**COSC 1436, Dr. Pershwitz**

**Assignment 9a: 32 points**

---

## Write a Program: DNA Sequence Revisited (yet again)

Use Assignment 6b as your starting point.

Every DNA sequence consists of four nucleotides: Adenine, Thymine, Cytosine, and Guanine, referred to by the first letters of their chemical names (A, T, C, and G). I have provided an entire DNA sequence in the file **dnaSequence.txt.**

This time, in addition to computing the number of occurrences for each nucleotide, your program also needs to determine the position of each occurrence in the sequence and provide this information when asked.

You need to read the entire sequence into an array. Since you don't know how many nucleotides there are in the sequence you will need to grow your array dynamically in 100-element increments.

You also need to have an array for each nucleotide that holds positions of its occurrences in the sequence. Since you don't know how many occurrences there are of each nucleotide these arrays will need to grow dynamically also.

After you read and process information from the input file your program needs to present the user with the menu to select a nucleotide. Once a valid nucleotide is selected you should display the number of occurrences for that nucleotide to the user and ask to enter an occurrence. Once a valid occurrence is entered you should display the position of that occurrence in the original sequence.

You should also provide a report of the number of each nucleotide within the sequence, and the percent each nucleotide makes up of the total. You should include your output file in the submission in addition to our usual submission format (console output + code). This part has not changed from 6b but your implementation will now use arrays.

You should define the following 4 functions (5 if you count 2 overloaded functions):

1. A function responsible for growing your arrays. The function takes a pointer to a dynamically allocated array and returns a pointer to an array 100 elements larger with the data from the original array. Since you need to grow your nucleotide array holding chars and your nucleotide position arrays holding ints, you will need to overload this function for these two types.
2. A function that reads the DNA sequence from a file with a given filename. The function takes a filename, reads the nucleotide sequence from the file into a dynamically allocated array and returns a pointer to this array.
3. A function that outputs a single formatted line for a nucleotide to a given output stream. This function is the only place in the code that knows how the line is formatted. **Hint:** you have to use `ostream&` type parameter to pass output streams, including `cout`.
4. A function that outputs the entire report to a given output stream. This function needs to call function 3 to output a line for each nucleotide.

**Your program should have the following:**

- Include 4 comment lines at the top: description of the program, author, section, and date. (1 point)
- Create your variables, use the appropriate type, name them appropriately, and remember to not leave them uninitialized. (1 point)
- Create named constants for what's appropriate. Decide the scope for each constant. Use the standard convention for constant names. (2 points)
- Create five parallel arrays: (3 points)
    1. Holding nucleotide names
    2. Holding nucleotide characters as represented in the input file
    3. Holding the number of nucleotide occurrences in the DNA sequence
    4. Holding pointers to arrays with positions of nucleotide occurrences
    5. Holding the sizes of these arrays
- Read the DNA sequence from the provided file. Function 2 should do this work for you. (1 point)

- Process the DNA sequence by storing the position of each nucleotide occurrence in a dynamically created array for each nucleotide. Pointers to these arrays are maintained by array 4 above. As these arrays fill up grow them as needed using an overloaded Function 1. (3 points)
- Output the same report as in 6b to the output file. Use function 4 to do it. If the output file fails to open output an error message and the report to the console and keep going. (1 point)
- Display a menu to let the user select a nucleotide. Use a loop to display enumerated nucleotide names and ask the user to enter a number. Validate user input using a loop. If an invalid number is entered output an error message and ask the user to enter a valid number. (2 points)
- Once a valid nucleotide is selected output the number of occurrences of that nucleotide and ask the user to enter an occurrence. Validate user input using a loop. If an invalid occurrence is entered output an error message and ask for a valid one. (2 points)
- Once a valid occurrence is entered output its position in the DNA sequence. (1 point)
- Don't forget to delete any dynamic arrays you have been using. (2 points)
- Define the 4 functions as described above.
  Function 1: (5 points)
    - The function takes a pointer to, and the size of, a dynamically created array it needs to grow
    - It dynamically creates an array 100 elements larger and copies the data from the original array to the new array
    - It then returns a pointer to the new array.

  **Note 1:** In addition to returning the new array don't forget that the calling code also needs to know about the new size.

  **Note 2**: Don't forget to delete the memory for the original array, otherwise you get a memory leak.

  **Note 3:** Do not pass a pointer to a statically created array to this function. An attempt to delete a statically created array will not end well.

**Note 4:** It's best if you use the same function to create your dynamic arrays initially by passing it a null pointer and the size of 0.

Function 2: (3 points)

- Open the file for input and make sure the operation was successful. Output an error message if there was an error opening the file.
- Read in nucleotides from the file and store them in a dynamically created array
- As you read from the file and the array gets filled up call an overloaded function 1 to grow it as needed.
- Return a pointer to the array holding the nucleotides or a null pointer if the file failed to open.

**Note 1**: Do not forget that the calling code also needs to know about the number of nucleotides that was read from the file.

**Note 2**: Once read from the input file, the nucleotide sequence array is not supposed to be changed, so function 2 should be the only function allowed to modify this array.

Function 3: (2 points)

- The function takes all needed parameters to output a single line of formatted output to an output stream. The output stream is also passed to it as a parameter. The type of this parameter needs to be `ostream&`.
- How the output is formatted is the responsibility of this function. If you need to change it you shouldn't have to go to any other place in the code.

**Note**: If you have this function done correctly in 6b it doesn't need to change.

Function 4: (3 points)

- The function takes all needed parameters to output the entire report to an output stream. Similarly to function 3, the output stream is also passed to it as a parameter of type `ostream&`.
- What goes into the report is the responsibility of this function. If you need to change anything about the report (other than the formatting of the lines – that's the responsibility of function 3) you shouldn't have to go to any other place in the code.

**Note**: The implementation of this function will change from 6b since you are using arrays now.

**Notes:**

- When you run your program, you should test invalid user input for both nucleotides and occurrences. You should also test your program when the output file cannot be opened. Insufficient console output is a 1-point deduction.
- Pay attention to where you create and how you initialize your variables. Unsafe code is a 1-point deduction.
- Comment your code. Uncommented code is a 1-point deduction.
- Remember to include your output file in the submission. An insufficient submission is a 1-point deduction.
- Style: reference is a part of the type and needs to go with the type, e.g., `int& x` rather than `int &x`. Even though the language allows both notations, the latter is more error-prone, needs to be avoided, and will carry a 0.25-point deduction if used.
- Style: pointer is also a part of the type and needs to go with the type, e.g., `int* x` rather than `int *x`. Even though the language allows both notations, the latter is more error-prone, needs to be avoided, and will carry a 0.25-point deduction if used.

**Example Output:**

**File output:**

```
DNA sequence analysis:
29782 nucleotides in the sequence

Sequence breakdown:
Adenine:  8892      29.86%
Thymine:  9581      32.17%
Cytosine: 5462      18.34%
Guanine:  5847      19.63%
```

**Console output:**

**Run 1**

```
Nucleotides in the sequence:
```

```
1. Adenine
2. Thymine
3. Cytosine
4. Guanine
Enter your choice: 0
ERROR: Valid choices are from 1 to 4
Enter your choice: 5
ERROR: Valid choices are from 1 to 4
Enter your choice: 4
There are 5847 occurrences of Guanine
Enter occurrence: 1
occurrence 1 of Guanine is at position 1 of the
sequence
```

**Run 2**

```
Nucleotides in the sequence:
1. Adenine
2. Thymine
3. Cytosine
4. Guanine
Enter your choice: 1
There are 8892 occurrences of Adenine
Enter occurrence: 9000
ERROR: Valid occurrences are from 1 to 8892
Enter occurrence: 0
ERROR: Valid occurrences are from 1 to 8892
Enter occurrence: 2225
occurrence 2225 of Adenine is at position 7165 of the
sequence
```

**Run 3**

```
Error opening file /report.txt
DNA sequence analysis:
29782 nucleotides in the sequence

Sequence breakdown:
Adenine:  8892      29.86%
Thymine:  9581      32.17%
Cytosine: 5462      18.34%
Guanine:  5847      19.63%
```

```
Nucleotides in the sequence:
1. Adenine
2. Thymine
3. Cytosine
4. Guanine
Enter your choice: 2
There are 9581 occurrences of Thymine
Enter occurrence: 5
occurrence 5 of Thymine is at position 10 of the
sequence
```