CS 105 – Lab 4

Today's lab will give you some practice with loops and strings.

Insert your USB drive, and log into the computer. Start a couple of Terminal processes. It would be nice to use one terminal for editing, and the other for running programs. Use the `cd` command to navigate to the folder containing your USB drive. With the `ls` command, you should see folders from earlier labs, such as lab2 and lab3.

Create a new folder called lab4 with this command: `mkdir lab4`

Next, let's make a copy of the convert.pas program that you wrote last week. Enter this command: `cp lab3/convert.pas lab4/.` (make sure to include the '.' at the end). This means that your lab4 folder now has a duplicate copy of the convert.pas that is in lab3.

Enter the lab4 folder: `cd lab4` and enter the `ls` command. You should see convert.pas listed. You will make a modification to this program shortly.

Part 1: Error checking of user input

As we saw in class, there is a straightforward way to get a program to check the validity of the user's input. If the input is invalid, we can print an appropriate error message and allow the user to try again without having to quit the program and start over. For example, here is how we could check to make sure that the user enters a positive number:

```
NeedInput := true;
while NeedInput do
  begin
    write('Enter a positive integer:  ');
    readln(n);

    if n > 0 then
      NeedInput := false
    else
      writeln('Sorry, input is invalid.  Please try again.');
  end;
```

1. What data type is the value contained in NeedInput?

Modify convert.pas so that it performs error checking on the first input. It must be the capital letter F or the capital letter C. If it is any other character, this would be considered invalid. Be sure to use quotes around the F and the C, so that the computer does not think that these are variable names. Also note that some of the code that you wrote earlier now needs to be indented.

2. Write down the if-statement's condition that helps you perform the error checking.

Part 2: Practice with a for loop

Let's practice with nested for-loops to create rectangular designs on the screen. Create a new source file called box.pas. This program should first ask the user to enter two integers. The first number indicates the number of (horizontal) rows of the rectangle. The second number is the number of (vertical) columns. Both numbers need to be at least 3 for the program to show meaningful output, but you do not need to perform error checking of the input.

After the program has obtained the number of rows and columns from the user, it can draw rectangles. The first rectangle will be a *solid* rectangle of stars (asterisks). In pseudocode, the algorithm could go like this:

```
for i = 1 to the number of rows,
    for j = 1 to the number of columns,
        print a star
    print a newline character to finish the line
```

The second rectangle will be a *hollow* rectangle. Here is its pseudocode:

```
for i = 1 to the number of rows,
    for j = 1 to the number of columns,
        if i = the first or last row,
               or j = the first or last column
           print a star
        else
           print a space
    print a newline character to finish the line
```

Remember that if the body of a loop has two or more statements, you need to enclose the body with `begin` and `end;`.

Write a Pascal program that implements the above pseudocode loops. Be sure to label each rectangle in your output and print a blank line between them, so that we know where one rectangle ends and the next one begins. Here is what an example run should look like:

```
How many rows?  5
How many columns?  8
Solid rectangle:
********
********
********
********
********

Hollow rectangle:
********
*      *
*      *
*      *
********
```

Part 3: Let's experiment with strings in Pascal. Create a new source file called str1.pas. Type in the following program. Compile and run. Write down the output you see.

Program	Output

`program str1;`
 	
`var`
`  s : string;`
 	
`begin`
`  s := 'South Carolina';`
`  writeln(length(s));`
`  writeln(s[1]);`
`  writeln(s[4]);`
`  writeln(s[4] = 'T');`
`  writeln(s[length(s)]);`
 	
`  writeln();`
`  writeln(copy(s, 4, 5));`
`  writeln(pos('o', s));`
`  writeln(pos('c', s));`
`end.`	

Use the above output and more experimentation ☺ to help you answer these questions.

1. Fill in the following diagram to show the position of each character of the string ‘South Carolina’. Underneath each cell, write down the index number. Leave blank any unused cells.

2. If the 4th letter of South Carolina is a t, then why does the expression s[4] = ‘T’ return false?

3. What error message results if you try to access a character beyond the end of the string?

4. How does the `pos` function respond if the character we seek occurs in the string:
   a. More than once?

   b. Not at all?

5. Using the copy function, write an expression that returns exactly the following string parts:
   a. ‘South’

   b. ‘Carolina’

6. Add the following loops to the end of the program. Describe the output of each loop.

```
for i := 1 to length(s) do
  writeln(s[i]);
```

```
for i := 1 to length(s) do
  writeln(i, s[i]);
```