CS 105 – Lab #8 – Introduction to encryption

In today's lab, you will practice with two simple cipher systems – the Caesar cipher and the transposition cipher.

Log into the computer as usual. It's a good idea to have two terminal windows open so that you can see them side by side on the screen. Create a new folder called lab8 to hold today's files. Make sure that your current working directory is lab8 before continuing. In other words, when you enter the pwd command, the system should tell you that you are inside lab8.

Part 1: Using the chr() and ord() functions in Pascal

Behind the scenes, most cryptography requires us to manipulate the individual characters of a message as though they were numbers. So, we need a convenient way to quickly convert a character into an integer and back again. Fortunately, Pascal has this functionality.

- The chr function returns the ASCII code of a character
- The ord function returns the character corresponding to a given ASCII code.

Using nano, create a program called onechar.pas as follows:

```
(* onechar.pas - Experiment with the ord() and chr() functions
in Pascal. *)
program onechar;

var
  i : integer;

begin
  writeln(ord('a'));
  writeln(ord('b'));
  writeln(ord('c'));

  writeln(ord('A'));
  writeln(ord('B'));
  writeln(ord('C'));

  (* A loop to print ASCII codes. *)
  for i := 48 to 122 do
    writeln(i : 3, ' ', chr(i));

end.
```

Compile and run onechar.pas.

1. According to your program's output, what are the ASCII codes of 'a', 'b' and 'c'?

2. What are the ASCII codes of 'A', 'B' and 'C'? By how much does the ASCII code of a capital letter differ from its corresponding lowercase letter?

3. Write down the ASCII codes of the letters in your first name.

4. The loop at the end of the program prints out what we might call an abbreviated ASCII table. According to the output, which symbol has ASCII code 50?

5. In the final writeln statement, you should see "i : 3". What is the ": 3" telling Pascal to do? Why might this be an appropriate choice in this program? (If you are not sure, take out the ": 3" and see how the output differs.

Part 2: Caesar cipher

Now, we are ready to encrypt some messages! The first cipher system you learned was the Caesar cipher. How would you describe the essence of how to encrypt a message using a Caesar cipher?

Use nano to type in the following program, called casear1.pas. This program will encrypt the Casear way.

```
(* caesar1.pas - Caesar cipher encryption. *)
program caesar1;

const
  key = 5;

var
  plaintext, ciphertext : string;
  onechar : char;
  i, value : integer;

begin
  writeln('Enter a phrase to encode:  ');
  readln(plaintext);

  (* Remember to start the ciphertext as an empty string. *)
  ciphertext := '';

  (* Grab the next character from the message.
   * Convert it to a number.
   * Add the key to this number.
   * Convert back to a character, to insert into ciphertext.
   *)
  for i := 1 to length(plaintext) do
    begin
      onechar := plaintext[i];
      value := ord(onechar);
      value := value + key;
      onechar := chr(value);
      ciphertext := ciphertext + onechar;
    end;

  writeln();
  writeln('The ciphertext version is:');
  writeln(ciphertext);
end.
```

Compile and run the program. Choose a suitable input string to use as your plaintext message. The program will then show you the corresponding ciphertext. Write them in the blanks below:

Plaintext message = _____

Ciphertext equivalent = _____

Next, it would be nice if we could go the other way: write a decryption program for the Caesar cipher. That is what we will do next. Our next program, caesar2.pas, will perform the decryption. When we run caesar2, we will type in the ciphertext, and the program should be able to figure out the original plaintext message.

It turns out that Caesar decryption is so similar to Caesar encryption, that we can reuse a lot of the code you have already written. So, let's make a copy of casesar1.pas, call it caesar2.pas, and make the necessary edits. Here are the steps:

- Enter this command at the terminal: cp caesar1.pas caesar2.pas
- Use nano to open caesar2.pas for editing.
- In the code, find the assignment statement that adds the key to each character of the plaintext. Change the plus sign to a minus.
- Throughout the program, reverse the roles of the variables plaintext and ciphertext. In other words, every time you see the variable plaintext, change this to ciphertext, and vice versa.
- Change the wording of the input prompt so that we ask for the ciphertext. Similarly, at the end of the program, indicate that we are going to output the recovered plaintext message.

Save, compile and run your program. As input, use the ciphertext you obtained from caesar1.pas that you wrote down earlier. Verify that caesar2 prints out the correct original message.

Part 3: The Eavesdropper

How hard would it be to break the Caesar cipher? That is what we are going to find out. Imagine that you have intercepted a ciphertext message. You do not know the key. Therefore, you would need to perform trial and error with many keys until you recover the message.

Let's create a new program caesar3.pas that is based on caesar2.pas. Recall that casear2.pas was able to decrypt a ciphertext message that had used a Caesar cipher. But that task was relatively straightforward because we knew that the key was 5, and we made that a constant in the program. But in caesar3.pas, we cannot make this assumption. We need to try several possible values for the key.

Here are the steps to follow:

- Go back to caesar1.pas and change the key from 5 to a different value. Pick any other integer from 1 to 25.
- Edit caesar2.pas to change its key to the same value you used in caesar1.pas.
- Compile both caesar1.pas and caesar2.pas.
- Run caesar1, using a new message. From the output, write down the ciphertext here:

- Run caesar2, using the ciphertext as input. Verify that the plaintext output is correct.
- Find another student in the class who is NOT sitting next to you, and trade your ciphertext message with that person. In other words, give the other person your ciphertext message only, and take down their ciphertext message. Do not reveal what your key is.
- At the terminal enter this command: cp caesar2.pas caesar3.pas
- Make the following changes to caesar3.pas:
    - Make key an integer variable instead of a constant.
    - Immediately after the readln statement, create a loop that begins as "for key := 1 to 25 do". Enclose the rest of the code of the program inside the body of this loop. Use begin and end to enclose the code. Indent the body.
    - At the end of the loop, replace the three writeln statements, with this single writeln statement:

```
writeln('key = ', key:3, ' ', plaintext);
```

Save, compile and run caesar3.pas. From the output, what can you deduce about the key and the plaintext message?

Based on the implementation and results of caesar3.pas, do you think that the Casear cipher is a secure cipher system? Why or why not?

Part 4: Transposition cipher

When we work out a transposition cipher by hand, it is convenient to enter the letters of our message in a grid. The number of columns of the grid is the key that we have decided upon. Let's suppose we want a key of 4. Then, we draw 4 columns. The number of rows in the grid is entirely dependent on the length of the message. If the final row is incomplete, we pad the empty cells with the letter z.

To encrypt a message, enter the plaintext message row by row. The ciphertext is obtained by taking the letters out column by column.

To decrypt a message, you know how the ciphertext was derived, so all you really need to do is to recreate the grid. Enter the ciphertext letters column by column. Then, the plaintext can be read out row by row.

To practice, using a key of 4, decrypt this message. To reduce transcription errors, I have included spaces after every 5 characters. You should not enter these spaces in the grid, because they actually have no bearing on where words really begin and end:

WEETJ ENMED ZAUOE ONTEN IFLNE HSDCO EZ.

Next, using a key of 4, encrypt the phrase FURMAN UNIVERSITY. Do not include the space. What is the ciphertext?

Now, let's implement a transposition cipher in Pascal. Using nano, create a new source file called trans.pas. Here is the program to type in:

```
(* trans.pas - Transposition cipher with key 5. *)

var
  HowManyZ, column, DesiredColumn, i : integer;
  plaintext, ciphertext : string;

begin
  write('Enter plaintext:  ');
  readln(plaintext);

  (* Pad with the letter z if length is not a multiple of 5. *)

  howManyZ := 5 - length(plaintext) mod 5;
  if howManyZ = 5 then
    howManyZ := 0;

  for i := 1 to howManyZ do
    plaintext := plaintext + 'z';

  (* Grab the the 1,6,11,16, ..., then the 2,7,12,17, ... *)
  ciphertext := '';
  for DesiredColumn := 1 to 5 do
    for i := 1 to length(plaintext) do
      begin
        column := i mod 5;
        if column = 0 then
          column := 5;
        if column = DesiredColumn then
          ciphertext := ciphertext + plaintext[i];
      end;

  writeln('Ciphertext is: ', ciphertext);

end.
```

Save, compile and run your program. What is the resulting ciphertext when you enter the following message as plaintext?
LIVESTOCKFEED