Homework 4

Assigned: 10/19

Due: 10/30, 11:59pm

Possible Points: 100

Submit on Canvas a zip or zipped tar file containing your solution code for each problem (just the cpp files, please don't submit the solution/project files), turn in one cpp file for each problem. If a problem has multiple parts, it should still be one file. Each problem specifies what you should title the file containing your solution so that the autograder can find it. Incorrectly named programs will not be graded and thus will get 0 points for the problem.

Since the assignment will be graded using an automated grading system you should have your program's output match the sample output shown. If your program fails to compile using Visual Studio 2015 it will recieve zero points. If your program compiles but fails to run properly (runtime errors or wrong output) points will be deducted based on how correct the program is.

If you have any questions or concerns about the assignment do not hesitate to post on the discussion board on canvas, send us mail on canvas, or see us after class or during office hours.

Students are expected to write their source code from scratch, those who copy code from each other or online (including from Canvas examples) will be reported for cheating. We will also look at your code so definitely don't just hard-code the answer and print it.

1. Encryption: 10 Points

File: `encryption.cpp`

For this assignment you'll write a simple substitution cipher known as ROT13. This method works by shifting the letters of the source text by 13 letters to produce the encrypted output, successfully garbling the output but providing no real security. The signature for the function is given below, you'll take in a `std::string &` and should "encrypt" it in place. The signature of the function you should write is:

```
void encrypt(std::string &str);
```

You shouldn't need to perform any memory allocation to solve this problem. You may also find the ASCII table useful for this assignment. Your encryption function should also support upper and lower case letters and keep them distinct from each other, for example 'A' should encrypt to 'N' and 'a' should encrypt

to ‘n’. The input to your program will always just be lowercase and uppercase letters so don’t worry about handling numbers or punctuation or so on.

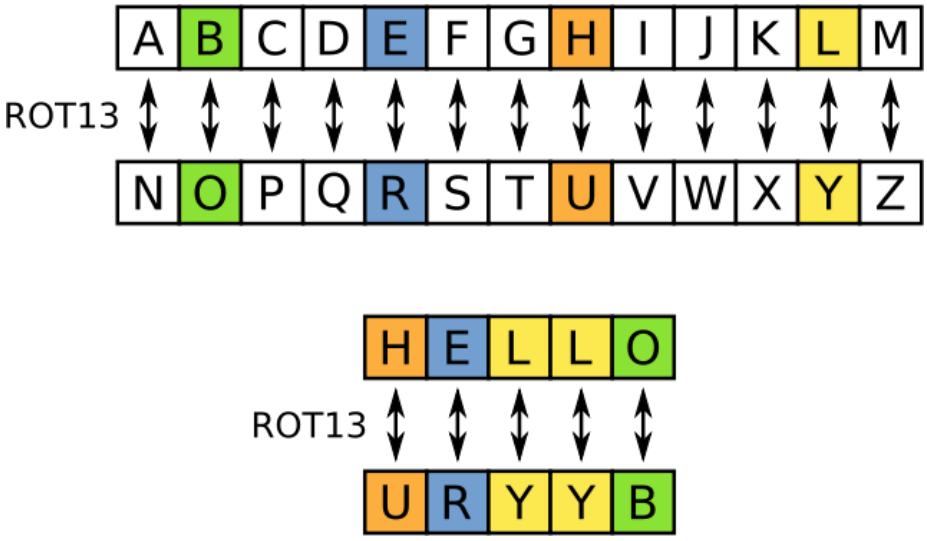

Figure 1: Example ROT13 encryption from Wikipedia

Example Input:

```
5
HelLozzZ
AbCdEfgHijklMnOpQrStuVwXYZGHAwdaowjwa
HiHowAreYoU
FunKadElic
SupErcaLifRagiliSticExpiAlidOcious
```

Expected Output:

```
Case 0:
UryYbmmM
Case 1:
NoPqRstUvwxyZaBcDeFghIjKLMTUNjqnbjwjn
Case 2:
UvUbjNerLbH
Case 3:
ShaXnqRyvp
Case 4:
FhcRepnYvsEntvyvFgvpRkcvNyvqBpvbhf
```

---

2. Parsing Integers: 30 Points

File: `parse_int.cpp`

In this problem you will write a function to parse an integer from a string that is **robust** (including overflow) to invalid integers. You **cannot** use any functions provided by the standard library for parsing an integer from a string, including `std::cin >>` to an integer variable. Your function will receive the string as a `const std::string &` and write the parsed integer to an integer out parameter. You'll need to use an out parameter because the function will return a bool (true or false) to signal if parsing was successful or not. The signature of the function you're writing should be:

```
bool parse_int(const std::string &str, int &val);
```

Your parser should be able to handle positive and negative numbers along with slightly odd styles, e.g. `+40` should parse as positive 40. You should read the string from `std::cin` into a `std::string`. In the case that you encounter something that is not a digit (recall the functions in the `cctype` header) you should stop parsing and return false to indicate parsing failed. In main you'll then see if `parse_int` returned true or false, if it returns true simply print out the number, if it returns false print out 'Parsing failed'.

How can we determine the value of an integer from a `char`? Recall that we saw that ASCII characters were just numbers, so we can subtract them from each other, for example: `'2' - '0' = 2`.

Example Input

```
7
1542
-9852
+92
10
hello!
10oops123
123456789012354444444444444444
```

Expected Output:

```
Case 0:
1542
Case 1:
-9852
Case 2:
92
Case 3:
10
Case 4:
```

```
Parsing failed
Case 5:
Parsing failed
Case 6:
Parsing failed
```

---

3. Label Generator: 20 points

File: `label_generator.cpp`

For certain applications, it is useful to be able to generate a series of names that form a sequential pattern. For example, if you were writing a program to number figures in a paper, having some mechanism to return the sequence of strings "Figure 1", "Figure 2", "Figure 3", and so on, would be very handy. However, you might also need to label points in a geometric diagram, in which case you would want a similar but independent set of labels for points such as "P0", "P1", "P2", and so forth.

If you think about this problem more generally, the tool you need is a label generator that allows the client to define arbitrary sequences of labels, each of which consists of a prefix string ("Figure" or "P" for the examples in the preceding paragraph) coupled with an integer used as a sequence number. Because the client may want different sequences to be active simultaneously, it makes sense to define the label generator as an abstract type called `LabelGenerator`. To initialize a new generator, the client provides the prefix string and the initial index as arguments to the `LabelGenerator` constructor. Once the generator has been created, the client can return new labels in the sequence by calling `next_label()` on the `LabelGenerator`.

The constructor's function signature should be:

```
LabelGenerator(const std::string &prefix, int start);
```

The signature of `next_label` should be:

```
std::string next_label();
```

Your implementation should match these function signatures, you should not change them. As a hint your label printing should behave similar to the following code:

```
LabelGenerator point_numbers("P", 0);
for (int i = 0; i < 3; i++) {
  std::cout << point_numbers.next_label() << " ";
}
```

should print `P0 P1 P2`.

Input and output

The input for each case will consist of two lines. On the first line is a prefix string which may contain spaces (including leading or trailing spaces). On the second line are two integers which are the initial and the final values of a sequence.

Your program will print the sequence of labels on one line, adjacent terms separated by a space. To turn an integer into a string, you can use the `std::to_string()` function defined in `<string>` (see cppreference.com for more information).

Example Input:

```
3
Figure
0 3
Figure_
1 2
Fig.
0 0
```

Expected output:

```
Case 0:
Figure0 Figure1 Figure2 Figure3
Case 1:
Figure_1 Figure_2
Case 2:
Fig.0
```

---

4. Simple Matrix and Vector Structs: 40 Points

File: `matrix_vector.cpp`

In this assignment you will create two `structs` called `Vec3` and `Mat3` which represent 3D vectors and matrices, and write functions that act on these `structs`. **This problem is time consuming thus plan properly, structure your code with functions you reuse systematically. Design your solution before you start implementing it.**

Below is the list of functions you will need to write. Please ***do not*** change the function signatures (that is, do not change the functions' parameter and return types).

- `Vec3 read_vec()`: read three `doubles` from the standard input, use them to construct a `Vec3` and return it. You can assume the three `doubles` will

be given in the standard input as three numbers separated by spaces.

- `Mat3 read_mat()`: read nine `double`s from the standard input, use them to construct a `Mat3` and return it. We use the row-major ordering for the nine numbers. That is, the first three numbers make up the first row of the matrix, the next three make up the second row, and so on.
- `void print(Vec3 v)`: print the given vector to the standard output. The format is `(x, y, z)`.
- `void print(Mat3 m)`: print the given matrix to the standard output. The format is `[ a, b, c, d, e, f, g, h, i ]`. The elements are in row-major ordering (see above for what this means).
- `Vec3 add(Vec3 u, Vec3 v)`: add two vectors and return the result.
- `double dot(Vec3 u, Vec3 v)`: return the dot product of two vectors.
- `double length(Vec3 u)`: return the length (or magnitude) of the input vector.
- `Mat3 transpose(Mat3 m)`: return the transpose of the given matrix
- `Vec3 row(Mat3 m, int i)`: return the i-th row of the given matrix (indexing starts from 0)
- `Vec3 col(Mat3 m, int i)`: return the i-th column of the given matrix (indexing starts from 0)
- `Vec3 multiply(Mat3 m, Vec3 u)`: multiply a matrix with a vector and return the resulting vector.
- `Mat3 multiply(Mat3 m1, Mat3 m2)`: multiply two matrices (in the order given) and return the resulting matrix.

Each case in the input will start with an operation, which is a string telling you what to compute for the current case. The operation can be one of { `add`, `dot`, `length`, `transpose`, `row`, `col`, `multiply` }. The operands to the operation come next. Some operations require two operands (`add`, `dot`, `multiply`), others work with one operand. A matrix operand will start with an `M` followed by nine numbers. A vector operand will start with a `V` followed by three numbers. You can safely assume all the inputs will be valid (you do not have to handle invalid inputs).

For each case, print the result to applying the given operation on the two given operands, using the `print` functions that you have written.

Example Input

```
8
add
V 1 2 3
V 2 3 4
length
V 0 3 4
dot
V 1 1 2
```

```
V 2 2 1
transpose
M 1 2 3 4 5 6 7 8 9
multiply
M 1 2 3 4 5 6 7 8 9
V 1 1 1
multiply
M 1 2 3 4 5 6 7 8 9
M 9 8 7 6 5 4 3 2 1
row
M 9 1 2 8 7 3 4 6 5
1
col
M 9 1 2 8 7 3 4 6 5
2
```

Example Output

```
Case 0:
( 3, 5, 7 )
Case 1:
5
Case 2:
6
Case 3:
[ 1, 4, 7, 2, 5, 8, 3, 6, 9 ]
Case 4:
( 6, 15, 24 )
Case 5:
[ 30, 24, 18, 84, 69, 54, 138, 114, 90 ]
Case 6:
( 8, 7, 3 )
Case 7:
( 2, 3, 5 )
```