

GENERAL INFORMATICS

Chapter 3.

The Representation of Processing Algorithms



3.1. Algorithm definition



3.2. Steps in computer problem solving process



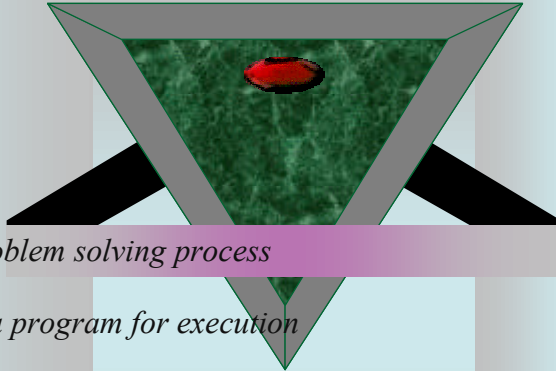
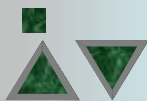
3.3. Steps for preparing a program for execution



3.4. Executing a program



3.5. The Description of Algorithms



Algorithm definition



An algorithm is represented as a set of rules (R_i) which can be applied to the same class of problems (CP_i) in order to obtain the solution (S) by means of some sequential unique operations (OS_i) started, eventually, with some initial conditions (CI_i) :

$$S = R_i(OS_i(CP_i([CI_i])))$$





- ◆ Algorithm is a fundamental concept of computer programming.
- ◆ A program is a set of instructions written in a language designed to make a (micro)computer perform a series of specific tasks. The instructions tell computers exactly what to do and exactly when to do it.
- ◆ A programming language is a set of grammar rules, characters, symbols and words – the vocabulary – in which those instructions are written. Programming is the designing and writing programs.



- ◆ A computer program – does no matter if is a single procedure or function, utility tool, an application or the operating system itself – is nothing else than a list of instructions that the microprocessor (or generally processor) can execute.
- ◆ An algorithm is a prescribed set of well-defined instructions for solving a problem in a finite number of steps.





- ◆ Any algorithm must have the following properties:
 - **Generality** – the algorithms must solve a class of problems not a particular one;
 - **Finality** – the algorithm must find a solution in a finite number of steps;
 - **Clarity** – the algorithm must specify all steps that must be realized to obtain a correct solution of the problem.



Steps in problem solving process by using a programming environment

- ◆ The problem solving process starts with the problem specification and ends with a concrete (and correct) program.
- ◆ The steps to do in the problem solving process may be: problem definition, problem analysis, algorithm development, coding, program testing and debugging, and documentation.





- ◆ The stages of analysis, design, programming, implementation, and operation of an information system forms the life cycle of the system.
- ◆ We briefly describe the steps in problem solving process by using a programming environment (it can allow the “around” application programming by the possibility of generating programs from general templates, for example) and by considering only a specific process from the whole system. In this context the stages can be:



- ◆ **1st. Defining/Specifying the problem** [Theme] - by answering to questions as:
 - What the computer program do?
 - What tasks will it perform?
 - What kind of data will it use, and where will get its data from?
 - What will be the output of the program?
 - How will the program interact with the computer user?





Specifying the problem requirements forces you to state the problem clearly and unambiguously and to gain a clear understanding of what is required for its solution.

Your objective is to eliminate unimportant aspects and to focus on the root problem, and this may not be as easy as it sound.



♦ **2nd. Analyzing the problem** [Analysis] involves identifying the problem (a) inputs, that is, the data you have to work with; (b) outputs, the desired results; and (c) any additional requirements or constraints on the solution.





- ◆ **3rd. Algorithm development:** find an algorithm for its solution [Design].

Write step-by-step procedure and then verify that the algorithm solves the problem as intended.



The development can be expressed as:

- **pseudocode** – a narrative description of the flow and logic of the intended program, written in plain language that expresses each step of the algorithm;
- **flowchart** - a graphical representation that uses graphic symbols and arrows to express the algorithms.
- ◆ After you write the algorithm you must realize step-by-step simulation of the computer execution of the algorithm in a so called desk-check process (verifying the algorithm).





- ◆ **4th. Coding** (or programming): is the process of translating the algorithm into the syntax of a given programming language [Programming]. You must convert each algorithm step into one or more statements in a programming language.



- ◆ **5th. Testing and debugging:**
 - **testing** means running the program, executing all its instructions/functions, and testing the logic by entering sample data to check the output;
 - **debugging** is the process of finding and correcting program code mistakes:
 - **syntax errors**;
 - **run-time errors**;
 - **logic errors** (or so called **bugs**).
 - **field testing** is realized by users that operate the software with the purpose of locating problems.





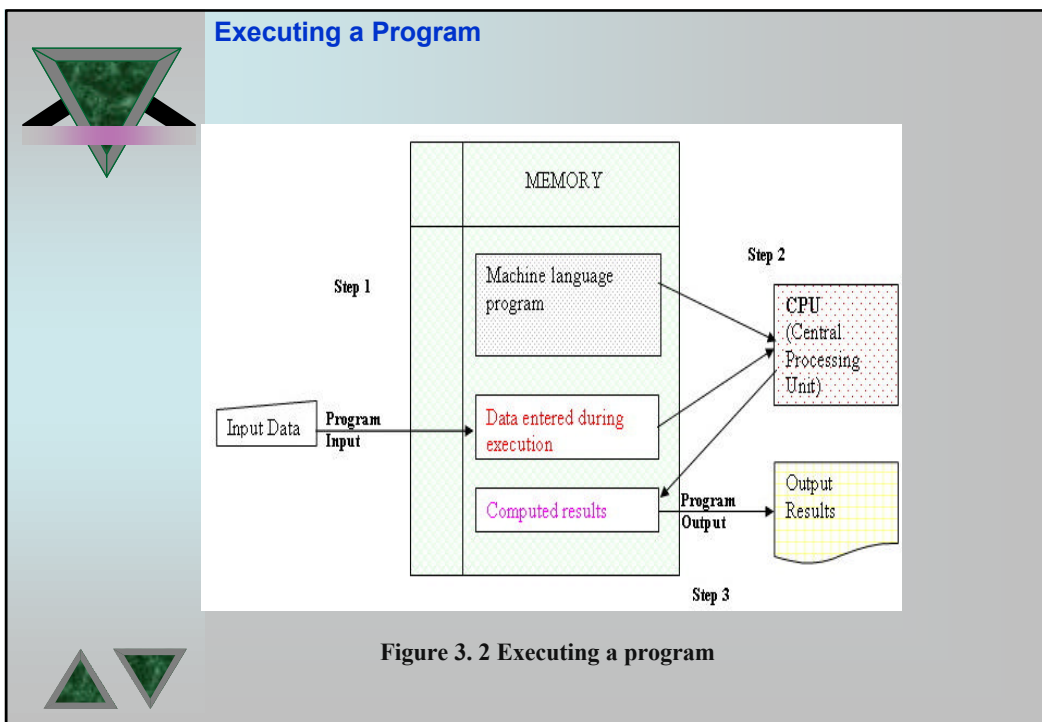
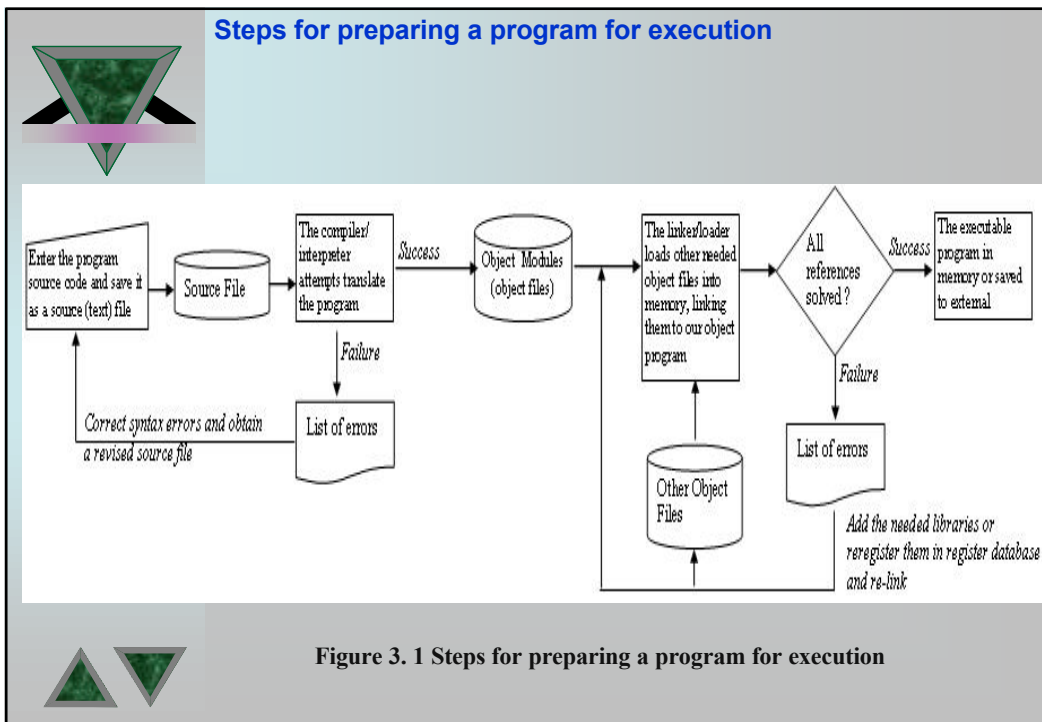
♦ **6th. Documenting the program by:**

- **internal documentation;**
- **external documentation.**



♦ **7th. Integrate the program in the data process flow (Implementation) and use the program to solve problems [Exploitation].**







The Description of Algorithms

Process, Program and Document Flowcharts

In the general analysis of an information system the designer can use three kinds of flowcharts: document, process, and program flowchart and/or narrative descriptions in pseudocode.

Document Flowchart - is a diagram illustrating where documents originate and where copies of them are sent.

Process Flowchart - is a diagram that shows the data and operation for a single process.

Program Flowchart - is a diagram that shows the data, steps, and logic of a process operation, does show logic and additional processing detail.

Pseudocode - is a set of succinct instructions to the programmer using some of the syntax of the language in which the application will be programmed.


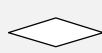
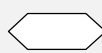





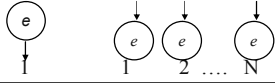

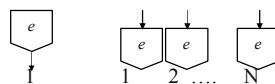

The Description of Algorithms by means of logical diagrams (program flowcharts)

The Basic Symbols Used in Drawing Program Flowcharts

	Terminal/Interrupt (Start/Stop) - It marks the START and the STOP of the logical diagram and/or the beginning and ending of a procedure (function, subroutine). Inside is written, case usage dependent, either the word START/STOP (for main programs) or the call model for the procedure (function or subroutine);
	Process/Calculation Block - It is used for representing calculus formula, changes of the index values, assigning of values. Inside are written the expressions of that operations;
	Procedure - A call to another process or procedure. Inside is written the name of the procedure followed by the list of parameters in the order as specified in the call model;



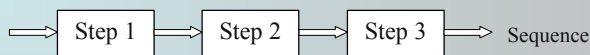
		Decision Block - It is used for the passing in different points, of the diagram, depending on the true state or false state of a logical condition;
		Preparation – used to specify Open/Close operations on files (or computer ports, or connections to host computers or to long distance databases);
		Input/Output Block - It is used to represent read and write operations on data. Inside is written the operation sense such as <i>reading</i> (expressed by commands as: Read, Input, Accept, Select, Get, In) or <i>writing</i> (expressed by commands as: Write, Display, Put, Update, Modify, Print, Out) followed by the logical device name (allowed by the system at opening) and the list of values/variables on which the command acts.

		Onpage Connector - Used to link different points of the logical diagram in the same page. Inside is written a label (can be a digit/number or a word – is preferable to be of significance for the reader) that must be defined only once as entry point (the arrow goes from symbol) and how many times needed as exits (or go to). The entry labels must be unique for a particular flowchart; 
		Offpage Connector - It links different points of the logical diagram in different pages. It has the same rules as the Onpage Connector. 
		Flow – Is a connection <i>from ... to</i> that links all the blocks in the diagram and shows the transfer sense of the information.

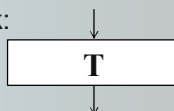


Sequential Structure

The sequential structure can be represented as a sequence of operations:



or as a transformation block:



where **T** from the graphical representation is a data transformation such as assignments and/or computations, data type declarations, input/output operations etc.



Sequential Structure

The *assignments* are represented, in the used programming languages, in one of the next following formats:

$x \leftarrow 0$ $x = 0$ $x := 0$
store 0 To x
 $x = \text{expression}$
 $\text{variable} = \text{expression}$

The interpretation of the last form is: the *variable* before the assignment operator is assigned a value of the *expression* after it, and in the process, the previous value of *variable* is destroyed.

An *expression* can be an expression on character string, a logical expression or arithmetic expression. It can be a variable, a constant, a literal, or a combination of these connected by appropriate operators.

An assignment statement stores a value or a computational result in a variable and is used to perform most arithmetic operation in a program.





Sequential Structure

1. An *expression on character string* can be built (in VB but not only) using:

- the concatenation operator: & or +

- intrinsic functions for extracting substrings from a string variable or string constant such as:

Right(*string*,*number_of_characters*) - extracting substring from the end

Left(*string*,*number_of_characters*) - extracting substring from the beginning

- functions that manipulate strings:

Cstr(*expression*) – convert the expression in a character string;

Lcase(*string_expression*) – convert the string in lower case;

Ltrim(*string*), **Rtrim**(*string*), **Trim**(*string*) – eliminates the spaces (trailing) from left (leading blanks), right and, respectively left-right;

Str(*number*) – converts number in string;

Ucase(*string*) – converts string to uppercase.



Sequential Structure

2. A *logical expression* can be:

- **simple**, with the general syntax:

<variable>[*<relation_operator>**<variable>*]

or

<variable>[*<relation_operator>**<constant>*]

<relation_operator>::=*<|<=>|>=>|<>*

- **complex**, with the general syntax:

*<logical_expression1>**<logical_operator>**<logical_expression2>*

where the *logical_operator* can be:

And, Or as binary operators (connectives);

Not as unary operator.

The precedence of evaluation of logical operators is Not, And, Or.

Other logical operators:

e1 Eqv *e2* - equivalence;

e1 Imp *e2* - logical implication;

o1 Is *o2* - compare two object reference string like pattern;

e1 Xor *e2* - exclusive or;





Sequential Structure

3. An *arithmetic expression* uses the syntax:

$\langle operand1 \rangle \langle arithmetic_operator \rangle \langle operand2 \rangle$

where:

- $\langle operand1 \rangle, \langle operand2 \rangle$ can be numeric variables, constants or arithmetic expressions (or calls to functions that returns numeric values)

- *arithmetic_operator* (binary operators) is one of the following:

Operator	Significance	Example (for $x=3$)	Evaluation Priority (in descending order)
+	Add	$x+1 \rightarrow 4$	1
-	Subtract	$x-1 \rightarrow 2$	1
*	Multiply	$4*x \rightarrow 12$	2
/	Divide	$x/2 \rightarrow 1,5$	2
\ or div	Integral division	$x \setminus 2 \rightarrow 1$	2
mod	Modulus	$x \text{ mod } 2 \rightarrow 1$ remainder 1	2
^	Exponentiation	$x^2 \rightarrow 9$	3



Sequential Structure

If an expression contains more than one operator and/or parentheses the following rules of evaluation applies:

1. **Parentheses rule:** All expression in parentheses must be evaluated separately. Nested parenthesized expressions must be evaluated from inside out, with the innermost expression evaluated first.

2. **Operator precedence rule.** Operators in the same expression are evaluated in the following order:

^	first
*, /, mod, \, div	second
+, -	last

3. **Left associative rule.** Operators in the same expression and at the same precedence level are evaluated left to right.





Sequential Structure - Variables and Constants Declarations

The naming of constants and variables in VB uses the rules:

1. An identifier must begin with a letter;
2. Can't be longer than 255 characters;
3. Can't contain embedded period or embedded type declaration character;
4. Must be unique in same scope (the range from which the variable can be referenced).

Variables: are named storage locations that can contain data that can be modified during program execution. The variables are the memory cells used for storing program's input data and its computational results. The explicit declaration of variables is realized in VB by using the Dim statement:

`Dim variable[As data_type] [,variable[As data_type]]...`

where:

data_type can be one of Byte, Boolean, Integer, Long, Single, Double, Currency, Decimal, String, Date, [*user_defined*], Variant, Object as described in table 3.1;

variable is a user identifier defined by following the naming rules.



Sequential Structure - Variables and Constants Declarations

Constants: can appear as such anywhere as literals, intrinsic constants available in the Visual Basic programming environment or in other Windows applications, or as declarations in the declarative part of the program.

- can be defined as a declaration by using the syntax:

`Const constantName[As data_type]=expression[,...]`

where:

data_type can be one of Byte, Boolean, Integer, Long, Single, Double, Currency, Decimal, String, Date, [*user_defined*], Variant, Object;

constantName is an user identifier defined by following the naming rules;

expression an expression evaluated to an agreed data type whose evaluation is considered the default value for the constant.





Sequential Structure - Input/Output operations by using Inbox and MsgBox functions

Input/Output Operations by Using Inbox and MsgBox Functions

InputBox. Displays a prompt in a dialog box, waits for the user to input text or click a button, and returns a string containing the contents of the text box.

Syntax:

InputBox(prompt[,title][,default][,xpos][,ypos][,helpfile,context])

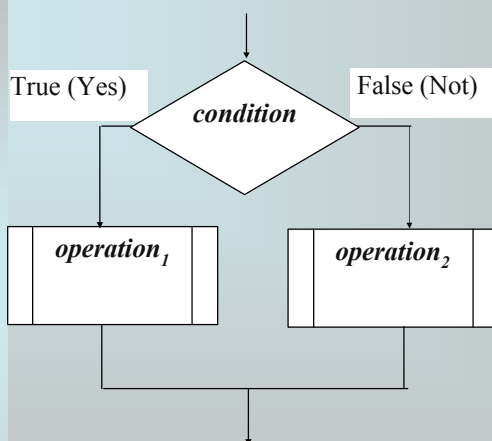
MsgBox. Displays a message in a dialog box, waits for the user to click a button, and returns an Integer indicating which button the user clicked.

Syntax:

MsgBox(prompt[,buttons][,title][,helpfile,context])

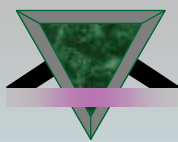


Decision (Alternative) Structure



If condition Then
 operation₁
Else
 operation₂
End If





Decision (Alternative) Structure

*The logical condition *<condition>* is a logical expression that will be evaluated either to True or either to False. The logical conditions can be simple or complex logical conditions.

A simple logical condition has the general syntax:

<variable> [<relation_operator> <variable>]

or

<variable> [<relation_operator> <constant>]



Decision (Alternative) Structure

*It is possible do not have a specific operation on the two branches :

If *condition* Then *statement*

*** Nested If**

***If ... Elself ...**

If *condition1*

Then

sequence1

Elself *condition2* Then

sequence2

.

.

.

Else

.

.

.

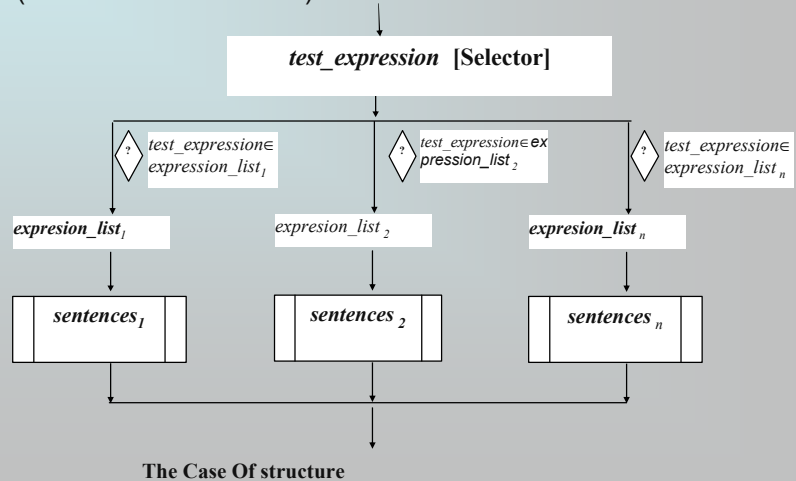
End If





Decision (Alternative) Structure

Case of. Executes one of several groups of statements depending on the value of an expression (called selector). The case structure (and statement) is especially used when selection is based on the value of a single variable or a simple expression (called the case selector).



Decision (Alternative) Structure

```

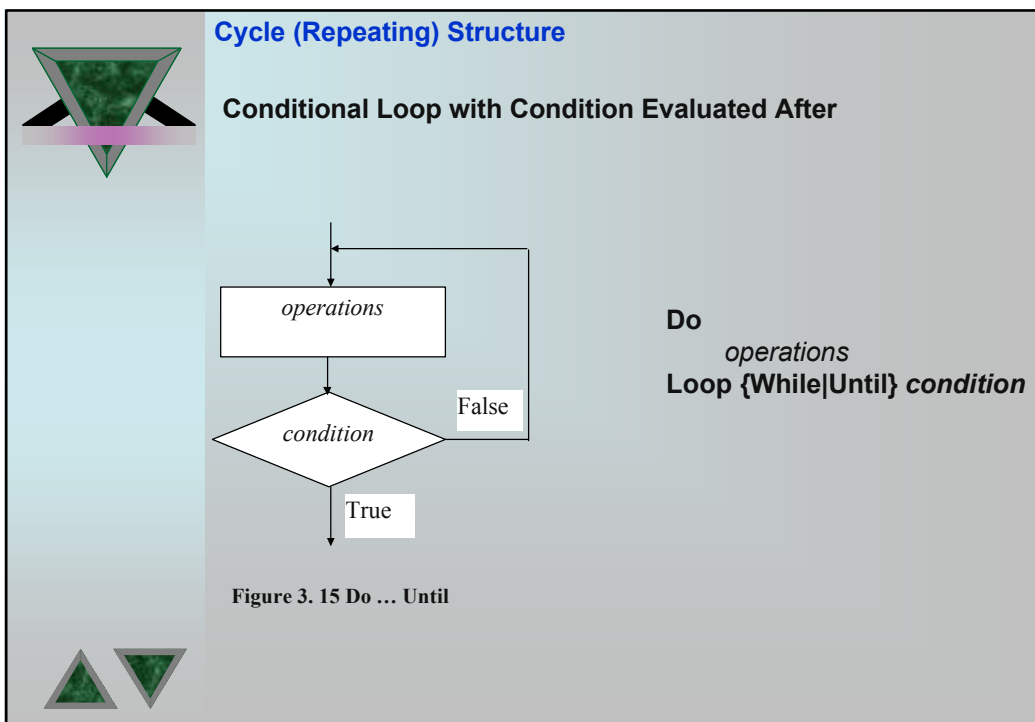
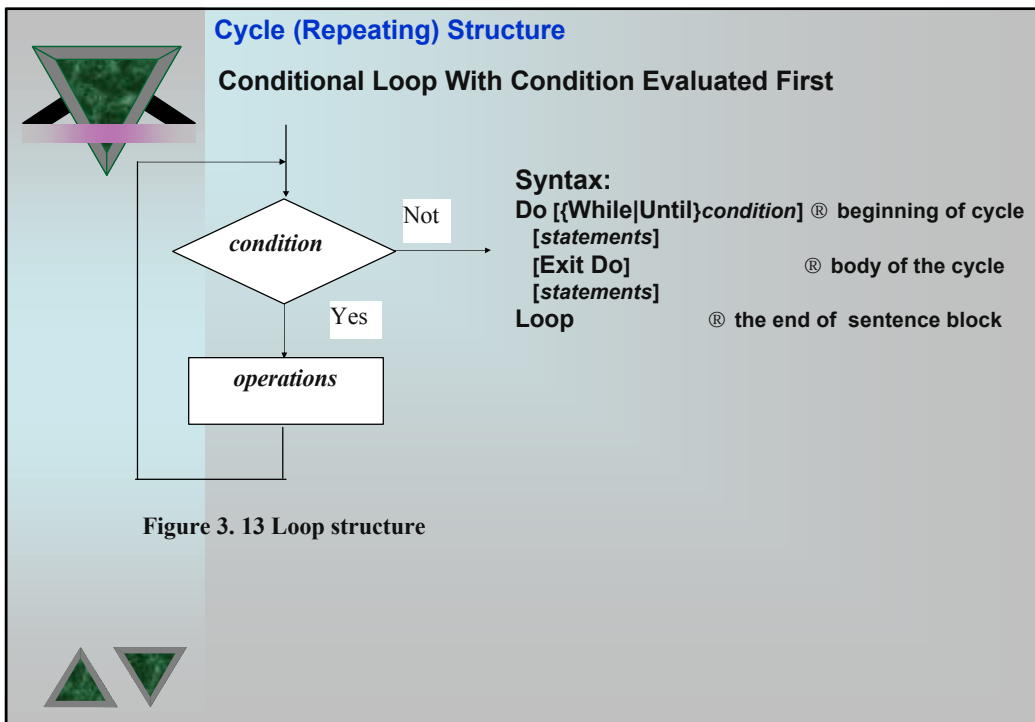
Select Case test_expression
[Case expression_list1
[sentences1]]
[Case expression_list 2
[sentences 2]]
.
.
.
[Case Else
[sentences n]]
End Select
  
```

Second Syntax:

Choose(index,choice_1[,choice_2,...[,choice_n]])

The return is Null if $n < index < 1$.





Cycle (Repeating) Structure



The commands Loop and Exit Do are used to do:

- Loop – an unconditional jump (or branch) to the beginning of the associated cycle (the evaluation of the condition);
- Exit Do – an unconditional ending of the cycle (a jump to the next sentence defined under the loop that ends the body of the cycle).

The block of commands between Do... and Loop will be executed while/until the conditional expression “condition” evaluates to True.

Do...Until work as:

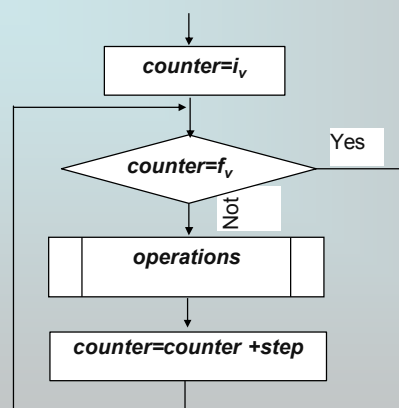
- 1) execute statements;
- 2) evaluate the condition Loop or Exit



Cycle (Repeating) Structure

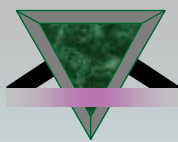


Counter Loop



For counter = *vi* **To** *fv* [Step *s*]
 operations
 [Exit For]
Next [counter]





Cycle (Repeating) Structure

The execution of For (VB) sentence follows the scenario:

- The value vi is assigned to the variable *counter*;
- The value of variable *counter* is compared with the end value fv (If the value for step is negative is checked if $counter < fv$);
- The operations are executed;
- The value of variable *counter* is incremented with the value *step* (1 if *step* not specified);
- Repeat the steps from 2 to 5.



Cycle (Repeating) Structure

The **For** instructions can be nested following the model:

```
For i...  
    sentences  
    For j ...  
        sentences  
        ...  
        For k ...  
            sentences  
        Next k  
        ...  
    Next j  
    ...  
Next i
```

Each **For i** associated with the line containing the **Next** sentence followed by the variable designated as counter.





Contents

Chapter 3 THE REPRESENTATION OF PROCESSING ALGORITHMS

3.1. Concepts

3.1.1. The Stages of Solving a Problem by Means of Computer

The General Steps for Preparing a Program

Executing a Program

3.1.2. The Description of Algorithms by Means of Logical Diagrams (Flowcharts)

Process, Program and Document Flowcharts

The Basic Symbols Used in Drawing Program Flowcharts

Fundamental Structures Used in Algorithm Representation

3.2. Sequential Structure

3.2.1. Assignments

3.2.2. Variables and Constants Declarations

3.2.3. Input/Output Operations by Using Inbox and MsgBox Functions

InputBox ; MsgBox

3.3. Alternative Structure (Decision)


If ... Then ... Else ; Case of / Switch

3.4. Repeating Structure

Conditional Loop With Condition Evaluated First

Conditional Loop With Condition Evaluated After

3.5. Commented Samples



Bibliography

1.[Av.00]Avram Vasile - *Sisteme de calcul si operare*, volumul II, Editura Dacia Europa Nova Lugoj, 2003

2. [AvDg.97]Avram Vasile, Dodescu Gheorghe - *General Informatics*, Editura Economica, Bucuresti, 1997 (you can find another pseudocode)

3. [AvDg.03]Avram Vasile, Dodescu Gheorghe – *Informatics: Computer Hardware and Programming in Visual Basic*, Editura Economica, Bucuresti, 2003, chapter 3, pages 83-126; chapter 4, pages 177-222

4. [AASA.02]V.Avram V, C.G.Apostol, T.Surcel, D.Avram – *Birotica Profesionala*, Editura Tribuna Economica, 2002, chapter 3, 103-189

5. [SMAA] T.Surcel, R.Marsanu, V.Avram, D.Avram – *Medii de programare pentru gestiunea bazelor de date*, Editura Tribuna Economica, 2004, chapter 4, pages 213-326

