Fundamentals of Programming, ETSII

Practice: Arrays, strings and structs

1 Let `typedef array<int,N> TFreqs;` be a type to contain frequencies, number of repetitions. Each frequency corresponds to a number: the index the cell is at, itself. For example, if we have `TFreqs f`, then `f[2] = 3` would mean the number 2 is repeated 3 times. This algorithm is supposed to work within the the range $[0..N)$ of numbers. Build a subprogram `TFreqs freqsOf(string s)` that receives a string with numbers [0..9] (one-digit numbers) and returns the corresponding `TFreqs` array. Build also `void printFreqs(TFreqs f)` to print from `main()` print the output of `freqsOf`. For example:

```
writeFreqs(
 freqsOf("1 8 7 3 4 8 5 9 5 0 0 4 8 4 5 3 2 8"));
//      -> 0:2 1:1 2:1 3:2 4:3 5:3 6:0 7:1 8:4 9:1
```

HINT: To take the numbers from the string, a complete traverse of the string is necessary. Each time a number is there, the integer value to add to our `TFreqs` is: `n = s[i] - '0';`

2 We want to divide a string in its tokens, so we need to build a function `TTokens tokens(string s)`, that for example, for the string `" a number of times, 3, is good"` would return: `{{"a", "number", "of", "times,", "3,", "is", "good"}}`. The array we will need here has to have in each cell, the `word`, `n` info. So each cell has to be a structure. An open array could be a solution, but not necessary here, since as you traverse the array from 0 up to its end, the moment you find an empty word, it tells you do not have any more words in the rest of the array.

3 In order to do mathematical operations with complex numbers we can define the next type:

```
struct TComplex {
  float rPart, iPart;
};
```

Remember that to add 2 complex numbers you have to add each part separately. Same applies for subtract. For multiplying 2 complex numbers you have for the real part of the multiplication:
`rPart1*rPart2 - iPart1*iPart2`
for the imaginary part:
`rPart1* iPart2 + rPart2*iPart1`

For the division we have: `c1*conjugate(c2) / modulus(c2)`

Based on that, write functions to operate this numbers:

```
float modulus     (TComplex a);
    // sqrt(rPart*rPart + iPart*iPart)
TComplex conjugate(TComplex a);
    // (rPart, -iPart)
TComplex add      (TComplex a, TComplex b);
TComplex subt     (TComplex a, TComplex b);
TComplex mult     (TComplex a, TComplex b);
TComplex mult     (TComplex a, float x);
TComplex div      (TComplex a, TComplex b);
```

Call this functions from the `main()` checking their correct behaviour. Try, for example with $c_1 = 78 + 104i$ and $c_2 = -3 + 2i$, and do $c = c_1 + c_2$. After calling the `add()` function check that $c - c_2 == c_1$. Do the same with $c = c_1/c_2$, then check that $cc_2 == c_1$. Check that, for example: $c_1c_2 \to -442 - 156i$ and $c_1/c_2 \to -2 - 36i$.

HINT: : A useful procedure here is

```
void printComplex(TComplex);
```

Then to check our functions, you would only need to do:

```
TComplex a = {78, 104}, b = {-3, 2};
printComplex(add(a, b));
printComplex(subt(a, b));
// etc
```

After all of this an easy and interesting function is:

```
TComplex pow(TComplex a, int n);
```

Base it on `mult()`. What if `n` was negative? (you would need to invert $1/r$ before returning the power). Check $c_2^7 \to 4449 - 6554i$

4 We want to work with polynomials $(a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0)$. So we define a the type `TMono` as a structure able to contain a number real `coef` and an integer `n`. A polynomial `TPoly` would then be an array of `TMono`. Build the types and a subprogram to `printPoly` to print them nicely on the screen. After that build another subprogram (procedure or function) to take the derivative `TPoly derivative(TPoly p)` of a polynomial. Check the derivative of $3x^2 + 2x - 4$ is $6x + 2$.

HINT: Especially for large `N`, it would be important to stop traversing at the first 0 `coef` assuming and ensuring, then, the rest of monomials from that monomial on, are all 0.

TO DO AT HOME

5 To generalise the frequencies calculator program, build an array `TFreqs` of structures each one with the number and its frequency: `struct TFreq {int n; int freq;};` For example, if we have `TFreqs f`, then `f.freqs[2] = {35, 3}` would mean the number 35 is repeated 3 times. The capacity of the array is limited to `N` maximum different numbers. Build a subprogram `TFreqs freqsOf(string s)` that receives a string with natural numbers and returns the corresponding `TFreqs` array. Build also `void printFreqs(TFreqs f)` to print from `main()` print the output of `freqsOf`. To know where the data ends inside our array, a frequency 0 indicates that the rest of elements of `TFreqs` do not contain values. For example:

```
% \begin{lstlisting}[basicstyle=\scriptsize\ttfamily]
writeFreqs(freqsOf(" 10 800 4 3 4 10"));
// -> 10:2 800:1 4:2 3:1
```

HINT: To take the numbers from the string, a complete traverse of the string is necessary. Each time a number is there, the integer value to add is: `temp = 10*temp + s[i] - '0';`; when the no number is there and after the complete traverse we have to add the `temp` integer found to the `TFreqs` and reset `temp` to 0 again.