastName = quizScore` | The data types do not match |
| `quizScore = 86` | `homeworkScore = firstName` | The data types do not match |
| `homeworkScore = quizScore` | `homeworkScore = "92"` | The data types do not match |
| `homeworkScore = 92` | `quizScore = "zero"` | The data types do not match |
| `quizScore = homeworkScore + 25` | `firstName = 23` | The data types do not match |
| `homeworkScore = 3 * 10` | `100 = homeworkScore` | Value on left must be a variable name, not a constant |

# Algorithmic Problem Solving
## The Structure Theorem – Overview 1/6

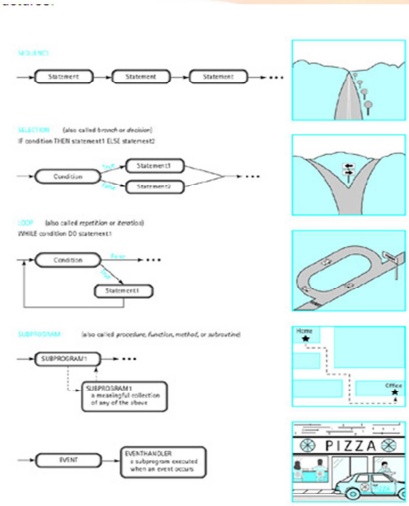



Figure 1.9: Basic control structures of programming languages

### Three basic rules

**1) Sequence**

- Get up from bed
- Dress up
- Go to the shower
- Go to work

**2) Selection**

- If car is broken, go to work by bus. Otherwise go to work by car

**3) Repetition**

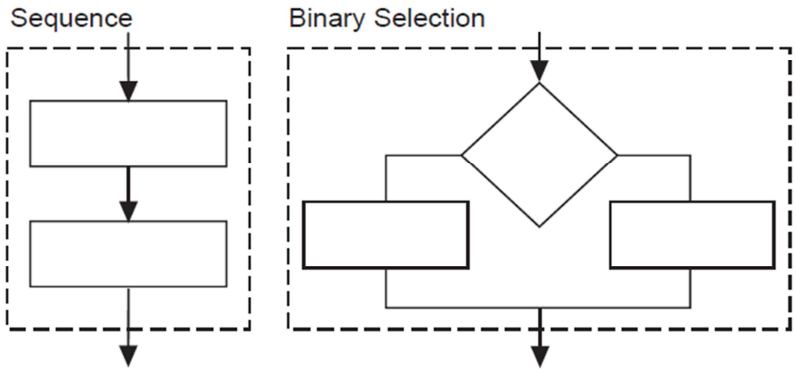
- Drink until the bottle is empty

## Algorithmic Problem Solving
### The Structure Theorem – Overview 2/6

- It is considered **good practice** for a single flowchart
  - **never** to **exceed** the bounds of **one page**.
- If a flowchart does not fit on one page,
  - this is one instance in which the **better solution** is to use **refinement**
  - which results in the **creation of subprograms**.
- **Subprograms** on separate pages are **more desirable than** using a **connector** to join flowcharts over more than one page.
- A flowchart expressing the solution to an involved problem may have:
  1. the **main program** flowchart on **one page**
  2. with **subprograms** continuing the problem solution **on subsequent pages**.
- *Regardless of page size*, it is also important to start any **complex algorithm** with:
  1. a **clear, uncluttered main line.**
  2. This should reference the **required subroutines**
  3. whose **detail** is shown **in separate flowcharts**.

## Algorithmic Problem Solving
### The Structure Theorem – Overview 3/6

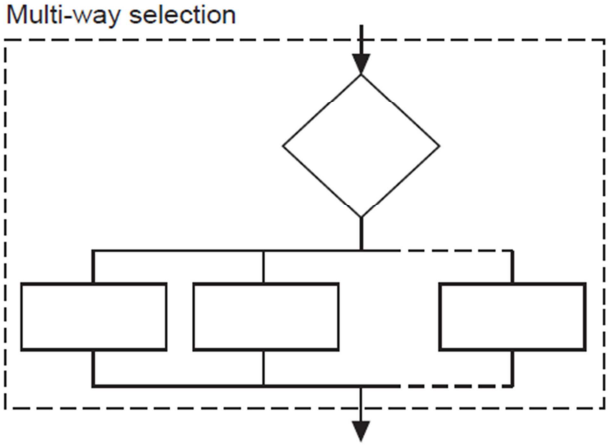
**Each of the five acceptable structures can be built from the basic elements as shown below.**



## Algorithmic Problem Solving
### The Structure Theorem – Overview 4/6

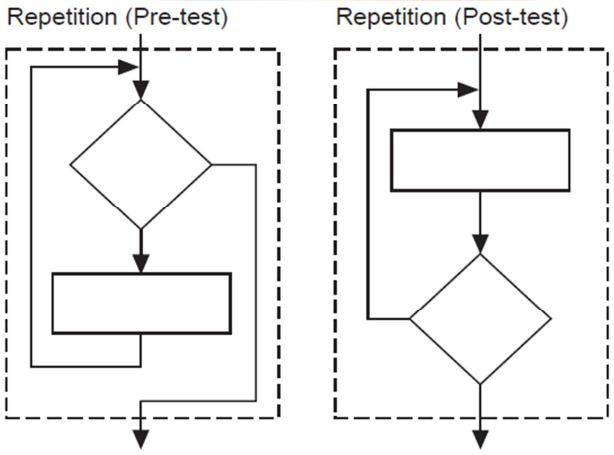
Each of the five acceptable structures can be built from the basic elements as shown below.

Multi-way selection



## Algorithmic Problem Solving
### The Structure Theorem – Overview 5/6

**Each of the five acceptable structures can be built from the basic elements as shown below.**



## Algorithmic Problem Solving
### The Structure Theorem – Overview 6/6

**Each of the five acceptable structures can be built from the basic elements as shown previously.**

In all cases note there is:

1. **only one entry point** to the structure
2. and **one exit point** as indicated by the dashed boxes.

Since each structure can be thought of as a *process*
*(as shown* by the dashed boxes containing the structure),

more complex algorithms can be constructed by
**replacing** *any single process* **by**
one or ***other*** of the ***structures***.

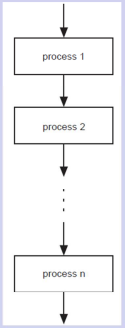
## Algorithmic Problem Solving
## The Structure Theorem - Sequence

In a computer program or an algorithm,
**sequence involves simple steps** which are
to be **executed one after the other**.
The steps are executed in the same order in which they are written.

In **pseudocode**,
sequence is expressed as:

process 1
process 2
…
…
process n

In a **flowchart**,
sequence is expressed as:
(The arrowheads are optional if the flow is top-to-bottom.)



# Algorithmic Problem Solving
## The Structure Theorem – Sequence Example

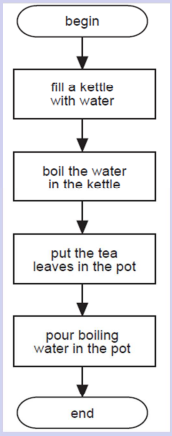
**An Example Using Sequence**

**Problem:** Write a set of instructions that describe how to make a pot of tea.

**Pseudocode**

```
BEGIN
    fill a kettle with water
    boil the water in the kettle
    put the tea leaves in the pot
    pour boiling water in the pot
END
```

**Flowchart**



# Algorithmic Problem Solving
## The Structure Theorem - Selection

**Selection** is used in a computer program or algorithm
to **determine which** particular step or set of **steps is to be executed**.

A selection statement can be used to choose a specific path dependent on a condition.
There are two types of selection:

1. **binary** (*two-way branching*) selection and
2. **multi-way** (*many way branching*) selection.

Following is a description of each.

### Binary Selection

As the name implies, binary selection allows the **choice between two possible paths.**

**If the condition is met** then *one path is taken*,
**otherwise** the *second possible path is followed*.

- In the examples that follow, the first case described requires a **process** to be completed **only if** the condition is **true**.
- The process is **ignored if** the condition is **false**.

In other words there is **only one path** that **requires processing** to be done, so the processing free path is left out rather than included saying 'do nothing'.

### Multi-way Selection

Multi-way selection allows for any **number of possible choices,** or cases.

The path taken is determined by the **selection of the choice** which is true.

Multi-way selection is often referred to as a **case structure**.

# Algorithmic Problem Solving
## The Structure Theorem – Binary Selection 1/3

**Selection** is used in a computer program or algorithm
to **determine which** particular step or set of **steps is to be executed**.

**Binary Selection**

In **pseudocode**, binary selection is expressed in the following ways:

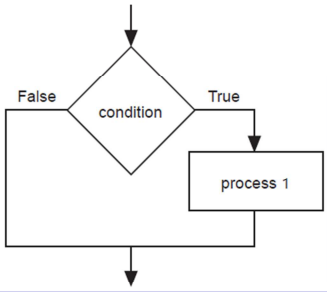
1.
```
IF condition THEN
        process 1
ENDIF
```

2.
```
IF condition THEN
        process 1
ELSE
        process 2
ENDIF
```

**Binary Selection**

In **flowcharts**, binary selection is expressed in the following ways:

1.



# Algorithmic Problem Solving
## The Structure Theorem – Binary Selection 2/3

**Selection** is used in a computer program or algorithm
to **determine which** particular step or set of **steps is to be executed**.

**Binary Selection**

In **pseudocode**, binary selection is expressed in the following ways:

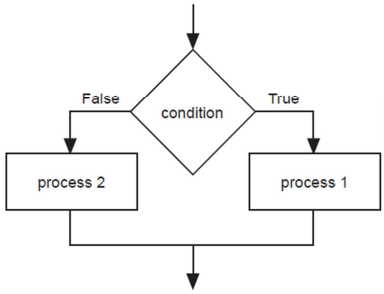
1.
```
IF condition THEN
        process 1
ENDIF
```

2.
```
IF condition THEN
        process 1
ELSE
        process 2
ENDIF
```

**Binary Selection**

In **flowcharts**, binary selection is expressed in the following ways:

2.



# Algorithmic Problem Solving
## The Structure Theorem – Binary Selection 3/3

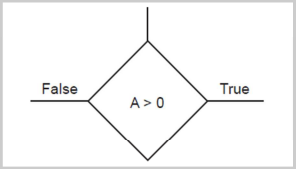
**Note:** In a flowchart it is most important to indicate

1. **which path** is to be followed when the **condition is true**, and
2. **which path** to follow when the **condition is false.**

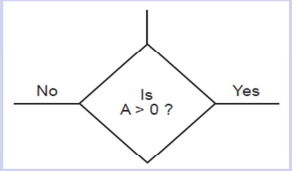
Without these indications the flowchart is open to more than one interpretation.

**Note:** There are two acceptable ways to represent a decision in all of the structures.
*Either method is acceptable. For consistency, the method 1 is used throughout this document.*

1. The **condition** is expressed as a **statement** and the two possible outcomes are indicated by
   - True
   - False




2. The **condition** is expressed as a **question** and the two possible outcomes are indicated by
   - Yes
   - No



# Algorithmic Problem Solving

## The Structure Theorem – Binary Selection Examples 1/2

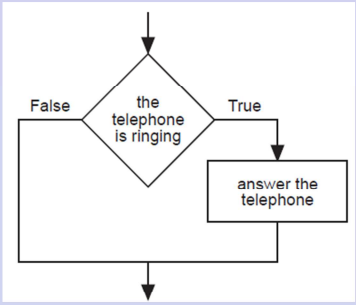
**Selection** is used in a computer program or algorithm
to **determine which** particular step or set of **steps is to be executed**.

**Examples Using Binary Selection**
**Problem 1:** Write a set of instructions to describe when to answer the phone.

**Binary Selection**
**Pseudocode**

```
IF the telephone is ringing THEN
        answer the telephone
ENDIF
```

**Binary Selection**
**Flowchart**



# Algorithmic Problem Solving

## The Structure Theorem – Binary Selection Examples 2/2

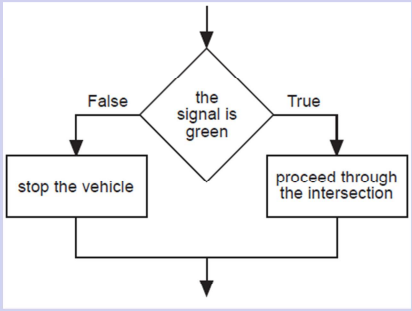
**Selection** is used in a computer program or algorithm
to **determine which** particular step or set of **steps is to be executed**.

**Examples Using Binary Selection**
**Problem 2:** Write a set of instructions to follow when approaching a set of traffic control lights.

**Binary Selection**
**Pseudocode**

```
IF the signal is green THEN
        proceed through the intersection
ELSE
        stop the vehicle
ENDIF
```

**Binary Selection**
**Flowchart**



# Algorithmic Problem Solving
## The Structure Theorem – Multi-way Selection

**Selection** is used in a computer program or algorithm
to **determine which** particular step or set of **steps is to be executed**.

**Multi-way Selection**

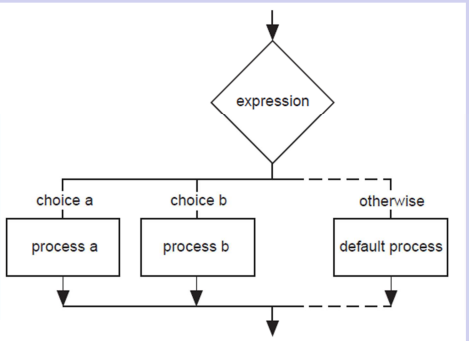
In **pseudocode**, multiple selection is expressed as:

```
CASEWHERE expression evaluates to
      choice a    :    process a
      choice b    :    process b
         .              .
         .              .
         .              .
   OTHERWISE  :    default process
ENDCASE
```

**Note:** As the flowchart version of the multi-way selection indicates, **only one** process on each pass is executed as a result of the implementation of the multi-way selection.

**Multi-way Selection**

In **flowcharts**, multi-way selection is expressed as:



# Algorithmic Problem Solving

## The Structure Theorem – Multi-way Selection Examples

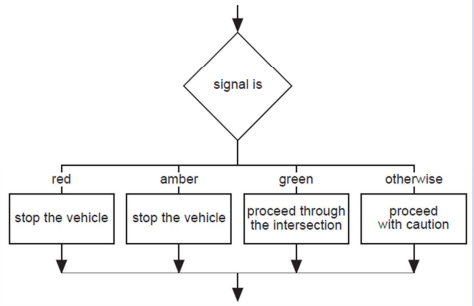
**Selection** is used in a computer program or algorithm
to **determine which** particular step or set of **steps is to be executed**.

**Example Using Multi-way Selection**
**Problem:** Write a set of instructions that describes how to:
respond to all possible signals at a set of traffic control lights.

**Multi-way Selection**
**Pseudocode**

```
CASEWHERE signal is
      red          : stop the vehicle
      amber        : stop the vehicle
      green        : proceed through the intersection
   OTHERWISE  : proceed with caution
ENDCASE
```

**Multi-way Selection**
**Flowchart**



## Algorithmic Problem Solving
### The Structure Theorem – Repetition 1/3

**Repetition** allows for a **portion of** an **algorithm** or computer program
to be done **any number of times**
dependent on some **condition being met**.
An occurrence of repetition is usually known as **a loop**.

An **essential feature** of repetition is that
each loop has a **termination condition**
to **stop** the repetition,
or the obvious outcome is that
the *loop never completes* execution (*an infinite loop*).

The **termination condition** can be checked or **tested**
1. at the **beginning** and is known as a **pre-test** loop *or*
2. at the **end** of the loop and is known as a **post-test** loop.

# Algorithmic Problem Solving
## The Structure Theorem – Repetition 2/3

**Repetition** allows for a **portion of** an **algorithm** or computer program
to be done **any number of times**
dependent on some **condition being met**.
An occurrence of repetition is usually known as **a loop**.

**Repetition: Pre-Test**

A pre-tested loop is so named because the **condition** has to be met at the **very beginning of the loop** or the body of the loop is not executed.
This construct is often called a *guarded loop.*
The body of the loop is executed repeatedly while the termination condition is true.

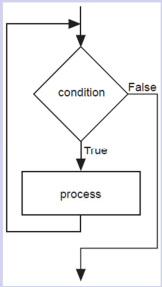
**Repetition**

In **pseudocode**, pre-test repetition is expressed as:

```
WHILE condition is true
        process(es)
ENDWHILE
```

**Repetition**

In **flowcharting** pre-test repetition is expressed as:



# Algorithmic Problem Solving
## The Structure Theorem – Repetition 3/3

**Repetition** allows for a **portion of** an **algorithm** or computer program
to be done **any number of times**
dependent on some **condition being met**.
An occurrence of repetition is usually known as **a loop**.

**Repetition: Post-Test**

- A post-tested loop **executes the body** of the loop **before testing** the **termination condition**.
- This construct is often referred to as an *unguarded loop.*
- The body of the loop is repeatedly executed until the termination condition is true.

An **important difference** between a pre-test and post-test loop is that the **statements of a post-test loop** are **executed at least once** even if the condition is originally true, whereas the **body of the pre-test loop** *may never be executed* if the *termination condition is originally true*.
A close look at the representations of the two loop types makes this point apparent.

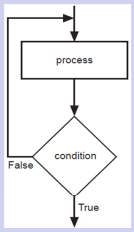
**Repetition**

In **pseudocode**, post-test repetition is expressed as:

```
REPEAT
        process
UNTIL condition is true
```

**Repetition**

In a **flowchart** post-test repetition is expressed as:



# Algorithmic Problem Solving
## The Structure Theorem – Repetition Examples 1/2

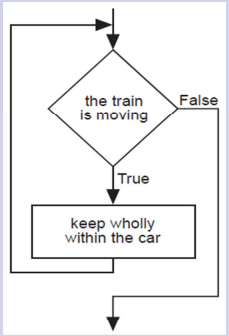
**Repetition** allows for a **portion of** an **algorithm** or computer program
to be done **any number of times**
dependent on some **condition being met**.
An occurrence of repetition is usually known as **a loop**.

**An Example Using Pre-Test Repetition**
**Problem:** Determine a safety procedure for travelling in a carriage on a moving train.

**Pre-test Repetition**
**Pseudocode**

```
WHILE the train is moving
        keep wholly within the carriage
ENDWHILE
```

**Pre-test Repetition**
**Flowchart**



# Algorithmic Problem Solving
## The Structure Theorem – Repetition Examples 2/2

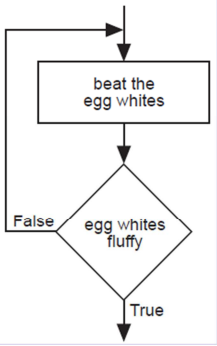
**Repetition** allows for a **portion of** an **algorithm** or computer program
to be done **any number of times**
dependent on some **condition being met**.
An occurrence of repetition is usually known as **a loop**.

**An Example Using Post-Test Repetition**
**Problem:** Determine a procedure to beat egg whites until fluffy.

**Post-test Repetition**
**Pseudocode**

```
REPEAT
        beat the egg whites
UNTIL fluffy
```

**Post-test Repetition**
**Flowchart**



## Algorithmic Problem Solving
### Pseudo codes - Practice Time…

- Practice 3 - Write an algorithm with pseudo codes

  Request the user to input two numbers. If the sum of those numbers are greater than 12, program outputs "Big numbers you gave". Otherwise the program outputs "Small numbers you gave"

- Practice 4 – Write an algorithm with pseudo codes

  Computing Weekly Wages;

  Gross pay depends on the pay rate and the number of hours worked per week. However, if you work more than 40 hours, you get paid time-and-a-half for all hours worked over 40. Pseudo-code the task of computing gross pay given pay rate and hours worked.

## Algorithmic Problem Solving
### Pseudo codes - Practice Time...

- Practice 5
  - Pseudo-code the task of computing the final price of an item after figuring in sales tax.
  - Note the three types of instructions: input (get), process/calculate (=) and output (display)
  - Note that the operations are numbered and each operation is unambiguous and effectively computable. We also extract and list all variables used in our pseudo-code. This will be useful when translating pseudo-code into a programming language

# Algorithmic Problem Solving
## The Structure Theorem – Subprograms 1/3

**Subprograms**, as the name implies, are **complete part-programs** that are used from **within the main program** section.

- They allow the process of **refinement** to be used to develop solutions to problems that are easy to follow.
- Sections of the solution are developed and presented in **understandable chunks**, and because of this,
- subprograms are particularly useful when using the **top-down** method of solution development.

When using subprograms it is important that
1. the solution expression indicates **where the main program branches to a subprogram**.
2. It is equally important to indicate exactly **where the subprogram begins**.

In **pseudocode,**
the **statement in the main program** that is expanded in a subprogram is **underlined** to indicate that *further explanation follows*.
The expanded subprogram section should be identified by
1. using the keywords BEGIN SUBPROGRAM
   - followed by the underlined title used in the main program.
2. The end of the subprogram is marked by the keywords END SUBPROGRAM
   - and the underlined title used in the main program.

When using **flowcharts**,
a subprogram is shown by **an additional vertical line** on **each side of the process box**.
This indicates that the subprogram is **expanded elsewhere**.
The **start** and **end** of the subprogram flowchart **uses** the **name** of the **subprogram** in the **termination boxes**.

# Algorithmic Problem Solving
## The Structure Theorem – Subprograms 2/3

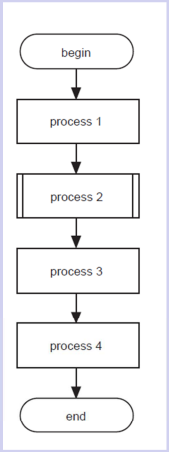
**Subprograms**, as the name implies, are **complete part-programs** that are used from **within the main program** section.

**Subprograms**
**Pseudocode**

```
BEGIN MAINPROGRAM
        process I
        process 2
        process 3
        process 4
END MAINPROGRAM

BEGIN SUBPROGRAM process 2
        do this
        do that
END SUBPROGRAM process 2
```

**Subprograms**
**Flowchart**



# Algorithmic Problem Solving
## The Structure Theorem – Subprograms 3/3

**Subprograms**, as the name implies, are **complete part-programs** that are used from **within the main program** section.

In many cases a subprogram can be written to **do the same task** at two or **more points in an algorithm**.
- Each time the subprogram is called, *it may operate on different data.*
- To indicate the data to be used one or more **parameters** are used.

The parameters allow the author to **write a general algorithm** using the formal parameters.
When the subprogram is executed, the algorithm carries out its task on the **actual parameters** given at the call.

The parameters to be used by a subprogram are provided as a **list in parentheses** *after the name of the subprogram.*
There is no need to include them at the end of the algorithm.

**Example of Using Subprograms with one Parameter in Pseudocode**

```
BEGIN MAINPROGRAM
        read (name)
        read (address)
END MAINPROGRAM

BEGIN SUBPROGRAM read (array)
        Set pointer to first position
        Get a character
        WHILE there is still more data AND there is room in the array
                store data in the array at the position given by the pointer
                Increment the pointer
                get data
        ENDWHILE
END SUBPROGRAM read (array)
```

The first time that the subprogram *'read'* is called:
1. the characters are read into the array called *'name'*
2. the second time, the data (characters) are read into the array called *'address'*.

## Communication between modules
### An Overview

- Necessary to consider flow of information between modules
- This flow of information is called 'intermodule communication' and can be accomplished by the scope of the variable

## Communication between modules
### Scope of a variable

- The portion of a program in which that variable has been defined and to which it can be referenced
- Variables can be global where the scope of the variable is the whole program
- Scope of the variable is simple the module which it is defined

## Communication between modules
### Global & Local data

**Global data**

- Data that can be used by all the modules in a program
- Every module in the program can access and change data
- Lifetime of a global variable spans the execution of the whole program

**Local data**

- Variable are defined within the submodule are called local variables
- The scope of a local variable is simply the module in which it is defined
- The lifetime of a local variable is limited to the execution of the single submodule in which it is defined

## Communication between modules
### Side effects

- Side effect is a form of a cross-communication of a module with other parts of a program,
- Occurs when a subordinate module alters the value of a global variable inside a module

## Communication between modules
### Passing parameters

- Parameters are simply data items transferred from a calling module to its subordinate module at the time of calling
- To pass parameters between modules, two things can happen:
  - The calling module must name the parameters that it wants to pass to the submodule
  - The submodule must be able to receive those parameters and return them to the calling module if required
- Parameters names that appear when a submodule is defined are known as formal parameters
- Variables and expressions that are passed to a submodule in a particular call are called actual parameters

## Communication between modules
### Value and reference parameters

Parameters may have one of three function:

1. To pass information from a calling module to a subordinate module
2. To pass information from a subordinate module to its calling module
3. To fulfil a two-way communication role

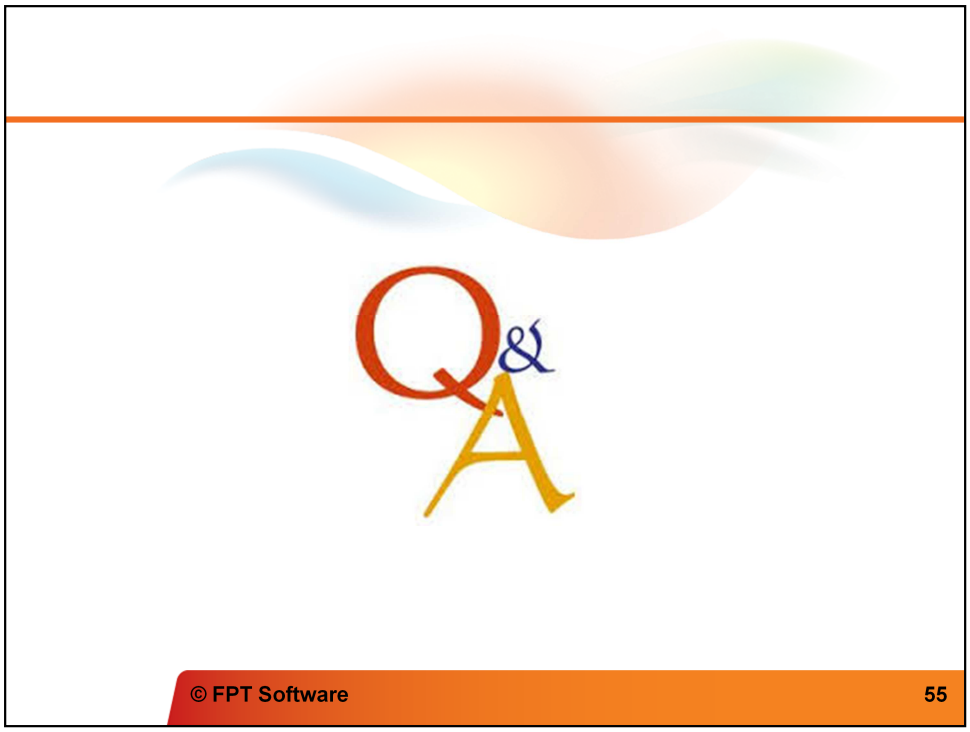
- Value parameters
  - Value parameters pass a copy of the value of a parameter from one module to another
- Reference parameters
  - Reference parameter pass the memory address of a parameter from one module to another

## Evolution of Programming Techniques

- Programming began in the 1940s, using memory addresses and machine code directly
- Higher level languages were developed to allow English-like instructions
- Older programs were “monolithic,” and ran from beginning to end
- Newer programs contain modules that can be combined to form programs

## Evolution of Programming Techniques

- Two major programming techniques:
  - Procedural programming
  - Object-oriented programming
- **Procedural programming**: focuses on
  - The procedures that programmers create
  - The actions performed on data
- **Object-oriented programming**: focuses on
  - The objects that represent real-world things and their attributes and behaviors
- Both techniques employ reusable program modules


Q&A