IDL Lab #2: What is an IDL Program

Name:_____

IDL Lab #2: A Sub-component of FOR 504 Advanced Topics in Remote Sensing

The objectives of this laboratory exercise are to introduce the student to

- A Simple IDL Program
- Introduction to Comments, Procedures, and Functions
- Different Variables in IDL
- The IF Statement

The tasks provided within this lab are designed to help the student better understand the practical details of programming in IDL and will help you prepare for the class assessment.

If you have problems: ASK!

Location: RS/GIS Lab

Login: XXXX
Password: XXXX

For further reading in this lab exercise, please refer to pages 17-23 (variables) and 90-96 (defining and compiling programs) in ***Practical IDL Programming***, by Liam Gumley, Academic Press, 2002.

Before you start:

Double click the ENVI icon on the desktop: 
ENVI 4.0.lnk

This starts both ENVI (*The Environment for Visualizing Images*) and the IDL (*Integrated Development Language*) programming interface

THE COMPONENTS OF A SIMPLE IDL PROGRAM

1. COMMENTS, PROCEDURES, AND FUNCTIONS

In the first lab session, you created a simple program that displayed "Hello World!" on the screen.

In this lab, we will expand on this program, but organize our code more efficiently.

We will learn the different components that make up an IDL program; including comments, procedure and functions, variables, and a typical statement.

You will find that you can write IDL programs without the following organizational structure; as I have done for many years; but it will make your life easier when writing large complex programs.

Lets us return to the simple Hello World Program:

```
1       PRO intro
2
3       ;; Two semi-colons allow you to enter comments within the program
4
5       PRINT, 'Hello World!'
6
7       END
```

On line 1, IDL is told to look in the current directory and execute the procedure called intro.pro

Now, this example is not an ordinary procedure, but this is how we write simple programs in IDL.

Normally, you would use a ***procedure*** if you wanted to produce more than one output to a query. For instance, say you had another program called circle_stats that asks this program to give it both the area and circumference of a circle when providing it with a radius.

If you only wanted one of these results, e.g. the area, then you would normally use a ***function*** instead of a procedure. Try the next two examples to check that each method works:

Example 1: Using a procedure to get the radius and circumference of a circle

```
1       PRO circle_stats
2
3       ;; Work out the area and circumference of a radius of user-defined size x
4
5       PRINT, 'Hello World!'
6
7       x = 2.0
8
9       ;;
10
11      area = (3.14159*x*x)
12      circum = (3.14159*2*x)
13
14      PRINT, 'The Area of a Circle with Radius',  x, 'is ', area
15      PRINT, 'The Circumference of a Circle with Radius', x, 'is ', circum
16
17      END
```

Example 2: Using 2 Functions to get the radius and circumference of a circle

```
1       PRO circle_stats, x
2
3       ;; Work out the area and circumference of a radius of user-defined size x
4
5       PRINT, 'Hello World!'
6
7       x = 2.0
8
9       area = ARE(x)
10      circum = CIR(x)
11
12      PRINT, 'The Area of a Circle with Radius',  x, 'is ', area
13      PRINT, 'The Circumference of a Circle with Radius', x, 'is ', circum
14
15      END
16
17      FUNCTION ARE, dummy
18      RETURN, (3.14159*dummy*dummy)
19      END
20
21      FUNCTION CIR, dummy
22      RETURN, (2*3.14159*dummy)
23      END
```

You will find that this does not work – This is because you are calling the function BEFORE it is defined. Put the functions before the procedure and try again.

Note that in the functions you cannot call the variable x; you instead have to choose a ‘dummy’ file name, which could be anything but not a IDL command word.

In addition, you need to write what variables you are receiving in the procedure:

```
1       FUNCTION ARE, dummy
2       RETURN, (3.14159*dummy*dummy)
3       END
4
5       FUNCTION CIR, dummy
6       RETURN, (2*3.14159*dummy)
7       END
8
9       PRO circle_stats, x
10
11        ;; Work out the area and circumference of a radius of user-defined size x
12
13        PRINT, 'Hello World!'
14
15       ;x = 2.0
16
17       area = ARE(x)
18       circum = CIR(x)
19
20       PRINT, 'The Area of a Circle with Radius',  x, 'is ', area
21       PRINT, 'The Circumference of a Circle with Radius', x, 'is ', circum
22
23      END
```

Although method 2 takes a little longer, splitting your tasks into functions makes it a lot easier to debug (check) your code when you make a mistake in a very big, complicated program. To enter variables into IDL before running the program, you only need to type x = ??? at the command prompt after compiling the program. Then run the program by typing the program name followed by ‘, x’ into the command prompt:

i.e.
Compile using CTRL F5
IDL> x = 2.0
IDL> circle_stats, x

Similarly, programs with more than one variable can be initiated in the same way:

IDL> a = 2

```
IDL> b = 10
IDL> prog, a, b
```

Task #1 Write a program using ***functions*** to calculate the area and perimeter of a rectangle using the length and breadth entered at the IDL prompt.

[Hint: Celsius = (Fahrenheit – 32)*5/9; and Kelvin = 273 + Celsius)

Task #2 Write a program using ***functions*** to convert both temperatures from Fahrenheit to degrees Celsius and Kelvin; and length from miles to kilometers.

[Hint: km = Miles *8/5)

2. VARIABLES

Like other high-level languages, IDL can use several variable types including integers, real numbers, and strings.

However, unlike languages like C, C++ etc; IDL is more flexible and more importantly you don't need to **define** and **fix** the variable type at the start of the program.

In IDL you can create variables at any time, which is particularly useful when using dummy variables as counters; as you don't need to define them at the very start of the program (which wastes code and can get confusing if you have multiple counters).

The different types of variables you can use in IDL are listed in Table 2.1:

Note: An unsigned integer is only positive, while a signed integer can be positive or negative.

Table 2.1 Variable Types in IDL (Modified from Gumley (2002) p18)

Variable	Description	Range
byte	Unsigned Integer	0 to 255
int	Signed Integer	-32,768 to 32, 767
uint	Unsigned Integer	0 to 65,535
long	Signed Integer	-2^{31} to $2^{31}-1$
ulong	Unsigned Integer	0 to $2^{32}-1$
long64	Signed Integer	-2^{63} to $2^{63}-1$
ulong64	Unsigned Integer	0 to $2^{64}-1$
float	IEEE floating-point	-10^{38} to $10^{38}-1$
double	IEEE floating-point	-10^{308} to $10^{308}-1$
complex	Real-imaginary pair	-10^{38} to $10^{38}-1$
dcomplex	Real-imaginary pair	-10^{308} to $10^{308}-1$

3. The IF Statement

As in most programming languages the IF statement is a very useful tool to master and it is likely that each program you write from now on will have at least one. They are particularly useful in functions if the process in the function depends on the input provided.

In this lab, we will just use IF statements in a few simple examples to demonstrate their functionality:

In summary: the IF statement assess whether a statement of set of statement are true.

In IDL programming you can write the IF statement in different ways. If there were only one test, you would write:

IF *condition* THEN *statement*

OR if you have several actions you want to do after the condition is met you would write:

```
IF condition THEN BEGIN
        statement(s)
ENDIF
```

In the scenario that there are two possible conditions for the input – i.e. yes and no; or true and false; and you want the program, to do something in each case, you would write:

```
IF condition THEN BEGIN
        statement(s)
ENDIF ELSE BEGIN
        statement(s)
ENDELSE
```

Examples of *conditions* for an IF statement include:

```
IF (X EQ 2) THEN BEGIN ;; EQ = 'Equals'
IF (Y LT 2) THEN BEGIN ;; LT = 'Less than'
IF (Z GT 2) THEN BEGIN ;; GT = 'Greater than'
```

Task#3 Write a program that prints the phrase 'very cold' to a temperature less than -10°C and 'very hot' for a temperature greater than 40°C.

[Hint: It might be easier in the functions to return an integer and in the main procedure to take this integer and in another IF function, output the phrase.]

Next week we will introduce Arrays and the FOR loop