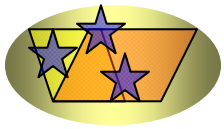
Pascal
Level 2
Notes

Written for Turbo Pascal® Version 7.0

Tench Computing Ltd
Pines
Glendale Road
Burgess Hill
West Sussex
RH15 0EJ

Web address: www.tenchcomputing.co.uk
Email address: jtench@globalnet.co.uk



About the author: *Jackie Tench MSc, ACIB, Cert Ed*

Jackie started her working career in branch banking with the Midland Bank (now HSBC) and was transferred to their Computing Department after achieving 100% in their ability test for programmers. She then worked for more than a decade in this department and was one of the first women to achieve a junior management grade at the age of 21. She attended a significant number of IBM programming training courses during her time there.

Jackie was the first woman to pass the ACIB (Associate Chartered Institute of Bankers) examinations in the Midland Bank (HSBC) and the youngest person at 21 years of age.

Jackie then left to raise a family but still found time to teach part-time at a college in Sheffield and to obtain a MSc in Computing and a Cert Ed in teaching.

When her children were old enough Jackie returned to work full-time and was a Senior Lecturer in Software Engineering and Computer Studies at a college in Brighton for nearly 10 years teaching all levels up to and including HND.

Therefore, Jackie has considerable business knowledge and qualifications plus wide experience in practical computing and training – covering areas such as structured design, analysis, coding, testing and implementing software applications plus training students to fulfil an important role in the computer industry.

Jackie has worked as a consultant for several blue chip companies and examination boards using her software engineering and educational training skills and is now one of the foremost experts in computing with an extensive knowledge of programming languages and applications.

Contents Page

Data Processing ... 1
- Manual system ... 1
- Computer system ... 2
- Programs ... 3
- Writing computer programs ... 5
- Stage 1 designing the program ... 5
- Stage 2 coding the program ... 8
- Stage 3 testing the program ... 8
- Stage 4 documentation ... 8

Integrated development environment ... 9
- Create the source code file ... 9
- Using the compiler ... 9
- Syntax errors ... 9
- Run-time errors ... 10
- Logical errors ... 10
- Compilation ... 10
- Object code file ... 10
- Executable file ... 10
- Library files ... 11
- Reserved words/keywords ... 11

Questions 1 ... 12
Introduction to Pascal ... 13
- Comments ... 13
- Program structure ... 13
- Indentation ... 13

Basic data types ... 14
- Variables ... 14
- Variable names ... 14
- Keywords/reserved words ... 14
- Integer ... 14
- Word ... 14
- Real ... 15
- Character ... 15
- String ... 15
- Constants ... 16
- Literals ... 16

Output to screen ... 17
- writeln() procedure ... 17
- Position and format numbers ... 18
- Rounding ... 18
- write() procedure ... 20

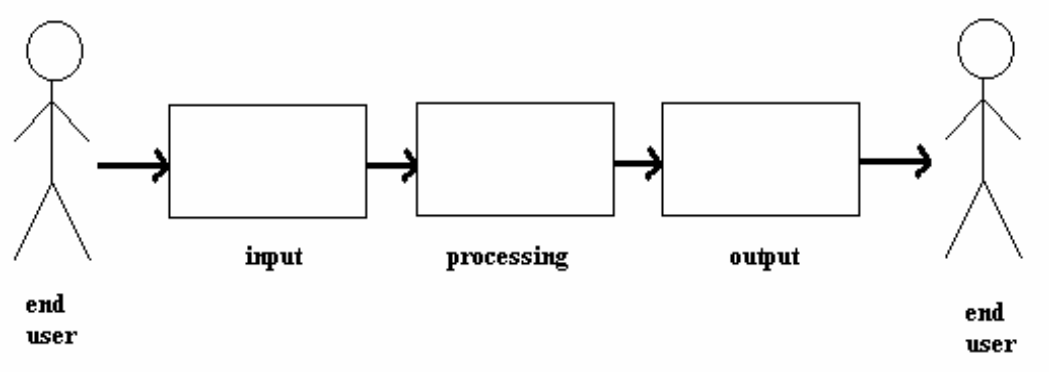
Input from keyboard ... 21
- readln() procedure ... 21
- read() procedure ... 22

Assignment operator ... 23
Arithmetic operators ... 23
Arithmetic operator precedence ... 25
Questions 2 ... 26

Relational operators 28
if…else statement 28
case statement 30
Questions 3 31
Logical operators 33
Loops 35
    while loop 35
    for loop 36
    repeat…until loop 38
Questions 4 40
Validation 43
    Types of validation 43
        Date checks 43
        Range checks 44
Procedures 45
Global and local variables 49
Unit 50
    upCase() function 50
    length() function 50
    val() function 50
Uses clause 51
Crt unit 51
    ClrScr() procedure 51
    GotoXY() procedure 51
    ReadKey() function 51
    Delay() procedure 51
Dos unit 52
    GetTime() procedure 52
    GetDate() procedure 52
Questions 5 55
Testing 57
    Test data 57
    Expected results 57
    Hardcopy 62
    Evidence 62
Questions 6 63
Sample assignment 65
Sample questions 68
Pascal Programming Reference Guide ..... Appendix A
Development environment ..... Appendix B

Data Processing

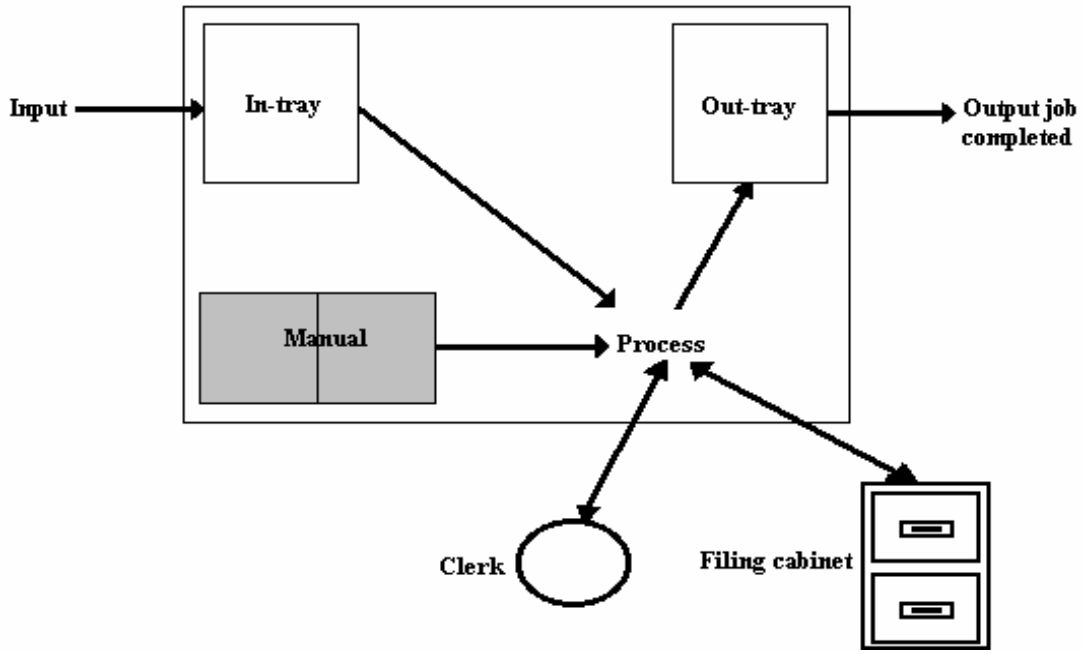
In any data processing system data has to be input then processed and there will be some output or result.



In a manual system all of the work including the processing is done by humans. In a computer system we are called the end users. We must supply some or all of the input and we make use of the output supplied by the computer's processing stage.

Manual system

In the following diagram the input is supplied in the in-tray and when the job is completed the output is placed in the out-tray. While the job is being processed it may be necessary to look up in a manual the instructions needed to process the job, if these are not already known. Also it may be necessary to look up further details from a file in the filing cabinet e.g. a customer's address.



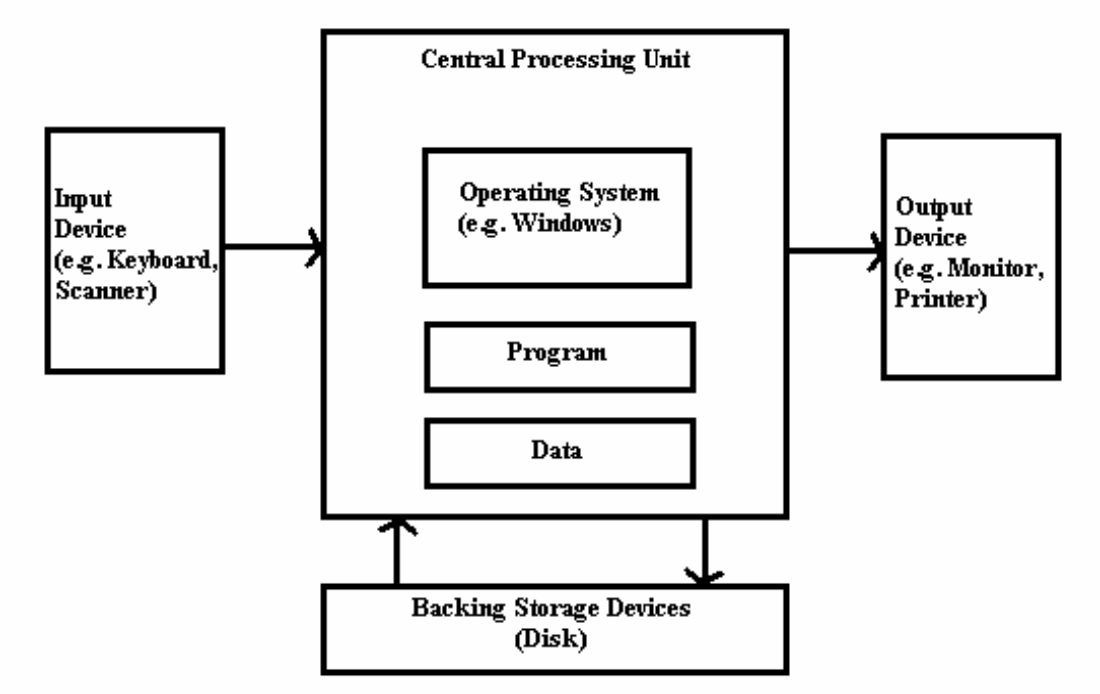
Computer system

In a computer system we also have to input data. This has to be done using input devices. We will be using two input devices, the keyboard of the microcomputer and the disk drives.

The computer needs to be given the instructions to perform the processing. A computer program is a list of instructions that tell the computer what to do. The program is loaded into the computer's memory, which is within the Central Processing Unit (CPU), by using an input device. The Central Processing Unit is the main component in the computer and controls the running of the program and also controls the input from and the output to the devices that are attached to the computer.

When the program is being run the data that is input can be held in the computer's memory while it is being processed. The computer memory is called **volatile** because when the computer is turned off the program and the data that is held in the memory will be lost. Therefore backing storage devices are used to hold the programs and any data that will be required at a later date. The backing storage devices are used in a similar way to a filing cabinet. Data can be written to records in a file on the backing storage and it can also be read from records in a file. In other words backing storage devices can be used as input or output devices.

We will be using the disk drives to store the programs and the data files. Output devices are used to output the results of processing. The printer will be used to give printed output, known as hardcopy.



Programs

Computers are not intelligent; we have to supply the instructions that they need, to be able to process data, by writing the programs that contain these instructions. We use computers to process repetitive tasks because once a computer has been given the instructions to perform a task correctly it can repeat that task as many times as required.

Conversely, if a computer is given the wrong instructions to perform a task, so that the result is incorrect, it will provide the wrong result every time it does that task. This is why it is essential that the programs are correct.

As an example of the type of instructions we need in a program we will write the instructions to make a cup of coffee. These instructions will include the following:

1. Put the coffee in the cup
2. Put the sugar in the cup
3. Put the milk in the cup
4. Turn the kettle on
5. Pour the water from the kettle into the cup

Sometimes it does not matter which order the instructions are in.

We could have put the instructions in the following order:

1. Turn the kettle on
2. Put the coffee in the cup
3. Put the sugar in the cup
4. Put the milk in the cup
5. Pour the water from the kettle into the cup

This would still be correct for making a cup of coffee but will make it quicker because while the kettle is boiling the coffee, sugar and milk are being put into the cup.

These instructions would not be precise enough for a computer. We must specify everything the computer needs to do in order that it can perform a task.

We would know that before we turned the kettle on we should check that it contained enough water. The computer would only execute the exact instructions that we had given it, therefore, it would turn the kettle on without checking that it contained enough water. The instructions should therefore be altered:

1. If the kettle does not contain enough water then fill the kettle
2. Turn the kettle on
3. Put the coffee in the cup
4. Put the sugar in the cup
5. Put the milk in the cup
6. Pour the water from the kettle into the cup

These instructions are still not precise enough for a computer. Again, we would know that the kettle must boil before the water should be poured into a cup but a computer would not. Therefore, we must tell the computer to wait until the kettle has boiled. Also we need to specify exactly how much coffee, sugar and milk is required. The other consideration is that when making a cup of coffee, sugar and milk are not always required. For the moment to keep things simple we will write the instructions to make a cup of coffee that always has sugar and milk.

1. Start
2. If the kettle does not contain enough water then fill the kettle
3. Turn the kettle on
4. Put one teaspoonful of coffee in the cup
5. Put one teaspoonful of sugar in the cup
6. Put 1/2 fl oz of milk in the cup
7. If the kettle has not boiled then go to instruction 7
8. Pour the water from the kettle into the cup until the cup is nearly full
9. Stop

Writing computer programs

"Computer Programming is easy, most people can easily learn to do it". This is a statement that is often heard today especially since the advent of microcomputers. Yes, it is true, that most people can learn to write computer programs, but unfortunately most people cannot do it well. The art of computer programming is not just to produce a program that will work. A program should be carefully thought out and designed before being coded into a computer programming language. It should be documented so that anyone reading the documentation can understand how to make alterations to the program and also how to run the program.

The 4 stages involved in producing a computer program are as follows:

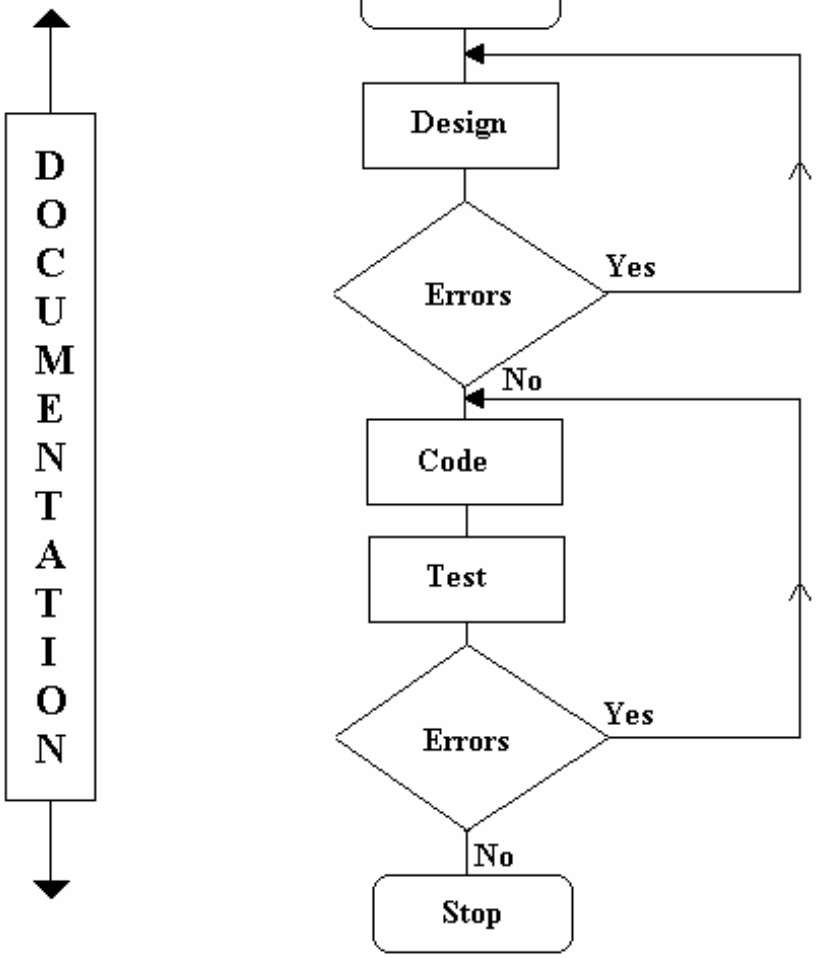
		Time factor
1	Designing the program	50%
2	Coding the program	10%
3	Testing the program	35%
4	Documentation	5%

Stage 1 designing the program

This stage seems to be omitted by a lot of programmers. It is in fact the most important part of producing a program. If a program is designed correctly then it will speed up the processes of coding and testing of a program.

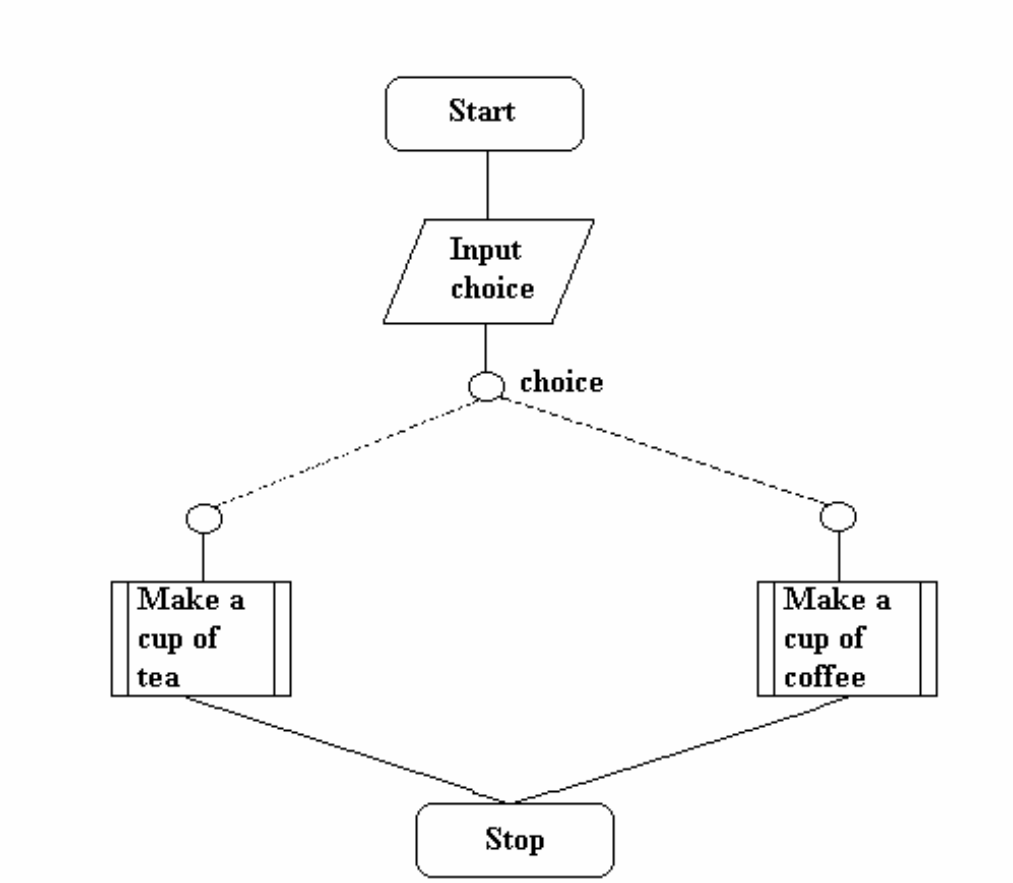
The program is designed and the design must be correct before the next stage of coding the program is started. Once the program has been coded then it must be tested. If any errors are found when the program is tested then the errors must be corrected and the testing restarted. If the program has been badly designed then this could mean that the design stage has to be re-entered so that the design can be altered. This should be avoided, since it is very bad practice to redesign a program once it has progressed as far as the testing stage. In other words it is far better to spend more time on the design and get it correct first time than to have to go back and redesign the program at a later stage.

The processes involved in producing a program are shown in the following diagram:



A program should be designed as a series of self-contained modules. This will aid the later testing stage since the modules can be tested independently. It also means that if an error is found in a module the amendment to correct the error will not affect the operation of the other modules. Most programs require amending at some stage in their life cycle. It is far easier to amend a program that is written in a modular form. This is because an amendment to one module should not affect the other modules in the program.

A modular program that would allow the user a choice of a cup of tea or a cup of coffee could be designed as follows:



The instructions to make a cup of tea would be written in a self-contained module and the instructions to make a cup of coffee would be in another self-contained module. This means that if we changed the instructions to make a cup of tea the module containing the instructions to make a cup of coffee would not be affected. Also the testing of the program, to prove that it is correct, is simpler because each module can be tested independently.

During the design stage exceptions must be looked for and incorporated in the design. It is the exceptions that will cause the program to crash if they have not been identified.

Example

If a program is doing a division then the divisor should be checked, to ensure that it is not zero, and action taken if it is. Division by zero causes a program to crash. It is not good enough to say that the divisor will never contain zero, if incorrect data is present then it is possible and must be allowed for.

Stage 2 coding the program

If the design specification has been done well this part of the process is the simplest. When coding the program the programmer should bear in mind that at a future date someone else may have to alter the program.

Therefore, variables should be given meaningful names; comment lines should be inserted to explain the purpose of routines and programs should be written so that they are easy to amend.

Actual values should not be used in the program if a value is liable to change at a later date. A good example of this is a payroll program. The rates for tax and national insurance are set by the government and are changed fairly regularly. Therefore, these rates should be held as variables, in the program data area, not as actual values within program instructions. Only one alteration is required to alter a value assigned to a variable held in the data area. If actual values are used in the program instructions these instructions have to be found in order to alter the values contained within them.

Stage 3 testing the program

This is another important part of the process of writing a program. The purpose of testing is to find any errors in the program. It is no good just entering data that you know the program will accept.

Invalid data must be entered to ensure that the program will reject it. If numbers to be entered can only be in the range 50 - 500 then numbers below 50 and above 500 should be entered to ensure that the program will reject them. If the program does not reject them then it will be working on incorrect data. All paths through the program should be tested and all the output checked thoroughly.

A test plan should be written out and a record kept of all the tests made and the results obtained. If major faults are found in the program it may be necessary to start the testing again from the beginning. Even the simplest amendment to a program can cause errors to occur which were not present before the amendment was made.

Stage 4 documentation

Although this is called stage 4 the documentation should be started during stage 1.

A design specification should be produced in stage 1. This will include the layout of input and output files, screen and printer layouts.

During stage 2 the programmer should be writing the program description and documentation. This documentation should be a clear explanation of what the program does and how it does it. It should also include instructions for running the program, that is, the operating instructions. If there are any limitations these should also be included, for example, if the program can process a maximum of 500 records, then this must be stated.

Integrated development environment

An integrated development environment is a development environment where the source code is typed in, compiled, linked and run all in the same environment. The development environment is normally set up so that the program is compiled, linked and run in the computer memory as this is faster than creating a file on disk, for the executable, each time. The executable file is then created only when the program has been fully tested.

Create the source code file

The program has to be keyed in, using a text editor, to create a source code file. The source code file is readable and can be edited.

The source code should be keyed in and then saved in a disk file with the extension **.pas** added to the filename to identify it as a Pascal source code file and distinguish it from the other files created by the compiler.

> Save your file frequently while you are keying it in or making alterations in case of a computer or software failure.

Using the compiler

Once the source code file has been created it must be compiled before the program can be executed. The function of the compiler is to convert the source code into machine code that can then be executed. The compiler converts each program instruction into multiple machine code instructions. This is because we are using a high level programming language, it is easy for us to use but the computer does not understand it, therefore the instructions have to be converted to the level that the computer does understand.

The organization of the storage of the data to be used by the program is also done by the compiler.

Syntax errors

If any errors are found in the syntax of the Pascal language (e.g. a semi colon missed out) then the lines in error will be displayed on the screen. You have to be exact when using a computer language because the computer cannot understand if you use the language incorrectly and the compiler will signal an error. The syntax errors must be corrected, using the editor, and the program recompiled until the compilation is error free. A program that has syntax errors cannot be executed.

> If you cannot see a syntax error on the line indicated, the error may be on the line before.

Run-time errors

The program can then be run and tested but may still need correction and recompilation if run-time errors are found during testing. A program crashes (stops running) with a run-time error which can be caused for example by the program attempting to divide by zero, accessing a file which does not exist or accessing a file using an incorrect filename.

Logical errors

Logical errors occur when you give the computer incorrect instructions. This may not cause the program to crash but provide incorrect results. For example if you intended to divide one number by another but instead multiply one number by another the result from the program will be incorrect.

Compilation

The editor is used to create and modify the source code file which can be saved on disk. The source code is then compiled and produces the object code file. The compiler creates the filenames for the output files by using the filename of the source code file and adding the appropriate extension.
e.g.
Source code file name - TESTPROG.PAS
The extension **.pas** is used by the programmer to identify the file as a Pascal source code file.

Object code file

An object code file is created from the source code file and given the extension **.obj**
Object code file name - TESTPROG.OBJ
The object code file is not readable but is in a binary format that the computer can understand. An object code file is an intermediate stage which is used to execute the program in a development environment.

Executable file

Once a program has been fully tested then it can be made into an executable file (.EXE) that is stored on the disk. The .EXE file is made by linking any OBJ code modules for the program with the library files that are required by the program. A standalone .EXE file can be executed outside of the development environment and can be copied and run on another computer that does not have the development environment or a compiler.

Some compilers produce object code that can only be run using a run-time system. This means that if you copy the program to another computer you must copy the run-time system (subject to licence) as well.

Library files

Pascal contains pre-written routines that are stored in library files. These library files can be included in a program, by naming them in a **Uses** clause, so that the program can use the pre-written routines. The library files are provided to save a programmer writing commonly used code from scratch. All the standard routines e.g. for input and output which are needed in a program are contained in the System library file unit, which is automatically included in a Pascal program. Other library file units can be included as required.

You can also write your own code for a unit and include it in your programs with a Uses clause so that the source code is not seen.

In Turbo Pascal the **Uses** clause is used to include library files (units) in the program.
Uses Crt; at the start of a program indicates that the library file **Crt** unit for screen handling e.g. clear screen, move cursor, is to be included in the program.

Reserved words/keywords

Reserved words are words that are part of the programming language. In Pascal words like ***while*** and ***case*** are reserved words that have a special meaning. You cannot use reserved words as names for any variables, constants, functions or procedures that you create. The compiler will signal a syntax error if you misspell a reserved word or use a reserved word in the wrong context e.g. for a variable name.

Questions 1

1. Type in the following program and save the file as **Example1.pas**.

```
program Example1;
var
FirstNumber, SecondNumber : integer;
begin
      FirstNumber := 30;
      SecondNumber := 5;
      writeln('First Number multiplied by Second Number is ',
            FirstNumber * SecondNumber);
end.
```

- Compile the program, correct any syntax (typing) errors and when the compilation is successful run the program.

The program should output
First Number multiplied by Second Number is 150

When using Turbo Pascal press the **Alt+F5** keys to toggle the screen between the source code editor and the user output.

2. Type in the following program and save the file as **Example2.pas**.

```
program Example2;
var
FirstNumber, SecondNumber : integer;
Answer : real;
begin
      FirstNumber := 35;
      write('Enter a number: ');
      read(SecondNumber);
      Answer := FirstNumber / SecondNumber;
(* :4:2 is used to set the display on the screen for the real number
*)
      writeln('First Number divided by ', SecondNumber, ' is ',
            Answer:4:2);
end.
```

- Compile the program, correct any syntax (typing) errors and when the compilation is successful run the program.

When the program is run the prompt **Enter a number** is displayed. Enter an integer and press the ENTER key. The program will then display **First Number divided by *n* is *a*** where n is the number you entered and a is 35 (FirstNumber) divided by the number you entered. Press the ENTER key to return to the source code editor.

If you enter the number 5 the output will be **First Number divided by 5 is 7.00**.

If you enter the number 0 the program will crash (stop) with a runtime error 200 as a computer cannot do a division by zero.