A quick look at programming

1 If you know nothing, computers know less!

Sad but true!

2 Know this ...

There are a few things you need to know about computers before you begin to program.

2.1 Computers are pedantic.

Computers are pedantic. They need explicit instructions, with these instructions a computer will do exactly what you tell it to do. This means that you have to be careful with what you ask the computer to do.

eg. The instructions to draw a one inch square on a plotter in some artificially simple programming language.

move the pen to the center of the page
move the pen onto the paper
move the pen up one inch
move the pen to the right one inch
move the pen down one inch
move the pen to the left one inch
move the pen off the paper

(In this language we can only move the pen up, down, left, right, onto, or off the paper.)

2.2 Computers make no assumptions for you.

Computers make no assumptions for you. If we’d forgotten to say,

move the pen onto the paper

in

move the pen to the center of the page
move the pen onto the paper
move the pen up one inch
move the pen to the right one inch
move the pen down one inch
move the pen to the left one inch
move the pen off the paper

the pen would have moved around in a square but our paper would have been blank.

2.3 A program is...

A set of instructions to a computer is called a program. The form of these instructions depends on which language we chose to write them in and how we plan to issue the instructions. At the end of this course you will be able to:

- Formulate a problem so that it can be solved on a computer, and
- Write programs in Pascal, a simple programming language that will make the principles of programming clear to you.

2.4 Your computer doesn’t understand Pascal!

Your computer doesn’t understand Pascal!

2.4.1 Machine Language.

So what does it understand? Your computer, infact all computers that you’ll use, understand something called machine language. Machine language is a sequence of binary instructions, it isn’t really very readable, or even writable, by human beings.

Binary instructions are groups of binary digits, typically organized into 8, 16, 32 or now, 64 digit long sequences.

A binary digit is either 0 or 1. A bit is a single binary digit, a byte is a group of 8 bits.

So a series of instructions that your computer will understand looks like

```
10010000
00000011
10100000
00000100
10010000
00011000
```

Typically a small program, like one that simply prints, "Hello World" to your screen will be a sequence of hundreds of these instructions.

To make matters more complicated different computers have different instruction sets which means the instructions above may work on a Mac but not on a PC.

A complicated program, like a powerful word-processor that runs with a windowed interface will have millions of these instructions.
2.4.2 Assembly Language.

To make machine language readable computer scientists use assembly language, this might look something like,

```
MOV 3, RA  // Put the value 3 into a memory cell
        // called RA
ADD 4, RA  // Add the value 4 to the value in RA
MOV RA, 48 // Move the value in RA to another
        // part of memory
```

It’s readable, but it’s just as painful as machine language to program in. It isn’t without its merits however, assembly language is often used by programmers and engineers to write programs for controllers and in the past assembly language was used to write small parts of programs that needed to be as efficient as possible.

2.4.3 High Level Programming Languages.

Obviously writing programs this way is a bad idea and we need to come up with a better system for writing complex programs. We write our programs in what is known as a high level language. So to add 3 and 4 we issue the instruction

```
3 + 4;
```

which is a more natural representation of what we wanted the computer to do.

2.4.4 Low vs High level languages.

Machine and Assembly languages are known as low level languages. They are machine specific and generally not portable. Once can write extremely efficient programs in these languages but with greater difficulty.

Pascal and other languages like it are known as high level languages. A high level language allows the programmer to express a solution to a problem in a more natural way. In allowing the programmer to use more natural constructs it may lose some efficiency. We’ll talk about why in a while.

3 Compiling and Executing Programs

So now we know the following:

- We’re going to write programs in Pascal.
- The computer understands only machine language.

You might ask “How does one program a computer if we don’t speak the same language?”

3.1 The Compiler.

The answer is we need to use a translator. A **compiler** is a program that translates a high level language, in our case Pascal, to machine language.

The process of translating the program is known as **compiling**.

```
MOV 3, RA  // Put the value 3 into a memory cell
        // called RA
ADD 4, RA  // Add the value 4 to the value in RA
MOV RA, 48 // Move the value in RA to another
        // part of memory
```

Machine Language

```
3 + 4;
```

Compiler

Your Program

3.1.1 The advantages of compiling.

Compilers give us the following advantages:

- **Easier Programming.** We can write programs in a high level language and automatically translate the program to machine language that the computer understands.
- **Portability.** As long as we have a compiler for a language and a machine we can translate a program to that machine’s language.

This clearly has its advantages. If I have a Pascal compiler for a Macintosh and for a PC then I can translate a Pascal program into machine language instructions for both machines. I don’t have to write the program twice!

3.2 Execution.

The Pascal program is often known as the source code while the compiled binary code, machine language, is known as the executable.

Once a program is compiled into machine language, you can tell the computer to follow the sequence of machine instructions, this is known as **running** or **executing** the program.

4 Errors and Debugging

You’re going to make mistakes. Let me tell you what they’re going to be.
4.1 Compile Time Errors.

Compilers are hard to write properly, they involve a lot of thought, and in the present day will only allow languages which have very strict grammatical rules (known as the syntax of the language).

If you don't follow these rules the compiler will be unable to translate the program to machine language.

eg. If I tell you that you must end every line with a semicolon and you don't then the compiler will report an error.

\[ 3 + 4 \text{ would be wrong} \]

but

\[ 3 + 4; \text{ would be correct.} \]

These mistakes are known as syntax errors and occur at compile time.

4.2 Run Time Errors.

Once you have compiled your program successfully you may attempt to run it, if your program works as you expected it to then you’ve succeeded. You may get unexpected results however, this is known as a run time error, sometimes known as a bug.

Often a program will crash if you have run time errors. Sometimes however you may just get the wrong result.

5 “Don’t Panic!” just deal with your mistakes step by step.

You will make mistakes, most programmers do, so you must always test your programs carefully to make sure they behave in the way you expect them to. Programs that produce wrong answers, even intermittently are simply wrong. You must test your programs!

Often you will need to compile, correct any compile time errors, compile again until those errors are gone, then run your program and correct any run time errors that you have produced. This process is called debugging.

Don’t despair if you spend a lot of time debugging your programs. Giving instructions isn’t the easiest thing in the world when you’re dealing with something as pedantic as a computer. The process can be difficult especially with large programs where you issue thousands of instructions. We will write only small programs this semester, nothing that will cause you to lose much sleep over.

6 The Basic Layout of a Computer.

All computers typically are built around the following parts.

6.1 Central Processing Unit, CPU.

This is typically divided into

1. Arithmetic Logic Unit, ALU.
   This is where the computer does most of its computation.
   ie. It adds, multiplies, divides, etc.

2. Control Unit.
   This is where the computer directs the other components from.
   ie. it decides what instruction will be run next, where and how to get the next bit of information needed etc.

6.2 Memory.

This acts as a store for programs and data.

eg. Random Access Memory, Disk Drives, Magnetic Tapes, CDRoms.

6.3 Input/Output devices.

These are devices through which you communicate with the computer.

eg. Keyboards, screens, printers, card readers, modems.

6.4 The Bus.

These three components are connected by a bus. The bus is essentially a set of wires which allow different devices to communicate with one another.

6.5 The Operating System.

To make using a computer simpler we interact with the computer through an Operating System, OS.

eg. Dos, Windows95, MacOS, UNIX, VMS.

The main purpose of the operating system is to simplify tasks like, running programs, storing data, and managing interaction between the main components of your computer.