Program Planning.

Given an assignment, and you will be given assignments, you will need to do a little thinking before you can write the program that solves the problem. This is because finding the solution to a problem often involves many more steps than one anticipates at first.

1 Developing Programs.

There are four rough stages in the development of programs, they are:

1.1 Specification.

Normally when you write a program you have a specification of what your program must do and how it must behave. This usually involves describing the problem to be solved and how the program you wish to solve the problem should look and behave.

1.2 Design.

You then must design a program that meets the specifications required. This is a well studied area in computer science. Designing programs isn’t trivial. Over the years a few techniques have proved very useful. We will study some of them.

Spending a little time in the design of your program will most certainly save you a lot of time in the implementation.

1.3 Implementation.

Implementation is the stage of program development where you actually start to write your program on the computer. This involves writing and debugging your program.

In programming lingo this stage is called coding.

1.4 Verification.

In this step you must check to see if your program behaves exactly as specified. If it doesn’t you need to go back and fix the implementation, and possibly the design.

2 A Simple Problem.

It helps to look at a simple problem. This will give you a first taste of what it is like to conceptualise and solve a problem on a computer.

2.1 The Problem.

We will attempt to solve the following simple problem.

We wish to conduct a census of people in a city and find out whether they come from the north, south, east or west sides of the city.

2.2 A Simple Solution?

Here is a possible solution.

1. We start by drawing four boxes on piece of paper. We label the boxes N, S, E and W respectively. We will use each box to keep track of the number of people from each part of the city.

2. Initially the value in each box should be set to 0.

3. Now we conduct a survey asking each person which part of the city they are from adding one to the appropriate box based on each answer. To do this we arrange everyone in a line and ask them the questions one by one, moving from the beginning of the line to the end.

4. Print the contents of each box.

Solutions like the one described above are often called algorithms. They describe a precise mechanical way to find the solution to a problem.

3 Pseudocode

One of the clearest ways of expressing algorithms is to use Pseudocode. Pseudocode expresses a solution clearly, while trying to look like a program. Pseudocode programming is NOT actual program code, this makes it more flexible and certainly very easy to read.

It is an important design tool. You should practice writing programs in Pseudocode.

3.1 Our Pseudocode.

We opt to make Pseudocode look very much like Pascal. That way programs that you design are easy to translate into Pascal.

To write a program that surveys only three people we write the following pseudocode.

```plaintext
PROGRAM Survey;
BEGIN
Create a box for N, E, W and S;
Set the value of each box to 0;
Tell people to stand in line;
Question the next person;
```
Let's look at this carefully piece of pseudocode carefully.

1. Every instruction in our pseudocode is known as a statement.

2. All the words in CAPITAL letters are known as keywords, they are special and have a meaning associated with them. (More on this later).

3. The first line is known as the program heading. It indicates that what follows immediately, between the BEGIN and END is the solution to conducting a Survey. This line must end with a semicolon.

4. The part that follows the first line between the BEGIN and END keywords is known as a compound statement. The word END must be followed by a period. BEGIN and END are not statements, but they are used to group statements together.

5. Statements are separated from one another by semicolons. The line

   Print the contents of each box

   doesn’t need a semicolon after it because it’s the last statement in the program.

So far our statements in the Pseudocode have been rather vague. They aren’t legal pascal code, although they’re beginning to look like it.

We need to be more precise about almost everything we came up with. We need to be clearer about what we mean by a box, how to add 1 to each box, how to question each person, how to select the next person and lastly how to display the results on the screen.

4. TOP DOWN AND STEPWISE REFINEMENT.

4.1 Stepwise refinement.

The process we used to arrive at our pseudocode above is known as stepwise refinement. Our solution came from a series of increasingly detailed steps.

4.2 The zeroeth solution.

Our first description of the problem could well have been

   Conduct a survey;

4.3 The first refinement.

Then we came up with the first description of how to conduct this survey.

   Tell people to stand in line;
   Place a zero in each box;
   Question the people in line;
   Print the results;

4.4 The second refinement.

Then we refined the questioning step and became more explicit in our instructions

   Question the next person;
   Add 1 to the appropriate box;
   Question the next person;
   Add 1 to the appropriate box;
   Question the next person;
   Add 1 to the appropriate box;

   until finally we get our pseudocode

4.5 A third refinement.

PROGRAM Survey;
BEGIN
Create a box for N, E, W and S;
Set the value of each box to 0;
Tell people to stand in line;
Question the next person;
Add 1 to the appropriate box;
Question the next person;
Add 1 to the appropriate box;
Question the next person;
Add 1 to the appropriate box;
Print the contents of each box
END.

4.6 Top Down Design

We can use short hand to label parts of our algorithm so it becomes more compact. Look at the second stage of our developed code.

PROGRAM Survey;
BEGIN

Create a box for N, E, W and S;
Set the value of each box to 0;
Tell people to stand in line;
Question the next person;
Add 1 to the appropriate box;
Question the next person;
Add 1 to the appropriate box;
Question the next person;
Add 1 to the appropriate box;
Print the contents of each box
END.
CreateBox;
Prompt;
ZeroBox;
AskandRecord;
PrintResults
END.

Which diagrammatically looks like

```
PROGRAM Survey;
--------+----------------------
---------
|| | | |
CreateBox Prompt ZeroBox AskandRecord PrintResults L1
---------
| |
| Ques-
| tion Add L2
```

Each name in this diagram represents a different set of actions. We look at the design in a top down manner, identifying a set of actions for each level. This is what we mean by top down design. As the levels get deeper the actions to be taken get simpler, and hopefully easier to turn into Pascal statements.

5 Comments.

To make programs and pseudocode easier to read we often leave notes to ourselves and others embedded in the program. This is done using comments.

Comments are lines of text that lie in your program but are never actually translated into machine language by the compiler. In fact they are never even read by the compiler. The serve only to make the code easily readable. Comments in Pascal are lines of text that appear between (* and *).

It is extremely important to make your programs readable. Comments really help with this, especially in long programs. Less that 5 percent of all program development time goes into writing new code. The rest goes into maintaining code, this involves lots of reading.

eg.

```
PROGRAM Survey;
(* Will survey a line of three people and determine which part of the city they come from *)
BEGIN
Create a box for N, E, W and S;
Set the value of each box to 0;
Tell people to stand in line;
(* Question everyone and record their answers *)
Question the next person;
Add 1 to the appropriate box;
Question the next person;
Add 1 to the appropriate box;
Question the next person;
Add 1 to the appropriate box;
Print the contents of each box
END.
```

6 Loops.

Our pseudocoded program is fairly complete now. It isn’t flexible though. Suppose we wanted to design a program that surveyed 5 people, or even 10,000 people. Would we write

```
Question the next person;
Add 1 to the appropriate box;
```

out 5 or 10,000 times? Do we need a new program for a city with 1 person, 2 people, 3 people, 5 people and 10,000 people?

We can make our code more precise by REPEATING, or looping, the AskandRecord part of the pseudocode UNTIL we’ve finished surveying everyone in the city. So now our pseudocode will look like

```
PROGRAM Survey;
(* Will survey a line of three people and determine which part of the city they come from *)
BEGIN
Create a box for N, E, W and S;
Set the value of each box to 0;
Tell people to stand in line;
(* Question everyone and record their answers *)
REPEAT
Question the next person;
Add 1 to the appropriate box;
UNTIL everyone is surveyed;
Print the contents of each box
END.
```

After the word UNTIL we have a condition for looping. If the condition is met we stop REPEATING the instructions in the loop.

Now we have one program for as many people as we want, it’s shorter and much closer to our original description of how we proposed to solve the problem.