Organiser II LZ

MANUAL

© 1989 **Psion**

LZ/LZ64 Programming Manual

Contents

1 Introduction to OPL
- The Prog menu
- Creating, saving and running a procedure
- Edit, Print, Dir, Copy, Delete

2 Procedures and variables
3 Loops and branches
4 Operators
5 Handling data files
6 Handling any type of file
7 Error handling
- Common errors
- Run-time errors
- Error trapping

8 Example programs
9 OPL commands and functions
- Summary
- Command syntax
- Function syntax
- List of commands and functions

Appendices
A Organiser character set
- Printing non-keyboard characters
- Codes for special keys
- Control characters
- User-defined characters

B Technical Data
C Technical programming
- Memory addresses

D Error messages

1 Introduction to OPL

OPL is the Organiser Programming Language. It has a set of commands and functions suitable for all kinds of applications - including the manipulation of records in data files.

To enter OPL:

- Select Prog from the main menu, by pressing P.

From the Prog menu you can start new programs or continue with old ones.

Once you have written programs in OPL, you can run them from within Prog or directly from the main menu, or even whilst you are working on the calculator or the Pocket Spreadsheet.

An OPL program consists of one or more **procedures** each of which is typed in separately. A simple program may consist of just one and a more complex program of a main procedure which calls others.

The most efficient way to use OPL is to write short procedures which can be tested individually. Each one should ideally perform just one specific task. That way, programs which have similar requirements can share one common procedure to do the same job.

This chapter shows you how to write. save and run a simple procedure and covers all the options on the Prog menu.

The Prog menu

The Prog menu looks like this:

```
                 11:32a
Edit    New      Run
Print   Dir      Copy
Delete
```

The options on the Prog menu are:

Edit	Lets you alter an existing procedure.
New	Lets you type in and save a new procedure.
Run	Executes an existing procedure.
Print	Prints out a procedure on an attached printer or computer.
Dir	Provides a directory list of your procedures.
Copy	Copies procedures to another device.
Delete	Deletes procedures.

Creating, saving and running a procedure

The simple procedure below just clears the screen and displays the date until you press a key. The procedure's name is DATE:

```
DATE:
CLS
PRINT "TODAY IS",DAY;"/";MONTH
GET
```

1 Creating a new procedure

- Select **New** from the menu and the screen shows:

  ```
  New A:_
  ```

The current device is shown after the word New; in this case it is A: (the internal memory). If you want to work on a pack, press **MODE** to change device.

The first thing to type is the procedure name, this can be up to 8 characters long and must start with a letter.

- Type in DATE as the name for your first procedure. **(Don't type the colon.)** Press **EXE**.

The procedure name is shown with a colon at the end. The cursor is flashing at the end of it.

```
DATE:_
```

- Press **EXE** to move the cursor down to the next line and you can begin typing. Typing in the OPL editor is the the same as typing in the notepad.

If you are not used to the keys look at the Keyboard section in Chapter 2 of the operating manual.

Note: in OPL you can use either upper case or lower case letters in any combination.

- Type the first line:
  `CLS`
- This is a **command**. When you run the procedure, CLS is an instruction to OPL to clear the screen.
- Press **EXE** to start a new line, then type:
  `PRINT "TODAY IS",DAY;"/";MONTH`

Check that the line appear on screen exactly as it does here as even the spaces are important. Here is an analysis of the line:

- PRINT, is a **command**. It makes what. follows it print on the display screen. All commands are followed by a space.
- "TODAY IS" is the text to be printed. A piece of text in quotes like this is called a **string**.
- A **comma** makes the next thing to be printed follow on the same line after a space.
- DAY is a **function**. It finds out the date. Here it returns it to the print command, which displays it.
- A **semi-colon** makes the next thing to be printed follow on the same line without a space.
- "/" is another string of just one character and MONTH is another function.
- Now press **EXE** to start a new line, and type:
  `GET`

The GET function, waits for a keypress before running the rest of the program. So when you run DATE:, the date will remain on the screen until you press a key.

Editing what you've typed

You can correct what you've typed at any time:

- Use **LEFT** , **RIGHT**, **UP** and **DOWN** to move around.
- Use **DEL** to delete the character to the left of the character and hold down **SHIFT** and press **DEL** to delete the character under the cursor.
- To insert a new line between two existing ones, press **EXE** on the first character of the second one.
- To join two lines, press **DEL** on the first character of the second one.
- To delete a line, press **ON/CLEAR**.

You can also press **MODE** and use four of the options on the editor menu to help you:

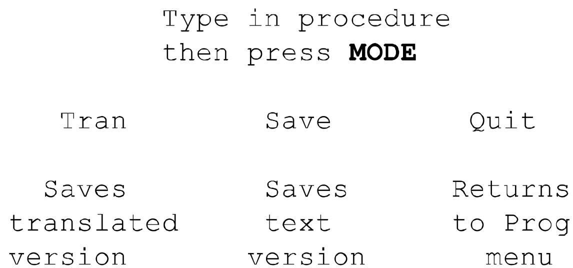
Find Takes you to a search-clue you specify (you have to press **ON/CLEAR** to remove the find prompt).
Home Takes you to the top of the procedure file.
End Takes you to the bottom of the procedure file.
Zap Clears all the lines deleting the whole procedure text so that you can start again.

2 Saving and translating procedures

When you have finished typing the procedure, you can:

- Translate it into a form which OPL can run.
- Save it as it is - so you can edit it but not run it.
- Quit and abandon it.
- Press **MODE** to get the editor menu. The first three options are: `Tran Save Quit`

This diagram illustrates the differences between them. The bold path shows the normal process of translating a new procedure:



You would normally select Tran, so that you can run the procedure, but to see what the other two do:

Quit

- Select Quit and you are prompted with the message: `Quit? Y/N`
- Press **N**, and you return to the procedure editor. (If you pressed **Y**, everything you typed during the editing session would be discarded.)

Tran

- Select Tran.

The procedure is translated into a form that OPL can run. When the procedure has been translated, the screen displays this prompt: `Save A:DATE`

You can now save the translated procedure.

- Press **MODE** to change device if you want to save on a pack, then **EXE** to save.

You return to the Prog menu. When you save the translated version, the original text version is saved with it. So you can run it and re-edit it later.

Syntax errors

If you make a typing error, OPL spots it during translation. For example, if you typed PRONT instead of PRINT or omitted one of the pair of quotation round a string, this message is displayed:

```
      ERROR
    SYNTAX ERR
....................
  press SPACE key
```

- Press **SPACE** and you return to the procedure editor, with the cursor near the syntax error.

- Correct it, press **MODE**, and select Tran again.

Save

When you just save a procedure rather than translating it, no error checking is done. The text is simply saved exactly as you typed it.

If you type in part of a procedure and intend to return to it later to complete it you can select Save instead of Tran so as not to waste memory by producing an unnecessary translated version.

Where you should save programs

The best place to save programs is on A: or a Rampak. Then, if a procedure takes a couple of versions before it runs properly, each version will not use up space:

- When an edited version is saved on A: or a Rampak, the old version is deleted completely each time the new version is saved or translated.
- On Datapaks, however, the old versions remain there taking up space, you just can't access them.

When a final version of a procedure has been produced on device A: it is a good idea to copy it to a Datapak.

3 Running a procedure

When you've successfully translated the procedure, you can run it.

- From the Prog menu, select Run. The name of the procedure you just translated is supplied for you:
  `Run A:Date`
- Just press **EXE** to run the procedure.

When DATE: is run, the screen will first clear and then show something like this, depending on the date:

```
TODAY IS 19/8
```

The last line of the procedure was GET. This instruction simply waits until you press a key before continuing. So, when you press a key, the procedure finishes and you return to the Prog menu.

Edit, Print, Dir, Copy, Delete

Like Run, the remaining Prog options all require you to specify an existing procedure file.

If you haven't left Prog, the name of the procedure you worked on last is supplied for Edit and Print. You can press **EXE** to accept it, or **DOWN** to get a list of procedures to select from. For detail on selecting from file lists, see Chapter 12 in the operating manual.

Editing an existing procedure

The Edit option from the Prog menu allows you to return to an old procedure to change it or add to it.

- Select Edit and press **MODE** to change device if necessary, then select a procedure file.

When you've finished editing the procedure, you can either:

- Press **MODE** then **EXE** to translate and save it.

or, if you make a mess of the editing:

- Press **MODE** then 9 to quit and delete this version of the procedure so you can edit the original again.

Printing a procedure

The Print option is used to print out a listing of a procedure on a printer or personal computer.

- Select Print then select a procedure file.

If no printer or computer is connected to the Organiser, this error message is displayed:

```
DEVICE MISSING
```

For more details on printing, see Chapter 17 in the operating manual.

The procedure directory

The Dir option on the Prog menu shows you the directory of procedures stored on the various devices.

- Select Dir from the Prog menu. The screen shows:

```
Dir A:
```

If necessary, change device with **MODE**, then list the procedures with **DOWN** in the same way as you do in Dir in Utils. See Chapter 15 of the Operating Manual.

Copying a procedure

The Copy option in the Prog menu is used to make copies of procedures from one device to another.

- Select Copy from the Prog menu:

```
       Copy
   Select type
....................
Opl Oplobj Opltxt
```

Opl Copies both the text and translated object code of the procedure.
Oplobj Copies only the translated object code part.
Opltxt Copies only the editable text part.

- If you select Opl you will be able to edit and run the copy.
- If you select Oplobj you won't be able to edit it.
- If you select Opltxt, you won't be able to run it until you have translated it again.

Copy in Prog works just like Copy in Utils.

Warning: If you copy a procedure, and one with the same name already exists on the destination device, the existing one is deleted - even if the existing one is text only and the one you are copying is object only.

Deleting a procedure

The **Delete** option allows you to delete procedures from any of the devices.

- Select Delete to get the prompt: `Delete A:_`

If necessary change device with **MODE**. Select a file or files to be deleted in the same way as you do in the Utils Delete option.

Running a procedure from the main menu

To insert the name of a procedure in the main menu:

- On the main menu, press **MODE** with the cursor where you want the name to be.
- Type in the name of the procedure.
- Select Opl.

When you select the name, the procedure is run.

For example, you could type in, translate and save a procedure like the one below, then enter its name, ID, in the main menu. When the item Id is selected, your Organiser will be identified.

```
ID:
CLS
PRINT "IF FOUND PLEASE RING"
PRINT "PAUL SMITH EXT. 998"
GET
```

Pausing and quitting procedures when running

Sometimes you may want to stop a procedure when it's running. To halt the execution of a procedure:

- Press **ON/CLEAR**.

This will pause it indefinitely. (Unless the procedure was waiting for a key, with a function such as GET.)

- Press **Q** to quit (or any other key to continue).

If the procedure was run from the main menu, the screen shows:

```
       ERROR
ESCAPE IN A:procname
....................
  press SPACE key
```

- Press the **SPACE** key to return to the main menu.

If the procedure was run from the Prog menu, the screen shows:

```
       ERROR
       ESCAPE
....................
Edit A:procname Y/N
```

Press **Y** if you want to enter the procedure text to edit it, or press **N** to return to the Prog menu.

Renaming a procedure

You can't use Copy to rename a procedure you have to re-save it under a different name. For example, to rename A:DATE to A:TODAY.

- Select Edit to edit A:DATE then select Tran.
- When you see the Save A:DATE prompt, press **ON/CLEAR** then enter the new name, TODAY and press **EXE**.

Translating a procedure for an XP or CM

If you are creating a procedure to be run on a two-line Organiser model CM or XP:

- Select Xtran instead of Tran when you've finished editing it

When you run a procedure created on an XP or translated with Xtran, anything printed on the screen has a border like this round it:

```
XXXXXXXXXXXXXXXXXXXX
XX  2-LINE PROC   XX
XX                XX
XXXXXXXXXXXXXXXXXXXX
```

Summary

Entering and running a procedure.

- Select Prog on the main menu.
- Select New.
- Type a name. Press **EXE**.
- Press **EXE** again to start a new line. Type in the procedure.
- Press **MODE** and select Tran (Xtran to run on a two-line Organiser XP or CM).
- Press **MODE** if you need to change device, then **EXE** to save.
- Select Run to execute the procedure.

Inserting a procedure name in the main menu

- On the main menu, press **MODE** with the cursor where you want the name to be.
- Type in the name of the procedure.
- Select Opl.

When you select the name. the procedure is run.

Escaping from a running procedure

- Press **ON/CLEAR**.
- Press **Q** to quit (or any other key to continue).

2 Procedures and variables

The previous chapter dealt with the mechanics of using the Prog menu. The next five chapters cover the basic concepts of OPL programming. If you are familiar with programming languages just skim through these

chapters, or refer instead to the reference sections which follow.

Procedures generally consist of the four steps shown in this simplified example:

1 Name	`SINE50:`
2 Declaration of variables	`LOCAL x`
3 Operations upon variables	`x=SIN(50)`
4 Communication of variables	`PRINT x`

This chapter deals with these four steps.

1 Procedure names

e.g. **print:**, **shares89:**, **TAXCALC:**

- They may be up to 8 characters long and can combine numbers and letters. **The first character must be a letter.**
- They end with a colon, but it is only in certain circumstances that you have to type this in. For example, you don't have to type it in when naming a new procedure but you do when you call a procedure from within another one.
- They may be typed in upper or lower case letters.

2 Declaring variables

If you were adding together two numbers, in algebra you would write 'x+y=z'.
In OPL you would write z=x+y but first you would have to **declare** x,y and z as variables in order to reserve memory space for them. You would do this:

1 Declare the variables in order to reserve 3 spaces in memory, calles x, y and z.

```
LOCAL x,y,z
```

2 Assign values to x and y.

```
x=5
y=2
```

3 Add x and y together and assign the result to z:

```
z=x+y
```

A variable is therefore a named region of memory which at the very beginning of your procedure you **declare**. This means that you tell the Organiser you are going to use the variable so that it will have space to store the number or text you later assign to it. So in the SINE50 example, the line 'LOCAL x' reserves a memory space named x in which the value given to x in the next line can be stored.

Variable names

There are three kinds of variables identified when you declare them by the format of the variable name. The three kinds are:

Floating point variables e.g. x
Integer variables e.g. x%
String variables e.g. x$

All variables may be up to 8 characters long. The first character must be a letter. The other characters may be letters or numbers but not symbols - except for the identifiers % and $ at the end.

Floating point variables

A floating point number is one which has a decimal point and then any number of digits after that point, e.g. 13.567 or 8. or 0.05319 or 6.0

You should declare a floating point variable when you know that the sort of value you are likely assign to it will be a floating point number.

A floating point variable name does not have any special symbol at the end.

e.g. a, AGE, PROFIT89.

Floating point variables are stored to an accuracy of 12 digits and must be in the range ±9.99999999999E99 to ±1E-99 and 0.

Integer variables

An integer is a whole number, e.g. 6 or 13 or -3 or 11058

You should use integer variables wherever floating point numbers are not necessary and speed or space are important. Integer arithmetic is faster than floating point and it occupies two bytes of memory instead of eight.

An integer variable name ends with a % sign. (The % sign is included in the 8 characters length.)

e.g. a%, AGE%, PROFI89%.

Integer variables must be in the range -32768 to +32767.

String variables

A **string** is a sequence of characters, alphabetic, numeric or symbolic, which is treated literally rather than being evaluated. Examples of strings are: "x + y =" and "01-345-2908" and "profit".

A string variable name ends with a $ sign. (The sign is included in the 8 characters length.)

e.g. a$, NAME$, MAN6$.

When declaring a string variable, you must state the maximum length of the string you expect to assign to it. So if you want to enter names up to 15 characters long as the value of NAME$, you have to declare NAME$(15). The number goes in brackets.

The maximum length of a string is 255 characters.

The LOCAL and GLOBAL commands

You must declare your variables immediately after the procedure name. You may list together all 3 types, in any order; they must be separated by commas like this:

```
LOCAL x,y,a%,NAME$(15),YEAR3%.
```

To declare variables you use either the LOCAL or the GLOBAL command like this:

```
LOCAL x,a%,list$(8)
```

or

```
GLOBAL x,a%,list$(8)
```

LOCAL and GLOBAL define the **range** the variables are to be active in. The basic difference is that:

- Local variables are limited to the procedure in which they are declared.
- Global variables can also be used in any procedures called by the procedure in which they are declared.

Calling procedures

An OPL program can be made up of more than one procedure. However, you have to type, translate and save each procedure separately.

In the example below, the fourth line of proca: calls another procedure, procb:. To do this, you just type the procedure name (with the colon).

```
proca:                  procb:
GLOBAL a%               a%=a%+4
a%=2
procb:
PRINT a%
```

You declare a% as a global variable so that it can be used in the second procedure. In this example the value printed out when proca: is run is 6.

The only danger with global variables is that you may get mistakes occurring if you accidentally use the same variable name twice. So, you should use the LOCAL command unless the GLOBAL one is required.

If OPL comes across a variable which isn't declared in that procedure, it assumes it has been declared in a previous procedure. OPL will then report a MISSING EXTERNAL error if it can't find the global variable in a calling procedure.

Other variables

Calculator memories

There are ten floating point variables which are always available. These are the calculator memories m0 to m9. You do not declare these as variables, as they are always in existence.

Values may be assigned to these at any time in any procedure. You can then access them in the calculator.

Array variables

You may want to declare a large number of similar variables at the beginning of a program. For this reason. OPL has **array variables**.

The idea is simply that, instead of having to declare variables a, b, c, d and e, you can declare variables a1 to a5 in one go like this:

LOCAL a%(5)	(integer variable array)
LOCAL a(5)	(floating point variable array)
LOCAL a$(5,8)	(string variable array)

Numeric array variables may be thought of as a list of numbers, each with the same name, but with an index number to differentiate them.

When the array is declared, the number in brackets is the number of elements in it. Here is a simple example assigning values to the elements of in integer array:

```
procname:
GLOBAL num%(4)
num%(1)=1
num%(2)=3
num%(3)=5
num%(4)=7
PRINT num%(1)+num%(2)+num%(3)+num%(4)
```

This example just prints the sum of the four elements of the array num%.

With strings, you must declare the number of elements in the array **and** the maximum length of the strings.

For example GLOBAL ARRAY$(5,10) allocates memory space for five strings, each up to ten characters in length, under the names ARRAY$(1) to ARRAY$(5). As yet, each of the variables is empty (is a **null string**), but enough memory still has to be set aside to contain all of the five strings when full.

3 Operations upon variables

Once you have declared your variables you can perform any number of operations on them. This might be a combination of arithmetic operations, or you might pass the variables to other procedures for them to operate upon them, or use one of the OPL functions. Whatever you do, however, you need to understand how your variables will react according to what type they are and how you combine them.

For example, you cannot divide a string variable by an integer variable. And if you mix integers and floating point variables in the same sum one may convert the other into its own type of variable. The following sections give details of what problems may arise and how to avoid them.

Mixing integers and floating point variables: automatic type conversion

In the procedure below there is a potential mistake in the third line after the name, where the integer variable y% is assigned a floating point value:

```
procname:
GLOBAL x%,y%
x%=7
y%=3.7+x%
PRINT y%
GET
```

OPL deals with this in the following way: instead of reporting an error, OPL carries out an **automatic type conversion** internally on the value assigned to the mismatched variable.

The right hand side of y%=3.7+x% is evaluated to 10.7. However, the fractional part of the number is dropped before the result is assigned to the left hand side, y%. So, the PRINT statement will display the value 10.

Since OPL does not report this as an error, the onus is on you to ensure that it does not happen - unless you want it to! You must always take care when mixing variable types that the answer produced is the one which you expect.

In the procedure below where only floating point variables are used, you can see that another type conversion is made, but does not cause the value to change:

```
procname:
GLOBAL a,b,c
a=1.2
b=2.7
c=3
PRINT a+b+c
```

In this procedure, the floating point variable c is given the integer value 3. An automatic type conversion is carried out in such cases. Here the result is converted to 3.0, so the real value of the variable remains the same.

If you assign a floating point number to an integer variable, then the automatic type conversion will generate an integer rounded **down**. So if you say a%=2.3 then the value of a% will be 2; but a%=2.9 would also give a% the value 2. And if you assign a negative floating point number to an integer variable, then the number is still rounded down - rather than toward zero. So if you say a%=-2.3 then a% will take the value -3. This may not be desirable..

If you expect an expression to return a floating point number, you must ensure that the correct types of number are used within that expression.

It is possible to control how floating point numbers are rounded when converted. For example, if you wanted to round floating point numbers to the nearest half (so 2.4 would round to 2.5 and 2.2 to 2) then you might try the following statement:

```
r=INT(2*n+0.5)/2
```

where n is the number to be rounded.

This would produce the wrong result, though. To see why substitute a trial value, say 3.4, instead of n:

```
INT(2*3.4+0.5) i.e. INT(7.3).
```

returns the integer 7. But 7/2, **when rounded down** gives the integer 3, not 3.5.

To obtain 3.5 you must force the division to give a floating point result. In this case the simplest way to do this is to divide by the floating point value of 2.0, instead of the integer 2. So the expression:

```
r=INT(2*n+0.5)/2.0
```

will give the required result.

There is more about integers and floating point variables in the chapter on Operators.

Mixing strings and numbers

If you try to allocate a number to a string variable, an error will be reported. There is no automatic type conversion between string and numeric variables. However, OPL does have facilities for forcing conversion of numbers to strings and vice versa. These are the SCI$, FIX$, GEN$, NUM$ and VAL functions.

Concatenating strings

Adding strings together is as easy as adding numeric variables:

- If a$ is "DOWN" and b$ is "WIND", then the statement c$=a$+b$ means c$ is "DOWNWIND".

Alternatively, you could give c$ the same value with the statement c$="DOWN"+"WIND".

When concatenating strings, the result cannot be longer than the maximum length you declared.

Slicing strings

You can also split strings up. This process is known as **string slicing**.

There are three functions which allow you to do this. They are LEFT$, RIGHT$ and MID$. These allow you to access the left, right or middle portions of a string respectively. For example:

- If a$="01-234-5782", then b$=LEFT$(a$,2) assigns the variable b$ the string 01.

String slicing operations leave the original string unchanged - i.e. a$ would still have the value 01-234-5782 after b$ had gained the value 01. The exception would be when the left hand side of the expression is the same string as appears in the right hand side. Eg a$=LEFT$(a$,4) would return a string containing the leftmost four characters of a$ and assign it to a$, thus overwriting the original value.

Note: If you need to define a string which includes the quotation mark character (ASCII character 34) then this character must be included twice in the string. So, if you say a$="x""y""z", then the resulting value of a$ will be x"y"z.

4 Communication of variables

In the first example procedure at the beginning of this chapter, the statement PRINT x simply takes the value found in the memory space of variable x and prints it on the screen.

Values can also be passed between procedures. In the example on page 2-5 [See Calling procedures], the procedure proca: called another procedure, procb: which returned the adjusted value of a% back to proca:. The variable a% was declared global to allow this.

The rest of this chapter deals with two other ways of communicating values: entering values from the keyboard using the INPUT command, and passing values between procedures using parameters.

The INPUT command

Values can be entered from the keyboard using the INPUT command, as in the example below.

This simple procedure just asks you to enter a number. The number you enter is assigned to the variable x and is then printed out to the screen with a message:

```
INPUT:
LOCAL x
CLS
PRINT "ENTER NUMBER"
INPUT x
CLS
PRINT "YOU ENTERED",x
GET
```

For full details of the INPUT command, see Chapter 9.

Procedure parameters

Values can be passed between procedures using parameters. In the VAT example below, the second procedure VAT1: is followed by a parameter name (p).

The last line of the first procedure PROC1: calls VAT1 with the value of x. The value of x is copied to the parameter p. VAT1: then prints out this value plus VAT at 15%.

```
PROC1:
LOCAL x
CLS
PRINT "ENTER PRICE"
INPUT x
VAT1:(x)

VAT1:(p)
CLS
PRINT "PRICE INCLUDING VAT = ",p*1.15
GET
```

- Calling a procedure which has parameters means that your first procedure can use local rather than global variables which is sometimes neater.
- Parameters differ from variables in that you cannot assign values to them. They have an external value passed to them from the calling procedure and it is then illegal to assign another value by saying, in the above example, p=p+1.
- Parameter names are typed in brackets immediately after the colon of the procedure name. Their type is specified as with variables e.g. p for a floating point value, p% for an integer and p$ for a string.

Multiple parameters

In this similar VAT example the second procedure VAT2: has two parameters.

The value of the price variable, x, is passed to the parameter p1 and the rate of VAT is also a variable, r, which is passed to the parameter p2. VAT2 then prints out the price plus VAT at the rate specified.

```
PROC2:
LOCAL x, r
CLS
PRINT "ENTER PRICE"
INPUT x
CLS
PRINT "ENTER VAT RATE"
INPUT r
VAT2:(x,r)

VAT2: (p1,p2)
CLS
PRINT p1+p2/100*p1
GET
```

- You can pass variable names, strings in quotes, or actual values to parameters.
- When you pass multiple values to multiple parameters you must make sure that the order of the values when the procedure is called corresponds to the order of parameters after the procedure name. For example:
  Procedure call - PROC:("a string",9,3.7)

Procedure name - PROC:(p$,p%,p)

The RETURN command

This VAT example differs from the previous two in that control does not end with the second procedure but returns instead to the first.

The RETURN command is used to return the value of X plus VAT at r percent - to be printed out in PROC3:. VAT3: is now just doing the calculation and not printing. This means it can be called by other procedures which need this calculation but do not necessarily need to print it.

```
PROC3:
LOCAL x,r
CLS
PRINT "ENTER PRICE"
INPUT x
CLS
PRINT "ENTER VAT RATE"
INPUT r
CLS
PRINT "PRICE INCLUDING VAT =",VAT3:(x,r)
GET

VAT3:(p1,p2)
RETURN p1+p2/100*p1
```

- Only one value may be returned by the RETURN command.
- The name of a procedure which RETURNs a value must end with the correct identifier eg. PROC$: for one which returns a string; PROC% for one which returns an integer; PROC: for one which returns a floating point value (as in this example).
- The RETURN command returns to the point where the procedure was called.