Course Title: Visual Programming
Mid Term Examination
Fall Semester- 2022
Registration no :BS(IT)/3-19/M01002
Student's Name:Zeeshan Rasheed
Father's Name Rasheed khan
Faculty of Computer Science
Instructor Name: Sir Waheed Ahmed

Qno1: (a) Briefly Explain the difference between the following terms
ANSWER: 1.Structured Programming

Kenneth Leroy Busbee and Dave Braunschweig

Overview

Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines in contrast to using simple tests and jumps such as the go to statement, which can lead to "spaghetti code" that is potentially difficult to follow and maintain.

Discussion

One of the most important concepts of programming is the ability to control a program so that different lines of code are executed or that some lines of code are executed many times. The mechanisms that allow us to control the flow of execution are called control structures. Flowcharting is a method of documenting (charting) the flow (or paths) that a program would execute. There are three main categories of control structures:

Sequence - Very boring. Simply do one instruction then the next and the next. Just do them in a given sequence or in the order listed. Most lines of code are this.

Selection - This is where you select or choose between two or more flows. The choice is decided by asking some sort of question. The answer determines the path (or which lines of code) will be executed.

Iteration - Also known as repetition, it allows some code (one to many lines) to be executed (or repeated) several times. The code might not be executed at all (repeat it zero times), executed a fixed number of times or executed indefinitely until some condition has been met. Also known as looping because the flowcharting shows the flow looping back to repeat the task.
A fourth category describes unstructured code.
Branching - An uncontrolled structure that allows the flow of execution to jump to a different part of the program. This category is rarely used in modular structured programming.
All high-level programming languages have control structures. All languages have the first three categories of control structures (sequence, selection, and iteration). Most have if then else structure (which belongs to the selection category) and the while structure (which belongs to the iteration category). After these two basic structures, there are usually language variations.
The concept of structured programming started in the late 1960's with an article by Edsger Dijkstra. He proposed a "go to less" method of planning programming logic that eliminated the need for the branching category of control structures. The topic was debated for about 20 years. But ultimately - "By the end of the 20th century nearly all computer scientists were convinced that it is useful to learn and apply the concepts of structured programming."

Key Terms

Branching
An uncontrolled structure that allows the flow of execution to jump to a different part of the program.

control structures
Mechanisms that allow us to control the flow of execution within a program.

iteration
A control structure that allows some lines of code to be executed many times.

selection
A control structure where the program chooses between two or more options.

sequence
A control structure where the program executes the items in the order listed.

spaghetti code
A pejorative phrase for unstructured and difficult to maintain source code.

structured programming

A method of planning programs that avoids the branching category of control structures.

2.Object oriented programming

Object Oriented Programming (OOP) approach identifies classes of objects that are closely related to the methods with which they are associated. It also covers the concepts of attribute and method inheritance. The Massachusetts Institute of Technology was the first institution to utilize terminology referring to "objects" in the sense that we use object-oriented programming today, in the late 1950s and early 1960s.

It is a method for storing data and the operations required to process that data based on the mathematical field known as abstract data types. Programming could advance to a more abstract level thanks to OOP. Nearly all developers employ the core programming paradigm known as object-oriented programming at some point in their careers.

The well-known programming paradigm, OOP, is taught as the norm for most of a programmer's educational career. OOP is based on the idea of classes and objects. It organizes a computer program into basic, reusable blueprints of code or "classes." These classes are then used and reused to create new and unique objects with similar functions. This paradigm represents a system that interacts with actual items in real life - such as the user.

Different parts of it perform actions on real-world items, creating actual interactions between people and machines. The strategy is advantageous for collaborative development when projects are divided into groups due to the organization of object-oriented software. Code reuse, scalability, and efficiency are other advantages of OOP.

The first stage in OOP is to gather all the objects that a programmer wishes to work with and determine their relationships, a process known as data modeling. Data and functions are combined to create an object from the data structure. Programmers can also establish connections between several objects. Objects can, for instance, acquire traits from other objects. A human is a straightforward illustration of an object.

You would logically anticipate that a person would have a name. This would be regarded as being in the person's possession. Another thing you could expect from someone is their ability to do, like walk or drive. One might view this as one

of the person's methods. Objects serve as the framework for object-oriented programming code.

Once your objects are in place, you may use their interactions to achieve the desired outcome. Consider the possibility of a show where someone gets in a car and drives it from point A to point B. Beginning with the objects like a person or a vehicle is how you would describe them.

The use of ways is one example of this: a person can drive a car, and a car can be driven. For the individual to drive, you must gather your items so that they are all in one place. When the object is identified, it is assigned a class of objects that describes the type of data it contains and sequences logic that could modify the data in any way. A method is any particular logic sequence. With clearly specified interfaces known as messages, objects may communicate.

See More: What Is Jenkins? Working, Uses, Pipelines, and Features

Key Concepts of OOP

To understand and use object-oriented programming, it is necessary to know the following key concepts:

1. Class

A class is the fundamental unit of C++ that paves the way for object-oriented programming. It is a user-defined data type that can be accessed and used by creating an instance of that class. It has its own data members and member functions. A class is comparable to an object's blueprint. Both member functions and data members are found in classes. The data members inside the class are manipulated using these member functions.

2. Object

At the point of creation of a class, the description is the first object to be defined. An instance of a class exists in an object. Notably, the system does not allocate any memory space when a class is specified, but it's allocated when it is instantiated, i.e., when an object is formed. Real-world things have state and behavior in common, a pair of features. An object conceals its behavior through methods and keeps its information in attributes.

3. Syntax

The principles that specify how a language is structured are known as syntax. In programming languages (rather than natural languages like English), syntax is the set of rules that define and guide how words, punctuation, and symbols are

organized in a programming language. Without syntax, it is almost difficult to comprehend the semantics or meaning of a language. A compiler or interpreter won't be able to understand the code if the syntax of a language is not adhered to.

4. Encapsulation

Encapsulation is the process of grouping functions and data into a single entity. To access these data members, the member function's scope must be set to "public," while the data members' scope must be set to "private." According to this theory, an item contains all important information; only a small subset is made available to the outside world. Each object has a private class that contains its implementation and state.

5. Polymorphism

Multiple classes can use the same method name using polymorphism, which also involves redefining methods for derived classes. Compile-time polymorphism and run-time polymorphism are the two different types of polymorphism. In addition to having several forms, objects are made to have shared behaviors. To avoid writing duplicate code, the software will determine which usage or meaning is required for each time an object from a parent class is used.

6. Inheritance

In its broadest sense, inheritance refers to the process of gaining properties. One object in OOP inherits the properties of another. Developers can reuse common functionality while retaining a distinct hierarchy by assigning relationships and subclasses between items. This characteristic of OOP speeds up development and provides more accuracy by requiring a more in-depth investigation of the data. The parent-child relationship is symbolized via inheritance.

7. Abstraction

One of the OOP concepts in Java is abstraction, which is the act of representing key features without including supporting information. It is a method for developing a brand-new data type appropriate for a particular application. It avoids providing extraneous or pointless facts and only displays the precise portion the user has requested. It is crucial since it prevents you from performing the same task more than once.

8. Coupling

Coupling describes the degree to which one software element is connected to another. Software elements can be a class, package, component, subsystem, or

system. It denotes the level of familiarity one object or class has with another. This means that if one class changes its attributes, the dependent changes in the other will also change. The magnitude of interdependence between the two classes will determine how these changes occur.

9. Cohesion

A class's cohesion is determined by how closely and meaningfully coupled to one another its methods and properties are, as well as by how intently they are focused on carrying out a single, clearly defined goal for the system. This is a measure of how narrowly focused a class's responsibilities are. Because their methods and properties don't relate to one another logically, low cohesive classes are challenging to maintain.

10. Association

An association is a relationship between two distinct classes that are established with the aid of their objects. One-to-one, one-to-many, many-to-one, and many-to-many associations are all possible. An association is a connection between two things. The diversity between objects is defined by one of Java's OOP concepts. There is no owner in this OOP concept, and each object has a distinct lifecycle.

11. Aggregation

In this method, each object has a distinct lifecycle. Ownership, however, prevents the child object from being a part of another parent object. Java aggregation depicts the link between an object that contains other objects and is a weak association. This illustrates the connection between a component and a whole, where a part can exist without a whole. A unique type of semantically weak link called an aggregation occurs when unrelated things are combined.

12. Composition

Composition is an association that depicts a relationship between a part and a whole in which a part cannot exist without a whole. Aggregation can take a variety of forms, including composition. Since child objects lack a lifecycle, they all automatically disappear when the parent object does. One object cannot exist without the other in any composition between two entities. As a result, both entities depend on one another in their composition.

13. Modularity

Modular design refers to the division of a system into many functional pieces (referred to as modules) that can be combined to create a more extensive application. Modularity and encapsulation are inextricably related. When

mapping encapsulated abstractions into actual, physical modules, high cohesion within the modules and limited inter-module interaction or coupling can be seen as the definition of modularity.

14. Constructors and methods

A constructor is a specific kind of subroutine called to create an object. It sets up the new object for use and frequently accepts arguments from the constructor to set up necessary member variables. In OOP, a method is a procedure connected to a message and an object. An object's state data and behavior make up its interface, which describes how any of its numerous consumers may use it. A method is a consumer-parameterized object activity.

Advantages of OOP

Despite the rise of various programming models, OOP remains popular in DevOps. This is due to the following advantages it provides:

1. Enables code reusability

The idea of inheritance is one of the critical concepts offered by object-oriented programming. A class's attributes can be passed down through inheritance, eliminating the need for duplication of effort. Doing this prevents the problems associated with repeatedly writing the same code.

Thanks to introducing the idea of classes, the code section can be used as many times as necessary in the program. A child class that uses the inheritance method inherits the parent class's fields and methods. One can readily alter the parent class's available methods and values.

2. Increases productivity in software development

We can create programs from pre-written, interconnected modules rather than having to start from scratch, which would save time and increase productivity. Thanks to the OOP language, we can break the software into manageable, discrete problems. Because it allows for the division of labor in the creation of object-based programs, object-oriented programming is modular.

It is also extendable, as you may add new characteristics and actions to objects. One can utilize objects in several applications. Object-oriented programming increases software development productivity, compared to conventional procedure-based programming techniques, due to modularity, extensibility, and reusability.

3. Makes troubleshooting simpler

When object-oriented programming is used, troubleshooting is made simpler since the user knows where to look in the code to find the source of the problem. Since the error will indicate where the issue is, there is no need to inspect additional code areas. All objects in object-oriented programming (OOP) are self-constrained, which is one benefit of employing encapsulation. DevOps engineers and developers gain a lot of advantages from this multimodal behavior because they may now work on several projects at once with the benefit of avoiding code duplication.

4. Reinforces security

To maintain application security and provide vital data for viewing, we are filtering out limited data through data hiding and abstraction mechanisms. The concept of data abstraction in OOPS allows only a small amount of data to be displayed to the user, which is one of OOP's strong points.

When only the necessary info is accessible, the rest is not. As a result, it makes security maintenance possible. Another set of OOP's advantages in Java's idea of abstraction is used to conceal complexity from other users and display the element's information per the requirements.

5. Simplifies code maintenance

Object-oriented software is simpler to maintain in terms of code. Because of the design's modularity, one can upgrade a portion of the system in the event of problems without calling for significant adjustments. Additionally, you can modify already-existing objects to create new ones.

Any programming language would benefit from having this capability; it prevents users from having to redo work in a variety of ways. Maintaining and updating the current codes by adding new changes is always simple and time-saving. Since one can produce new objects with just minor variations from old ones, it is simple to maintain and modify current code.

6. Prevents the repetition of data

Redundant data refers to data that has been duplicated. As a result, the same information is repeated. The redundancy of the data is seen as a benefit in object-oriented programming. For instance, the user would like the capability

comparable to that of practically all classes.
In such circumstances, the user can construct classes with comparable functionality and inherit them when necessary. A significant benefit of OOP is the redundancy of data. Users who want a comparable feature in numerous classes can write standard class definitions for those features and inherit them.

7. Results in flexible code

Polymorphism is the idea that allows for flexibility. The following advantages of polymorphism for developers are extensibility and simplicity. One advantage of OOP is polymorphism, which allows a piece of code to exist in more than one version. For instance, you might act differently if the setting or environment changes.
Let us look at a simple example. In a market, a person will act like a customer; in a school, a person will act like a student; and in a home, a person will act like a son or daughter. Here, the same person exhibits various behaviors depending on the environment.

8. Addresses issues early on

Another benefit of object-oriented programming is that it may effectively solve problems by being divided into smaller components. It becomes good programming practice to deconstruct a complex issue into simpler parts or components. Given this information, OOPS uses a feature that divides the program code into smaller, more manageable chunks developed one at a time. Once the issue has been disassembled, you can put the individual pieces to use again to address additional problems. Additionally, *one might use the modules with the same interface and implementation details to replace the more minor codes.

9. Provides design advantages

A significant development in software engineering has been object-oriented development. Among other things, it promises to shorten development duration and give firms a competitive advantage. The design benefit that users will experience from OOPs is the ease with which they can design and fix things and the reduction of hazards, if any.
Here, object-oriented programs require a lengthy and thorough design phase from the designers, which produces better designs with fewer faults. It is simpler to program all the non-OOPs independently after a certain point when the

program has hit some fundamental constraints.

10. Lowers development costs

Using an object-oriented approach does make it possible to cut back on some of the direct costs involved with systems, including maintenance and development. Reusing software also reduces the price of development. In most cases, more time and effort are spent on object-oriented analysis and design, reducing the overall development cost.

The general cost of the improvement is reduced since more effort is typically put into the article-specific assessment and plan. The development cost is generally reduced since more time and effort are usually spent on object-oriented analysis and design.

3.Visual programming

Just how big the gap is between visual programming and traditional programming depends on the visual programming tool. At one extreme, the tool shields the programmer almost entirely from the gaping space between human thinking and computers shuffling bits around memory.

Here's an example. To create a to-do list with a visual programming tool, the programmer draws out the flow of the app. The resulting flowchart describes screens, user interactions, and the data at each stage. The tool then turns that into software.

Developers know that text-based programming languages focus entirely on the implementation: it's all about the precise steps the computer must take to create the experience we want to give the user. Sure, higher-level languages and modern frameworks give us convenient shortcuts. But, the developer's job is to translate human needs into processes that fit the computer's limited abilities.

Other visual coding tools follow the same processes and paradigms as text-based programming. Imagine drawing a class and its relationship with the objects you instantiate rather than typing it all out into a text editor.

In this article

Visual Programming vs. Traditional Programming

A Brief History of Visual Programming Software

The State of Visual Programming Today

So, Is There Room for Visual Programming Languages Today?

Next Generation of Visual Programming: Delivering on the Promise

Experience the power of low-code.

Start Free

A Brief History of Visual Programming Software

What is visual-programming 01

Although history seems to show it, it's not fair to say that visual programming in the 1990s was confined to game creation kits, multimedia tools, and databases. Rational Software (which was acquired by IBM in 2003) had been building a non-GUI Ada IDE since the mid-1980s. In addition, they also turned their hand to define the software development process.

Work on their Rational Unified Process and related efforts eventually led to the Unified Modelling Language that had the potential to document every last part of a system without ever writing a line of code. It looked like visual programming but without producing executable software.

UML provided a standardized and comprehensive language for describing object-oriented systems. However, UML-fever struck some architects. The co-author of The Pragmatic Programmer, Andy Hunt, tells the story of a software project where an architect spent two years creating UML diagrams before even a line of code was written.

Just as agile was gaining momentum, UML seemed to enable all the worst aspects of the old ways of building software: too much planning and too little implementation. Executable UML was an attempt to add that missing piece-the executable software. Several implementations surfaced but without making too much of an impact on a world that was rapidly switching its focus to PHP, Ruby on Rails, and other dynamic scripting languages.

Interestingly, one form of executable UML that did stick around also came out of Rational Software. Rational Rose is a suite of tools for creating software using UML and generating executable code in a target language, such as C++ or Java.

What is visual-programming-02

Case view in Rational Rose. Image source: Assignment Help.

The State of Visual Programming Today

Based on what history shows us, is visual programming dead? Visual programming enthusiasts will tell you that it's far from dead. Ask them, "what is visual programming?" and first, they'll name an obscure domain-specific tool. Then, they'll tell you that tool is proof that it's alive and kicking.

A quick search on Google will show you not just the tool they've mentioned but also the highly specialized world in which it exists.

Without a doubt, visual programming has its role, whether that's programming synthesizers or giving UML enthusiasts a sense of accomplishment. For general purpose software, though, the world is just too complex to model purely visually.

When your "code" looks like a CPU circuit diagram, it's perhaps time to rethink the suitability of visual programming to the task.

Similarly, visual programming software tends to be limited by the creator's imagination in a way that doesn't hamper general-purpose textual programming languages.

And yet tools like Visual Basic, Delphi, and their descendants have shown us that building software visually can be enormously efficient; it's just that there's a pragmatic trade-off where, sometimes, hand-written code is the right solution.

Those early days of programming were hard, that's for sure. But one person could understand and be an expert in everything necessary to create that software. If you're old enough, think back to software titles of the 1980s. It was common for a single programmer to become a brand in their own right.

Sid Meier, Mitch Kapor, and Jeff Minter gained some level of fame by creating well-known applications or games single-handedly or, at most, with one other collaborator. Back then, software and hardware upgrade cycles took years. Today, we joke that there's a new JavaScript library every day. Yet, there's some truth in the idea that modern software development moves at a pace that many of us can't keep up with.

Today, software is largely built by teams of specialists. Whereas early developers did everything themselves, a modern software development team might have one person whose only job is to look after the CI tool. Developers spend entire careers focused on one framework or platform. iOS developers are iOS developers, not

mobile developers. Once or twice a decade, perhaps, a web developer might switch their preferred framework. Very few people are manually writing assembly language professionally.

It's not just that the scope of software has changed. To some extent, developers themselves have changed, too. Software engineering is just another career these days. In decades gone by, it was a passion held by a few people who had the dedication to learn an entirely new system so they could write the Atari ST port, for example, of their successful Amiga game. But that's understandable: computing is no longer niche.

Today, we have a world where software development is made up of increasingly complex parts and where developers are ordinary people with extraordinary specializations. That complexity and specialization are badly suited to the pure visual programming of those early tools, but it also makes it increasingly hard to build rounded software engineering teams.

Where pure visual programming environments have failed, a whole cache of similar tools takes the best of visual programming and combines it with text-based coding. Whereas visual programming was "no-code" these new tools are low-code.

These tools let developers create software visually by drawing interaction flows, UIs, and the relationships between objects, but supplementing it with hand-written code where that's the better thing to do.

This pragmatic mix of visual and text-based programming is well suited to the needs of modern software development. Low-code platforms reduce the complexity of software development and return us to a world where a single developer can create rich and complex systems without learning all the underlying technologies.

Qno1(b): Write a VB program that should take 10 number from user. Your program should find and display the maximum number?

ANSWER:

```
Dim input From Console As String
Dim converted Int As Integer
Console Write Line("Please Enter a number using numerals (2, 5, 100, etc.)")
```

```
Input From Console = Console Read Line
Try
Converted Int = C Int(input From Console)
Catch ex As Exception

   Message Box Show ("You must enter a number in numeric from (2, 5, 100, etc.)", "Error")

   Environment Exit (0)

End Try
Console Write Line ("The first 10 multiples of " + converted Int. To String () + " are;")

 For index As Integer = 1 To 10

   Console Write Line (converted Int.To String () + " * " + index.ToString() + " = " + (index * converted Int).To String ())
Next
Console Read Key ().
```

Qno2(a): What are the different looping construct available in visual basic with example?

ANSWER:

Loop statements allow you to execute one or more lines of code repetitively. Visual Basic supports the following loop statements:

Do..Loop : The Do..Loop executes a block of statements for as long as a condition is True. Visual Basic evaluates an expression, and if it's True, the statements are executed. If the expression is False, the program continues and the statement following the loop is executed.

There are two variations of the Do..Loop statement. A loop can be executed either while the condition is True or until the condition becomes True. These two variations use the keywords.While and Until to specify how long the statements are executed. To execute a block of statements while a condition is true, use the following syntax :

```
Do While condition
statement block
```

Loop

To execute a block of statements until the condition becomes True, use the following syntax :

```
Do Until condition
statement block
Loop
```

Another variation of the Do loop executes the statements first and evaluates the condition after each execution. This Do.Loop has the following syntax :

```
Do
statements

Loop While condition
```

Or

```
Do

statements
```

Loop Until condition

For...Next : The For...Next loop is one of the oldest loop structures in programming languages. Unlike the Do..loop, the For ...Next loop requires that you know how many times the statements in the loop will be executed. The For.Next loop uses a variable(it's called the loop's counter) that increases or decreases in value during each repetition of the loop. The For...Next loop has the following syntax :

```
For counter=start to end[Step increment]
statements
Next [counter]
```

The keywords in the square brackets are optional. The arguments counter, start, end and increment are all numeric. The loop is executed as many times as required for the counter to reach the end value.

While ...Wend : The while..wend loop executes a block of statements while a condition is True. The while...wend loop has the following syntax :

While condition
statement - block

Wend

If condition is true, all statements are executed and when the Wend statement is reached, control is returned to the While statement which evaluates condition again. If condition is still True, the process is repeated. If condition is False, the program resumes with the statement following the Wend statement.

Qno2(b): Write a program that contains a label, a textbox and a button. The user enters some text in the textbox and clicks the button. The text in the textbox also appear on the label as its caption?

ANSWER:

Prerequisites

You need the following components to complete this walkthrough:
An edition of Visual Studio that includes the Microsoft Office developer tools. For more information, see Configure a computer to develop Office solutions.
Microsoft Word

Create the project

The first step is to create a Word Document project.

To create a new project

Create a Word Document project with the name My Word Button. In the wizard, select Create a new document.

For more information, see How to: Create Office projects in Visual Studio.

Visual Studio opens the new Word document in the designer and adds the My Word Button project to Solution Explorer.

Add controls to the Word document

The user interface controls consist of a button and a text box on the Word document.

To add a button and a text box

Verify that the document is open in the Visual Studio designer.

From the Common Controls tab of the Toolbox, drag a TextBox control to the document.

Note

In Word, controls are dropped in-line with text by default. You can modify the way controls and shape objects are inserted by changing the default on the Edit tab of the Options dialog box in Word.

On the View menu, select Properties Window.

Find TextBox1 in the Properties window drop-down box and change the Name property of the text box to displayText.

Drag a Button control to the document and change the following properties.

Property Value

Name insert Text

Text Insert Text

Now you can write the code that will run when the button is clicked.

Populate the text box when the button is clicked

Every time the user selects the button, Hello World! is added to the text box.

To write to the text box when the button is clicked

In Solution Explorer, right-click This Document, and then select View Code on the shortcut menu.

Add the following code to the Click event handler of the button.

C#

VB

C#

Copy

```
private void insert Text _Click(object sender, Event Args e)
{
```

```
    This display Text Text += "Hello World!";
}
```

In C#, you must add an event handler for the button to the Startup event. For information about creating event handlers, see How to: Create event handlers in Office projects.

C#

Copy

```
This insert Text Click += new Event Handler (insert Text_Click);
```

Test the application

You can now test your document to make sure that the message Hello World! appears in the text box when you select the button.

To test your document

Press F5 to run your project.

Select the button.

Confirm that Hello World! appears in the text box.

Next steps

This walkthrough shows the basics of using buttons and text boxes on Word documents. Here are some tasks that might come next:

Using a combo box to change formatting. For more information, see Walkthrough: Change document formatting using CheckBox controls.

Using radio buttons to select chart styles. For more information, see Walkthrough: Update a chart in a document using radio buttons.

Qno3(a): What are the difference between logical and syntax error with example?

ANSWER:

The problem considered and a solution

Let's suppose we have an array of int types and we want to print them in sequence to the screen in the following way: "output: 1 2 3 4 5".

Solution:

Now let's understand what I could have made wrong.

Syntax errors

Look at the following code:

The two files posted above are equal? Look at them.

They are not equal. In the second one I wrote "i < array Ex.lengt;". What is "lengt"? I don't know, you probably don't know, the compiler surely doesn't know. Here is the point! If you try to compile the second file the process will not succeed. The compiler will protest with a ®error: cannot find symbol¯ message. This is a syntax error!

What is a syntax error?

An syntax error occurs when during compilation the compiler finds something in your code it doesn't know what it means, or it not sure how to translate what you wrote. The compiler will always advise you about syntax errors, even though sometimes it could be wrong about the line (or lines) where the error occurred.

Look at the following example:

Where is the error?
I wrote "System.out.println numbers);". What is "printlnnumbers);"?? It is quite easy for us to understand that I forgot to write the "(" bracket. The compiler didn't get it and it tells me:

I didn't write a statement -> quite right, what I wrote it is not a valid statement.

";" expected -> False, I wrote the semicolon at the end!
So pay attention. The compiler advises you but sometimes it could miss the real problem.

Logical errors

Look at the following code:

What changed? Try to figure it out.

In the for loop this time I started from 1, I wrote "int i = 1;". Is it an error? It depends. On what? On what we wanted to do. I mean, it is not a syntax error (try to convince you), so the compiler wouldn't protest. The program would be finally

executable but the output wouldn't be the one we expected. Why? We initialized the variable "i" from value "1" so we will print the values of the array from the second element (arrayEx[1] is the second element!). This is is a logical error!

What are logical errors?

They are errors which occurs when the program executes but doesn't do what we expected. They are the most difficult to identify because the compiler doesn't do nothing for you (the program has already been compiled).

Some logical errors are so serious that exceptions are thrown (you can throw exceptions in your program too).

Example:

In this code in the for loop, the condition changed. I wrote "i<=arrayEx.length". I added the "=" operator! So? It means also the value of "i" equal to "arrayEx.length" will be used as index for the array, but that index doesn't exist in that array. An "Array Index Out Of Bounds Exception" will be thrown.

Conclusion

I hope I helped you to understand better the difference between "syntax errors" and "logical errors".

If you have an argument you want me to discuss about tell me in the comments.

Do you know the difference between "argument" and "parameter" in programming? I wrote an article about it.

Difference between "Argument" and "Parameter" in programming
Two words a lot of people think are synonyms. They are not!
medium.com

Thank you for having read my article. I hope I added value to your knowledge.

Consider following me and subscribing below with your email to know when I publish the new daily articles.

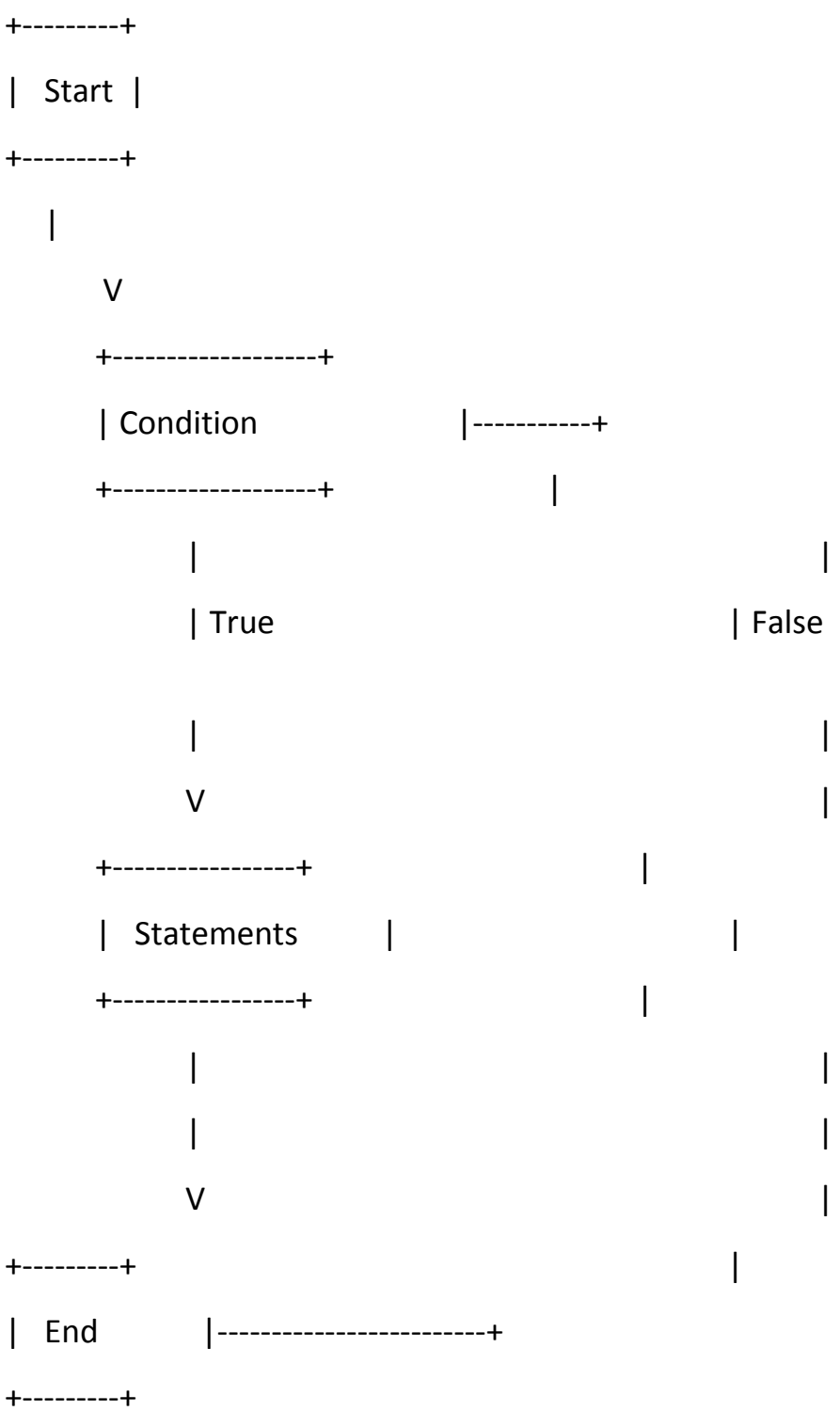
Qno3(b): Define the following:

- **Conditional and Repletion statement in VB.**

1.

As the name indicates it is quite simple. If a certain condition is satisfied, then perform some action.

Structure of If Then Statement



Linux Copy

Example:

```
Module Cosmic Tutorial
   Sub Main ()
      Dim account Balance As Integer = 0
      If (account Balance < 1000) Then
         Console. Write Line ("Close Account!")
         Console. Read Line ()
      End If
   End Sub
End Module
```

VB.Net Copy

Output:

```
Close Account!
```

In the example above, we are checking account balance in the if statement.

If the account balance comes to less than 1000, we are printing a line to close the account.

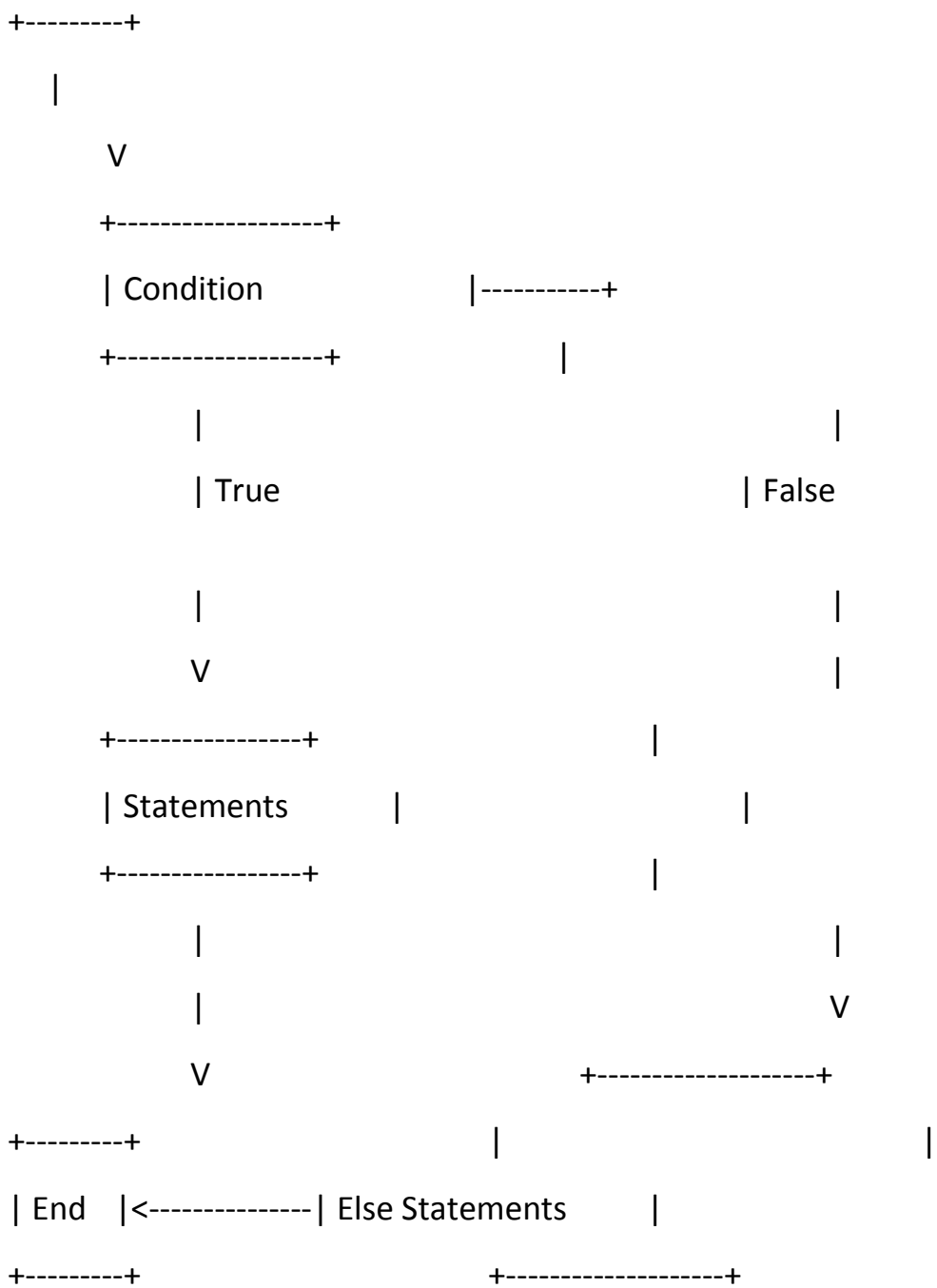
In a real world scenario this can be the case to check an account has minimum balance.

If Then Else - Back to top

An extra else statement can be added to an if condition, to execute what should happen if the if condition is not satisfied.

Structure of If Then Else Statement

```
+---------+
| Start  |
```



Linux Copy

Example:

```
Module Cosmic Tutorial
   Sub Main ()
      Dim account Balance As Integer = 1001
```

```
        If (account Balance < 1000) Then
            Console. Write Line("Close Account!")
        Else
            Console. Write Line("We love having you with us!")
        End If
        Console. Read Line()
    End Sub
End Module
```

VB.Net Copy

Output:
We love having you with us!

In the above example, if the account balance is >= 1000, we are printing a warm message.

If Then Else If Else - Back to top

If a primary if condition is not satisfied, we can add an Else If statement in between to check another condition if required.

Example:

```
Module Cosmic Tutorial
    Sub Main()
        Dim account Balance As Integer = 1000001
        If (account Balance < 1000) Then
            Console. Write Line("Close Account!")
        ElseIf (account Balance > 1000000) Then
            Console. Write Line("Please find a Europe tour cruise package in your mailbox!")
        Else
```

```
        Console. Write Line("We love having you with us!")
      End If
      Console. Read Line ()
   End Sub
End Module
```

VB.Net Copy

Output:

Please find a Europe tour cruise package in your mailbox!

In the above example, we added an else if condition.

For high valued customers who have more than a large threshold of account balance with us, we printed a separate message.

Nested If Then - Back to top

We can nest multiple If Then statements.

Be careful when using this as it can become a mess to read. You can use && or || operators to avoid it in some scenarios.

Example:

```
Module Cosmic Tutorial
   Sub Main ()
      Dim account Balance As Integer = 501
      If (account Balance < 1000) Then
         If (account Balance < 500) Then
            Console. Write Line("Close Account!")
         Else
            Console. WriteLine("Maintain a minimum balance Pal! You got 5 days time.")
```

```
        End If
    ElseIf (account Balance > 1000000) Then
        Console. Write Line("Please find a Europe tour cruise package in your mailbox!")
    Else
        Console. Write Line("We love having you with us!")
    End If
    Console. Read Line()
  End Sub
End Module
```

VB.Net Copy

Output:

Maintain a minimum balance Pal! You got 5days time.

In the above example, we added a nested if statement where we are further checking if account balance is less than 500.

If it is not, then we are giving a friendly and warm warning message.

Switch Statement - Back to top

A switch statement takes an expression and evaluates it.

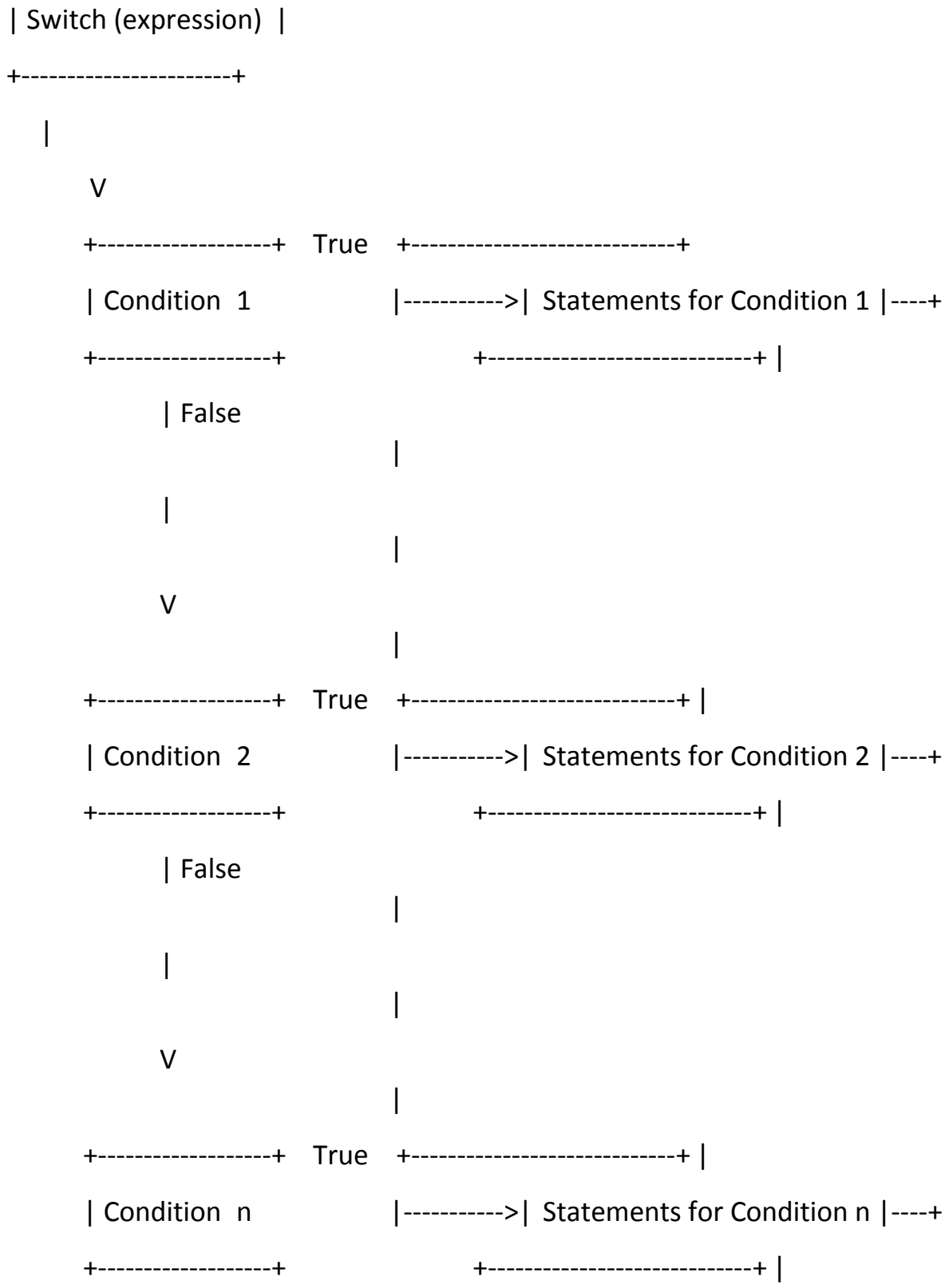
for each result of the evaluation, it executes statements. This is known as a case statement.

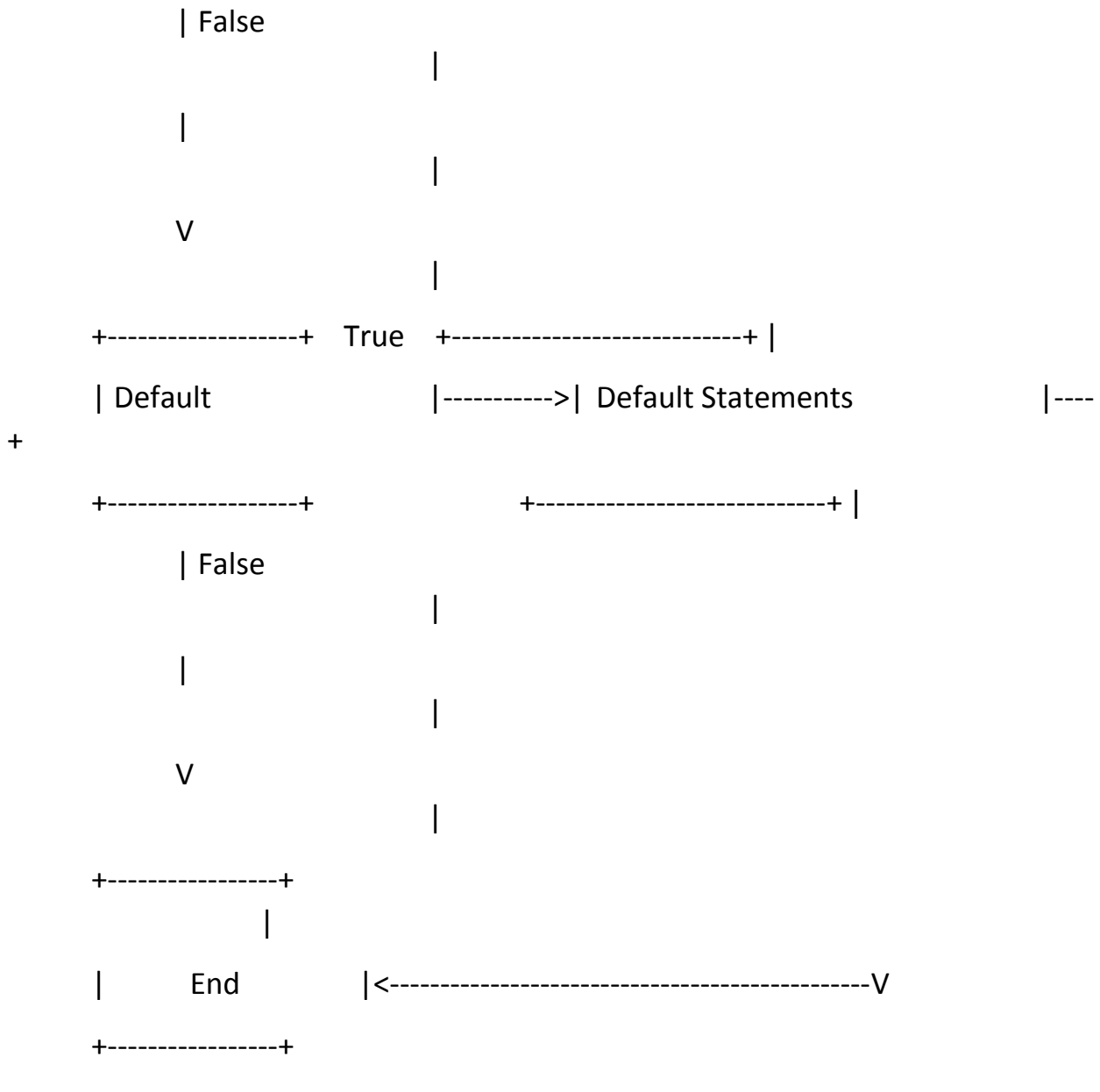
break statement must be present at the end of each case.

default case is executed if any of the case is not satisfied.

Structure of Switch Statement

```
+----------------------+
```


| Switch (expression) |
+----------+
|
V
+----------+ True +----------+
| Condition 1 |----------->| Statements for Condition 1 |----+
+----------+ +----------+ |
| False
|
|
|
V
|
+----------+ True +----------+ |
| Condition 2 |----------->| Statements for Condition 2 |----+
+----------+ +----------+ |
| False
|
|
|
V
|
+----------+ True +----------+ |
| Condition n |----------->| Statements for Condition n |----+
+----------+ +----------+ |



Linux Copy

Example:

```
Module Cosmic Tutorial
   Sub Main ()
      Dim num As Integer = 0
      Select Case num
         Case 1
```

```
            Console. Write Line("One")
        Case 2
            Console. Write Line("Two")
        Case 3
            Console. Write  Line("Three")
        Case 4
            Console. Write Line("Four")
        Case Else
            Console. Write Line("Other Number!")
    End Select
  End Sub
End Module
```

VB.NetCopy

Output:

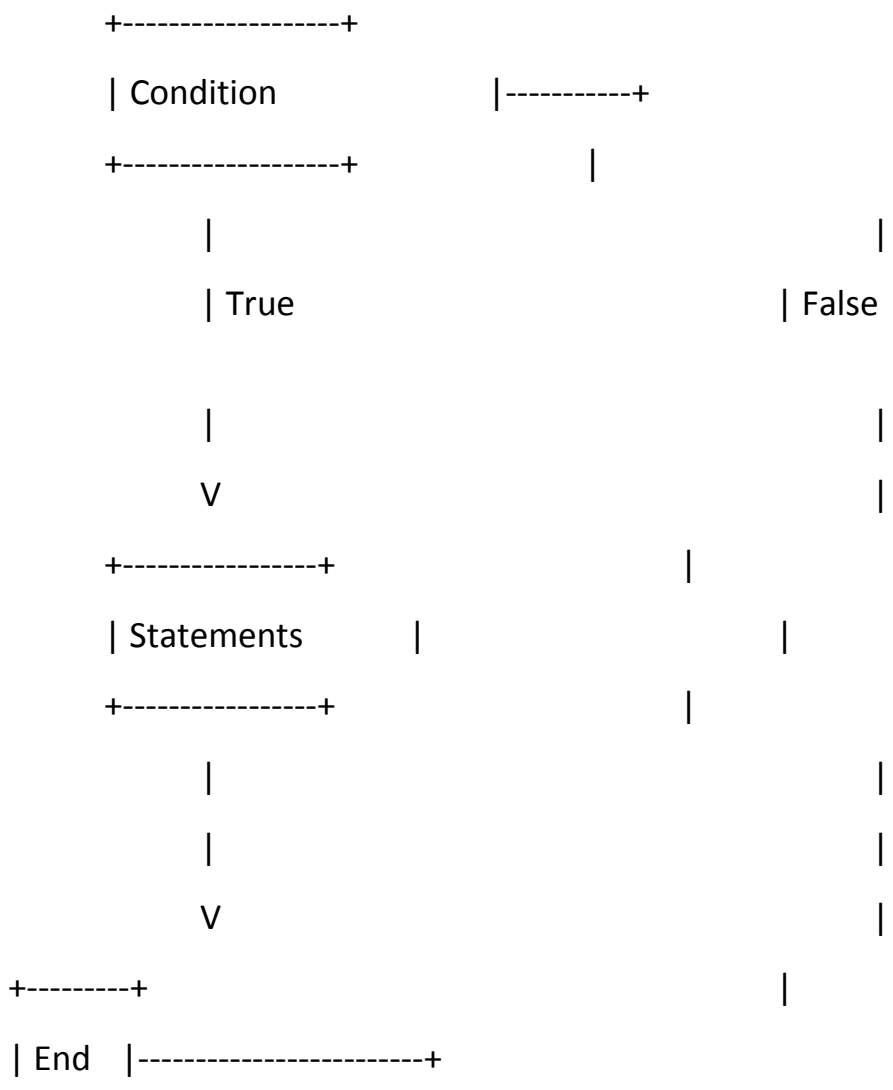
Other Number!

2.Variables in VB.

As the name indicates it is quite simple. If a certain condition is satisfied, then perform some action.

Structure of If Then Statement

```
+---------+
| Start  |
+---------+
   |
        V
```



LinuxCopy

Example:

```
Module Cosmic Tutorial
   Sub Main ()
      Dim account Balance As Integer = 0
      If (account Balance < 1000) Then
         Console. Write Line("Close Account!")
         Console. Read Line()
```

```
        End If
    End Sub
End Module
```

VB.Net Copy

Output:

```
Close Account!
```

In the example above, we are checking account balance in the if statement.

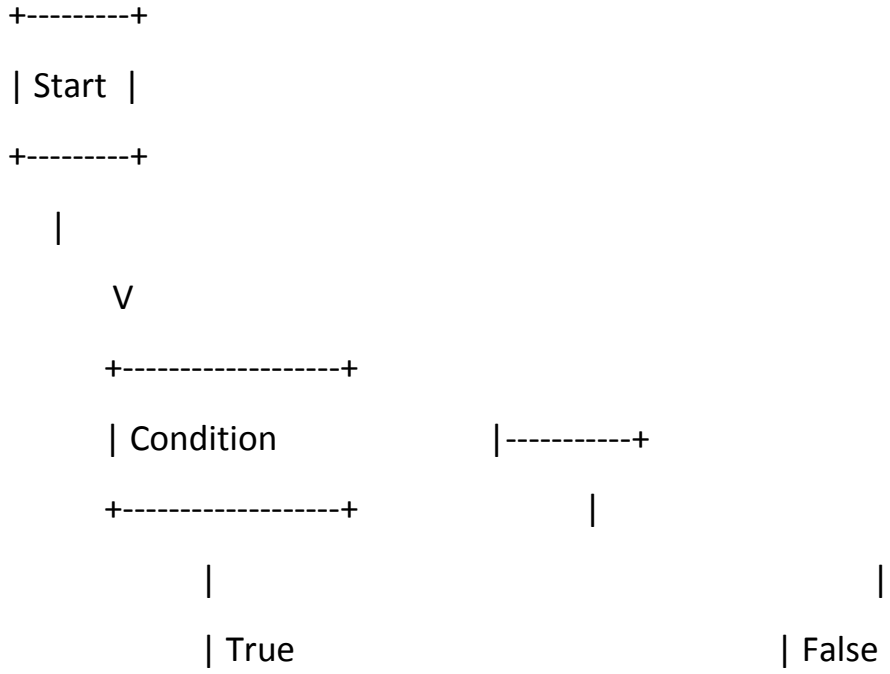
If the account balance comes to less than 1000, we are printing a line to close the account.

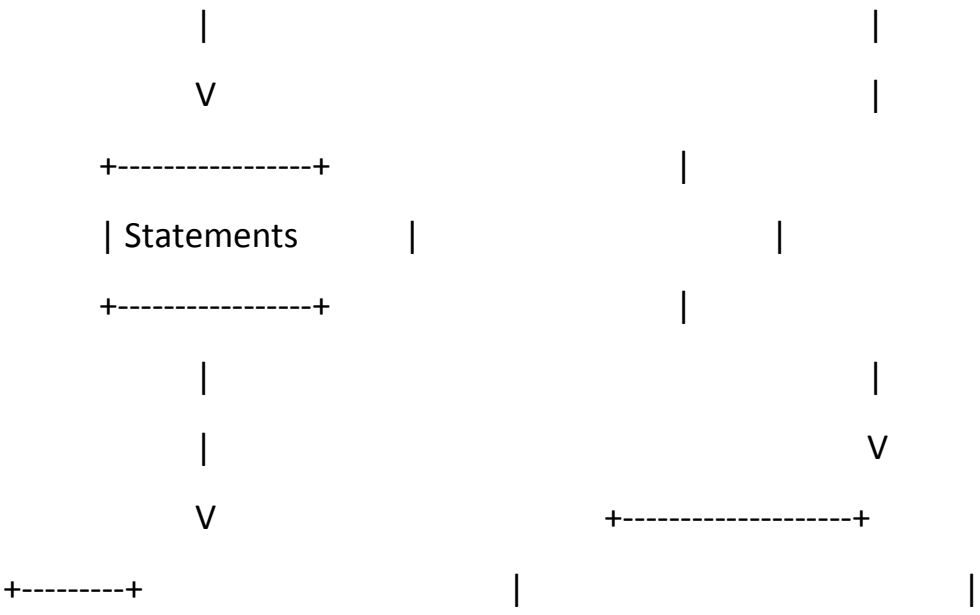
In a real world scenario this can be the case to check an account has minimum balance.

If Then Else - Back to top

An extra else statement can be added to an if condition, to execute what should happen if the if condition is not satisfied.

Structure of If Then Else Statement





Linux Copy

Example:

```
Module Cosmic Tutorial
   Sub Main ()
      Dim account Balance As Integer = 1001
      If (account  Balance < 1000) Then
         Console Write Line("Close Account!")
      Else
         Console. Write Line("We love having you with us!")
      End If
      Console. Read Line()
   End Sub
End Module
```

VB.Net Copy

Output:

We love having you with us!

In the above example, if the account balance is >= 1000, we are printing a warm message.

If Then Else If Else - Back to top

If a primary if condition is not satisfied, we can add an Else If statement in between to check another condition if required.

Example:

```
Module Cosmic Tutorial
   Sub Main ()
      Dim account Balance As Integer = 1000001
      If (account Balance < 1000) Then
         Console. Write Line("Close Account!")
      ElseIf (account Balance > 1000000) Then
         Console. Write Line("Please find a Europe tour cruise package in your mailbox!")
      Else
         Console. Write Line("We love having you with us!")
      End If
      Console. Read Line()
   End Sub
End Module
```

VB.Net Copy

Output:

Please find a Europe tour cruise package in your mailbox!

In the above example, we added an else if condition.

For high valued customers who have more than a large threshold of account balance with us, we printed a separate message.

Nested If Then - Back to top

We can nest multiple If Then statements.

Be careful when using this as it can become a mess to read. You can use && or || operators to avoid it in some scenarios.

Example:

```
Module CosmicTutorial
    Sub Main()
        Dim accountBalance As Integer = 501
        If (accountBalance < 1000) Then
            If (accountBalance < 500) Then
                Console.WriteLine("Close Account!")
            Else
                Console.WriteLine("Maintain a minimum balance Pal! You got 5 days time.")
            End If
        ElseIf (accountBalance > 1000000) Then
            Console.WriteLine("Please find a Europe tour cruise package in your mailbox!")
        Else
            Console.WriteLine("We love having you with us!")
        End If
```

```
        Console.ReadLine()
    End Sub
End Module
```

VB.NetCopy

Output:

Maintain a minimum balance Pal! You got 5 days time.

In the above example, we added a nested if statement where we are further checking if account balance is less than 500.

If it is not, then we are giving a friendly and warm warning message.

Switch Statement - Back to top

A switch statement takes an expression and evaluates it.

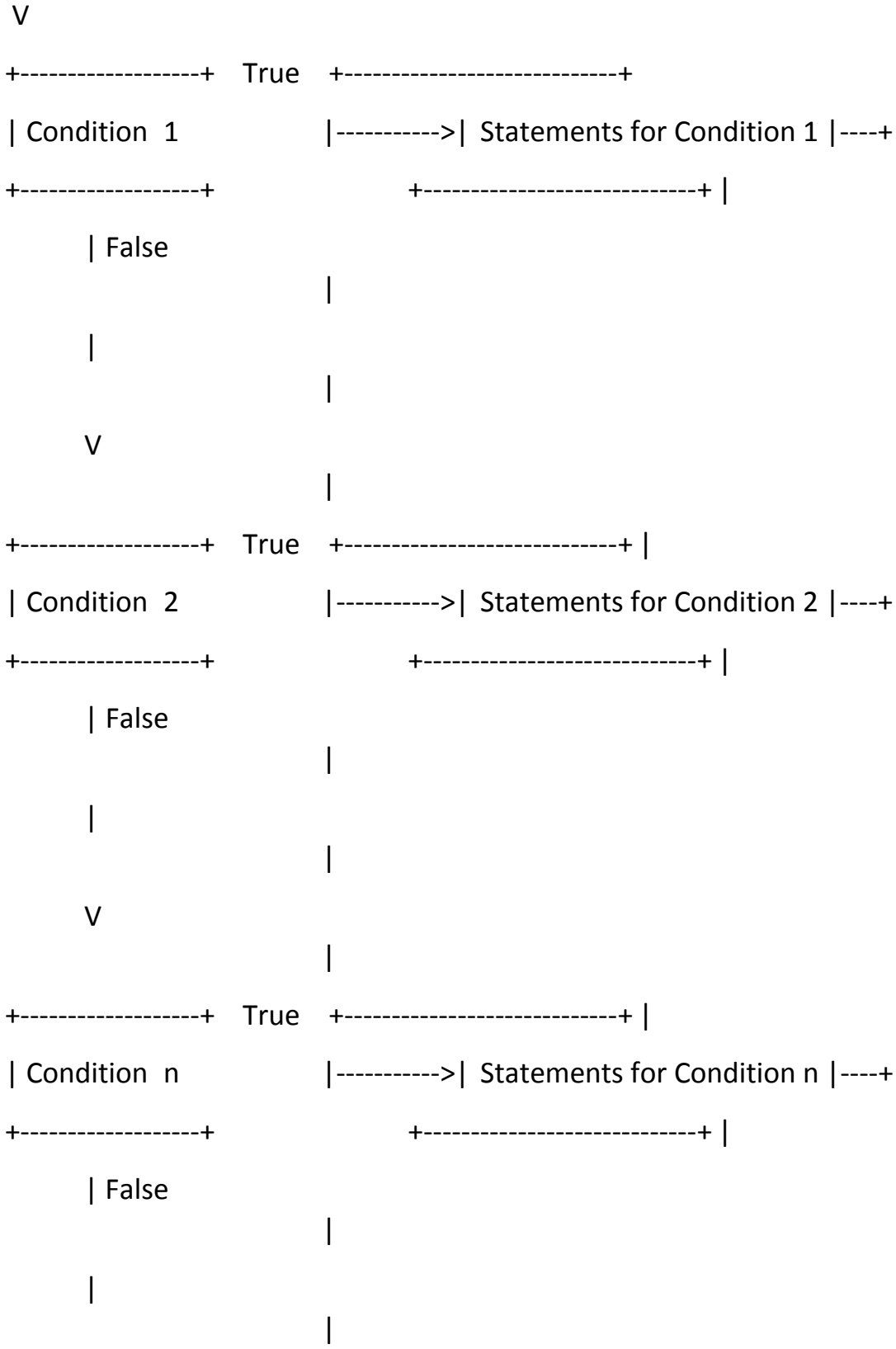
for each result of the evaluation, it executes statements. This is known as a case statement.

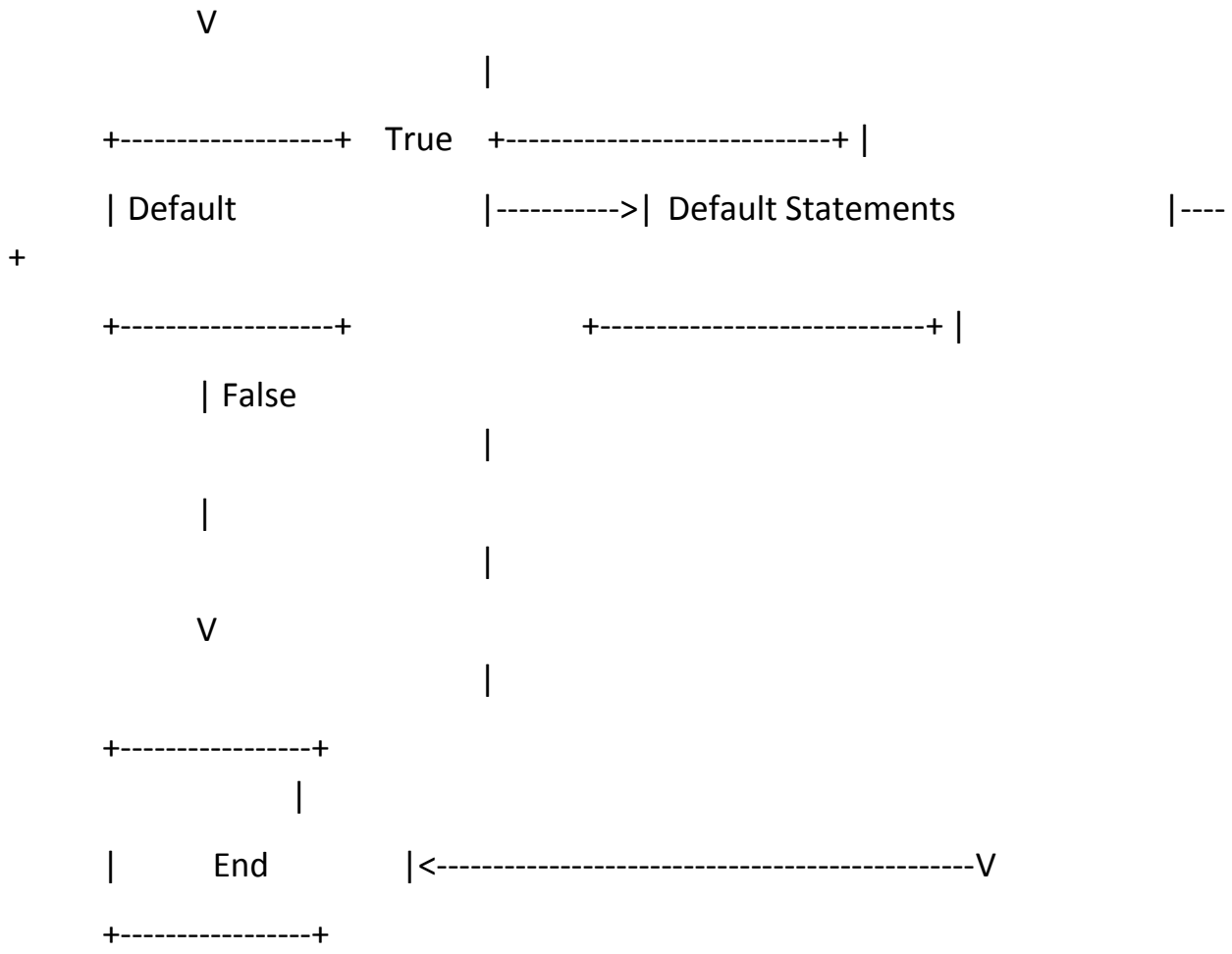
break statement must be present at the end of each case.

default case is executed if any of the case is not satisfied.

Structure of Switch Statement

```
+----------------------+
|  Switch (expression)  |
+----------------------+
    |
```


V
+------------------+ True +--------------------------+
| Condition 1 |---------->| Statements for Condition 1 |----+
+------------------+ +--------------------------+ |
| False
| |
V |
+------------------+ True +--------------------------+ |
| Condition 2 |---------->| Statements for Condition 2 |----+
+------------------+ +--------------------------+ |
| False
| |
V |
+------------------+ True +--------------------------+ |
| Condition n |---------->| Statements for Condition n |----+
+------------------+ +--------------------------+ |
| False
| |



LinuxCopy

Example:

```
Module CosmicTutorial
  Sub Main()
    Dim num As Integer = 0
    Select Case num
      Case 1
        Console.WriteLine("One")
      Case 2
        Console.WriteLine("Two")
```

```
        Case 3
            Console.WriteLine("Three")
        Case 4
            Console.WriteLine("Four")
        Case Else
            Console.WriteLine("Other Number!")
    End Select
  End Sub
End Module
```

VB.NetCopy

Output:
Other Number!

THE END