Syllabus Content:

2.4.1 Programming

- show understanding of the design, coding and testing stages in the program development cycle
- show understanding of how to write, translate, test and run a high-level language program
- describe features found in a typical Integrated Development Environment (IDE):
    - for coding, including context-sensitive prompts
    - for initial error detection, including dynamic syntax checks
    - for presentation, including prettyprint, expand and collapse code blocks
    - for debugging, including: single stepping, breakpoints, variables/expressions report window

Stages in the program development cycle

Problem solving

The first step in solving a problem is to define it clearly. This is usually done in **Structured English** and is known as a 'specification'. The next step is planning a solution. Sometimes there is more than one solution. You need to decide which is the most appropriate.

The third step is to decide how to solve the problem:

- Bottom-up: start with a small sub-problem and then build on this
- Top-down: stepwise refinement using **pseudocode**, **flowcharts** or **structure charts**.

Design

You have a solution in mind. How do you design the solution in detail?
An identifier table is a good starting point. This leads you to thinking about data structures: do you need a 1D array or a 2D array to store data while it is processed? Do you need a file to store data long-term?

Plan your algorithm by drawing a flowchart or writing pseudocode.

Coding

When you have designed your solution you may need to choose a suitable high- level programming language. If you know more than one programming language, you have to weigh up the pros and cons of each one. Looking at Chapter 13, you need to decide which programming language would best suit the problem you are trying to solve and which language you are most familiar with.

You implement your algorithm by converting your pseudocode into program code. Depending on your editor you may have some helpful facilities.

Some syntax errors may be flagged up by your editor, so you can correct these as you go along.

Syntax Error:

A **syntax error** is a 'grammatical' error, in which a program statement does not follow the rules of the high-level language constructs.

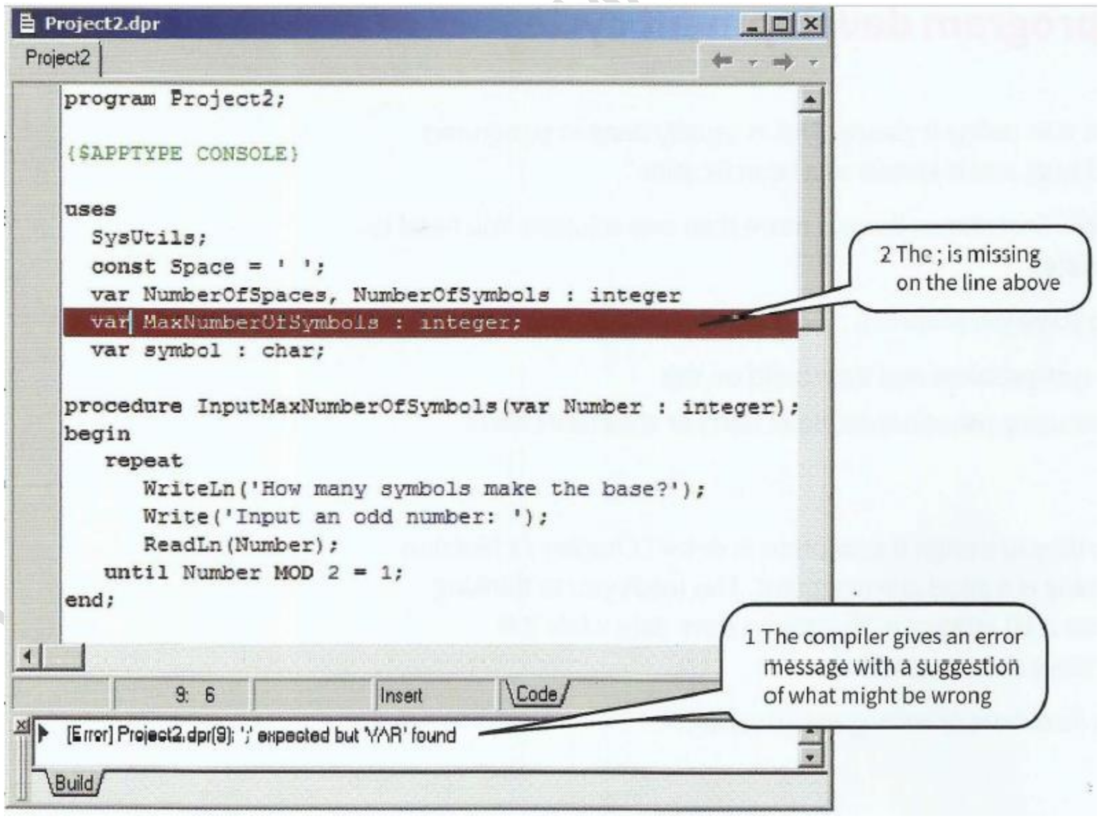
Syntax error: an error in which a program statement does not follow the rules of the language

Translation

Some syntax errors may only become apparent when you are using an **interpreter** or **compiler** to translate your program. **Interpreters** and **compilers** work differently.

When a program compiles successfully, you know there will be no syntax errors remaining. This is not the case with interpreted programs. Only statements that are about to be executed will be syntax checked. So, if your program has not been thoroughly tested, it may even have syntax errors remaining.

Figure below gives an example of how a compiler flags a syntax error. The compiler stops when it first notices a syntax error. The error is often on the previous line. The compiler can't tell until it gets to the next line of code and finds an unexpected keyword.


Project2.dpr
Project2
2 The ; is missing on the line above
1 The compiler gives an error message with a suggestion of what might be wrong
9: 6
Insert
Code
Build

```
program Project2;

{$APPTYPE CONSOLE}

uses
  SysUtils;
  const Space = ' ';
  var NumberOfSpaces, NumberOfSymbols : integer
  var MaxNumberOfSymbols : integer;
  var symbol : char;

procedure InputMaxNumberOfSymbols(var Number : integer);
begin
   repeat
      WriteLn('How many symbols make the base?');
      Write('Input an odd number: ');
      ReadLn(Number);
   until Number MOD 2 = 1;
end;
```

```
[Error] Project2.dpr(9): ';' expected but 'VAR' found
```

Execution

The compiler gives an error message with a suggestion of what might be wrong when you start writing programs you may find it takes several attempts before the program compiles. When it finally does, you can execute it. It may 'crash', meaning that it stops working. In this case, you need to debug the code. The program may run and give you some output. This is the Eureka moment: 'it works!!! !'. But does the program do what it was meant to do?

Testing

Only thorough testing can ensure the program really works under all circumstances

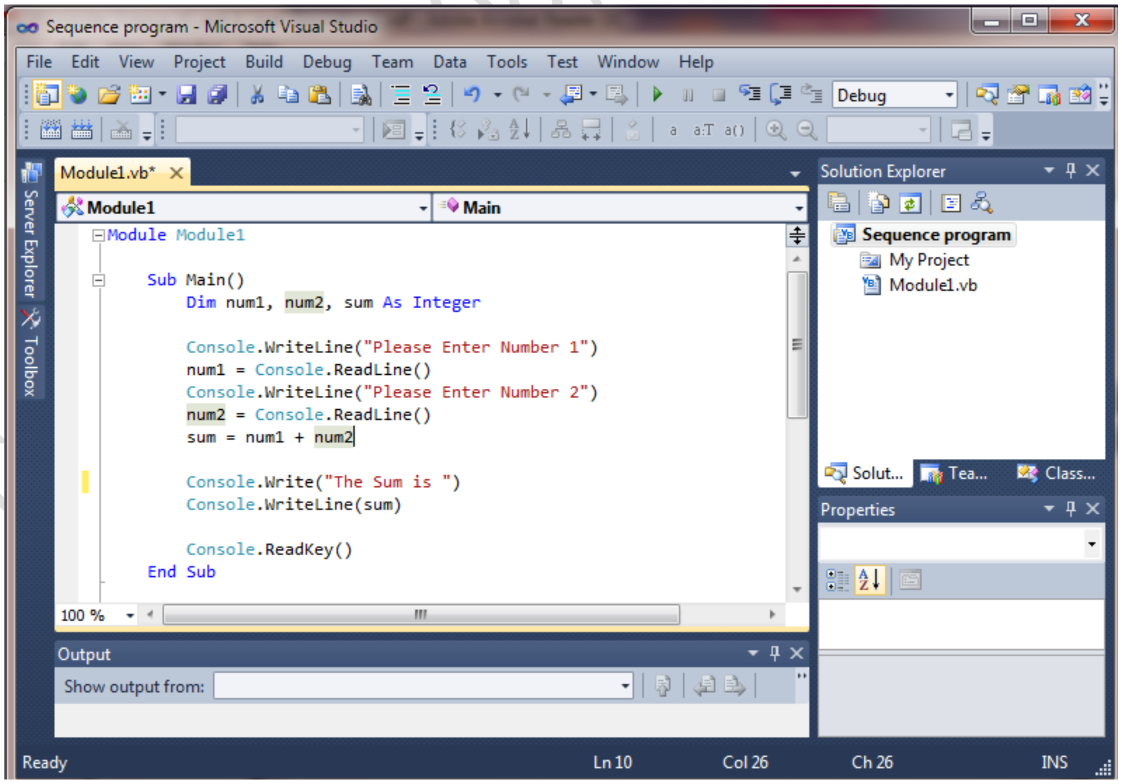
Features found in a typical Integrated Development Environment (IDE)

Pretty-printing

Pretty-print refers to the presentation of the program code typed into an editor. It includes indentation, colour-coding of keywords and comments.

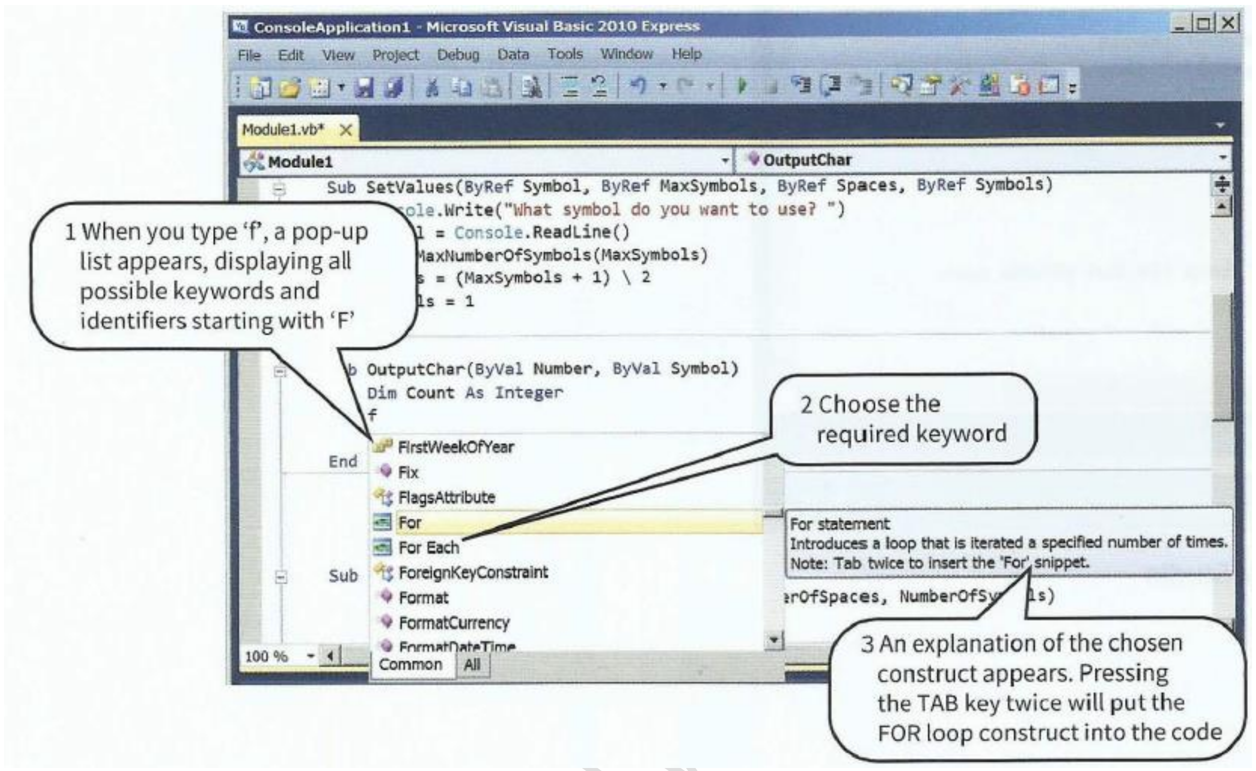
VB.NET

The editor provided by Visual Studio (see Figure 15.03) automatically colour-codes keywords, object references (such as console), comments and strings. The editor automatically indents blocks of code correctly.



```
Module Module1

    Sub Main()
        Dim num1, num2, sum As Integer

        Console.WriteLine("Please Enter Number 1")
        num1 = Console.ReadLine()
        Console.WriteLine("Please Enter Number 2")
        num2 = Console.ReadLine()
        sum = num1 + num2

        Console.Write("The Sum is ")
        Console.WriteLine(sum)

        Console.ReadKey()
    End Sub
```

Figure below shows an example of the Visual Studio editor responding to text typed in by the programmer


ConsoleApplication1 - Microsoft Visual Basic 2010 Express
File Edit View Project Debug Data Tools Window Help
Module1.vb*
Module1
OutputChar
100 %
Common All

```
Sub SetValues(ByRef Symbol, ByRef MaxSymbols, ByRef Spaces, ByRef Symbols)
    ...ole.Write("What symbol do you want to use? ")
    ...l = Console.ReadLine()
    ...MaxNumberOfSymbols(MaxSymbols)
    ...s = (MaxSymbols + 1) \ 2
    ...ls = 1

...b OutputChar(ByVal Number, ByVal Symbol)
    Dim Count As Integer
    f
End
Sub                                ...rOfSpaces, NumberOfSy...ls)
```

FirstWeekOfYear
Fix
FlagsAttribute
For
For Each
ForeignKeyConstraint
Format
FormatCurrency
FormatDateTime

For statement
Introduces a loop that is iterated a specified number of times.
Note: Tab twice to insert the 'For' snippet.

1 When you type 'f', a pop-up list appears, displaying all possible keywords and identifiers starting with 'F'

2 Choose the required keyword

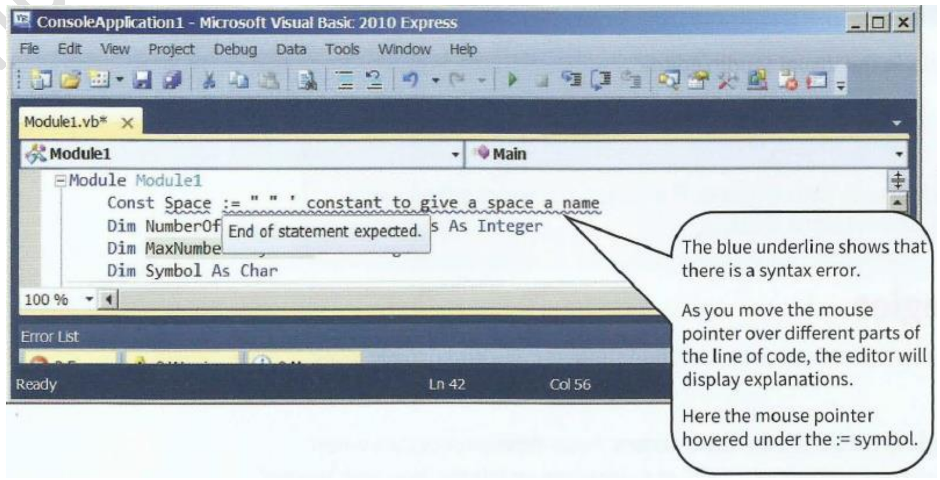
3 An explanation of the chosen construct appears. Pressing the TAB key twice will put the FOR loop construct into the code

Context-sensitive prompts

This feature displays hints or a choice of keywords and available identifiers appropriate at the current insert ion point of the program code.

Dynamic syntax checks

When a line has been typed, some editors perform syntax checks and alert the programmer to errors.
Figure below shows an example of the Visual Studio editor responding to a syntax error.


ConsoleApplication1 - Microsoft Visual Basic 2010 Express
File Edit View Project Debug Data Tools Window Help
Module1.vb*
Module1
Main
100 %
Error List
Ready
Ln 42
Col 56

```
Module Module1
    Const Space := " " ' constant to give a space a name
    Dim NumberOf...s As Integer
    Dim MaxNumbe...
    Dim Symbol As Char
```

End of statement expected.

The blue underline shows that there is a syntax error.

As you move the mouse pointer over different parts of the line of code, the editor will display explanations.

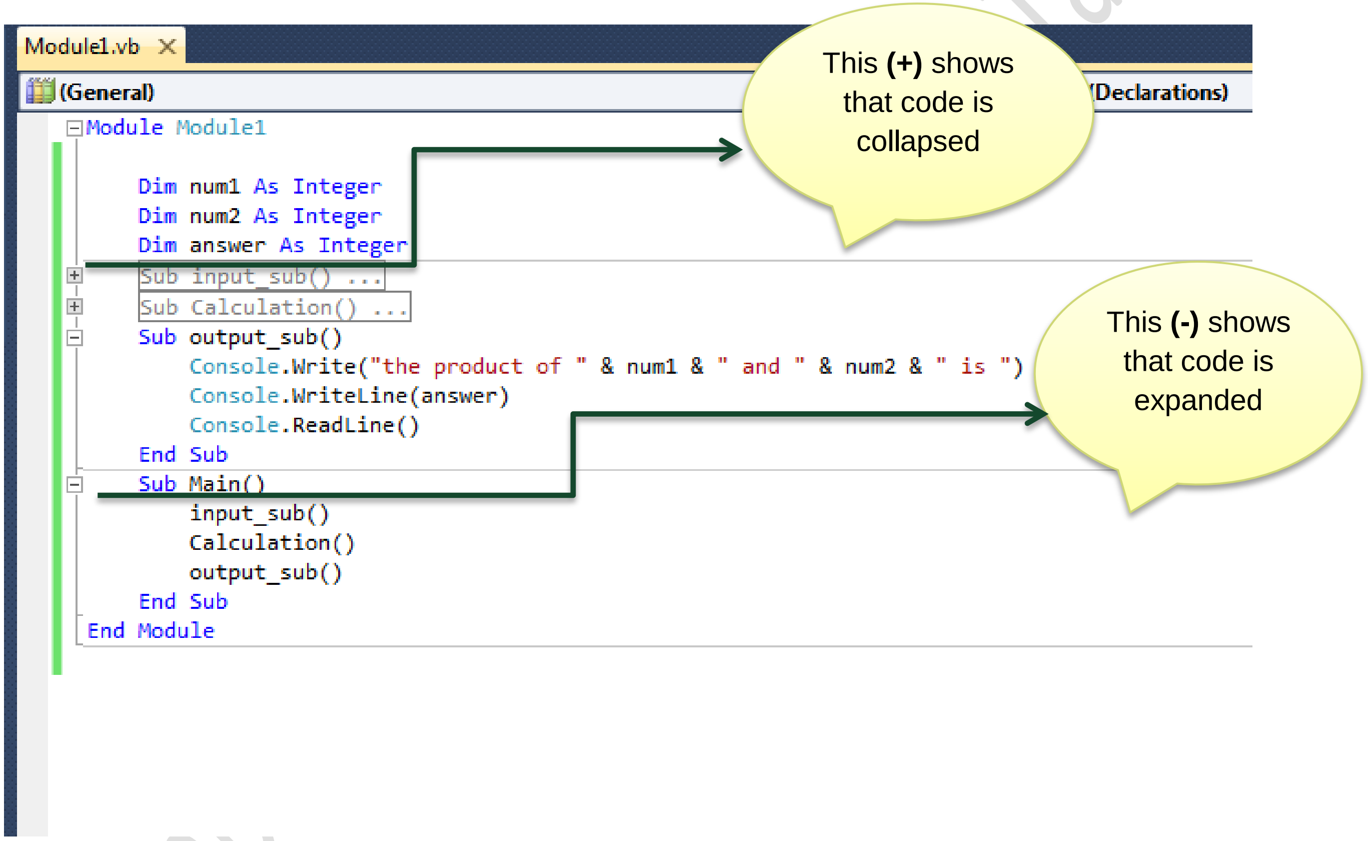
Here the mouse pointer hovered under the := symbol.

Expanding and collapsing code blocks

The blue underline shows that there is a syntax error. As you move the mouse pointer over different parts of the line of code, the editor will display explanations.

When working on program code consisting of many lines of code, it saves excessive scrolling if you can collapse blocks of statements.

Figure below shows the Visual Studio editor window with the procedures collapsed, so the programmer can see the global variable declarations and the main program body. The procedure headings are still visible to help the programmer supply the correct arguments when calling one of these procedures from the main program.


Module1.vb
(General)
(Declarations)
This **(+)** shows that code is collapsed
This **(-)** shows that code is expanded

```
Module Module1

    Dim num1 As Integer
    Dim num2 As Integer
    Dim answer As Integer
    Sub input_sub() ...
    Sub Calculation() ...
    Sub output_sub()
        Console.Write("the product of " & num1 & " and " & num2 & " is ")
        Console.WriteLine(answer)
        Console.ReadLine()
    End Sub
    Sub Main()
        input_sub()
        Calculation()
        output_sub()
    End Sub
End Module
```

References:

- Cambridge International AS & A level Computer Science Course book by Sylvia Langfield and Dave Duddell
- Visual Basics Console Mode Editor Window from notes of Sir Majid Tahir