ENGG1340 Programming Technologies / COMP2113 Computer Programming II

Assignment 4

Deadline: 4 May (Saturday), 2024 23:59

If you have any questions, please post to the Moodle discussion forum on Assignment 4.

Total marks: 100 marks

A maximum of 5 marks will be deducted if you fail to follow the submission instructions strictly.

General Instructions

Read the instructions in this document carefully.

In this assignment you will solve 3 tasks, and testers would be used to automatically test your submitted programs. If your submitted files and program outputs do not conform to the instructions given here, your programs cannot be evaluated, and **you will risk losing marks totally**.

Coding environment

You must ensure that your program can compile, execute and generate the required outputs on our standard environment, i.e., the CS Linux servers (`academy*`). C++ programs must be compiled with the compilation flags `-pedantic-errors -std=c++11`, while C programs must be compiled with the compilation flags `-pedantic-errors -std=c11`.

Here is a summary of how you should compile and test your work.

Problem	Provided files	Required files	Compilation	Testing method
1	*(none)*	`1.cpp`	`g++ -pedantic-errors -std=c++11 -o` *`[executable name]`* `1.cpp`	`./[executable name] < [test case input]`
2	*(none)*	`2.c`	`gcc -pedantic-errors -std=c11 -o` *`[executable name]`* `2.c`	`./[executable name] < [test case input]`
3	`3.h`	`3.c`	`gcc -pedantic-errors -std=c11 -o` *`[executable name] [test_main_program.c]`* `3.c`	`./[executable name]`

As a programmer/developer, you should always ensure that your code can work perfectly as expected on a target (e.g., your client's) environment, not only on yours.

While you may develop your work on your own environment, you should always try your program (compile & execute & check results) on our standard environment before submission. If your submitted files cannot be compiled using the compilation method stated above on our standard environment, or if they cannot be executed on our standard environment, or if they cannot produce output that pass the `diff` test as explained below, your program may not be graded.

Input and output format

For tasks that require **reading input from user**, your answer should read from the **standard input**. Also, your program should **print** through the **standard output**. If you failed to follow this guide, the tester may not be able to give a score for your program. Additionally, you should strictly follow the sample output format, otherwise, your answer might be considered as wrong.

Sample test cases

Sample test cases are provided for each problem. Note that the test cases may or may not cover all boundary cases for the problem. It is also part of the assessment whether you can design proper test cases to verify the correctness of your program. We will also use additional test cases when marking your assignment submission.

Take Problem 1 as an example. The sample input and the expected output are given in the files `input1_1.txt` and `output1_1.txt`, respectively. Suppose that you compiled your program to an executable named `1`, do the followings at the command prompt of the terminal to check if there is any difference between your output and the expected output.

```
./1 < input1_1.txt > myoutput.txt
diff -Bw myoutput.txt output1_1.txt
```

Testing against the sample test cases is important to avoid making formatting mistakes. The additional test cases for grading your work will be of the same formats as the sample test cases. Note that for this assignment, the flag `-Bw` will be used when checking the output of your programs. This will ignore leading/trailing spaces and empty lines when comparing outputs.

Evaluation

Your code will be auto-graded for technical correctness. In principle, we use test cases to benchmark your solution, and you may get zero marks for not being able to pass any of the test cases. Normally partial credits will not be given for incomplete solution, as in many cases the logic of the programs are not complete and an objective assessment could be difficult. However, your work may still be considered on a case-by-case basis during the rebuttal stage.

Academic dishonesty

We will be checking your code against other submissions in the class and from the Internet for logical redundancy. Please be reminded that no matter whether it is providing your work to others, assisting others to copy, or copying others will all be considered as committing plagiarism, and we will follow the departmental policy to handle such cases. Please refer to the course information notes for details.

Use of generative AI tools, like ChatGPT, is not allowed for the assignment.

Submission

Name your files as the following table shows and put them together into one directory. Make sure that the folder contains only these source files and no other files. **Compress this directory as a `[uid].zip` file where [uid] is your university number** and check carefully that the correct files have been submitted.

You are advised to prepare the zip file on the standard environment, then transfer the zip file to your own machine through FTP and submit it on Moodle. You should not transfer content of your files through copy-and-paste as the process is error prone. We suggest you to download your submitted file from Moodle, extract them, and check again for correctness. **You will risk receiving 0 marks for this assignment if you submit incorrect/incomplete files.** Resubmission after the deadline is not allowed.

Filename	Description
`1.cpp`	Problem 1
`2.c`	Problem 2
`3.c`	Problem 3

Late submission

If submit within 3 days after the deadline, there will be 50% mark deduction. After that, no mark.

Getting help

You are not alone! If you find yourself stuck on something, post your question to the course forum. We want this assignment to be rewarding and instructional, not frustrating and demoralizing. But we don't know when or how to help unless you ask.

Discussion forum

Please be careful not to post spoilers. Please don't post any code that is directly related to the assignments. However, you are welcome and encouraged to discuss general ideas on the discussion forums. If you have any questions about this assignment you should post them in the discussion forums.

Problem 1: Bigram analyzer

A character bigram is a sequence of two adjacent characters from a string. For example the word "bigram" has 5 character bigrams: `bi`, `ig`, `gr`, `ra`, `am`.

Write a C++ program reads words from user input until a single character word `!` is received. Then count all character bigrams in the words and output all words that consists of the most frequently occurring bigram.

IMPORTANT: For this question, you must utilize the Standard Template Library for the task. No mark would be given if your program does not use STL.

Input:

- The program should read words from user input until a single character word `!` is received.
- There will only be one word on each line of input. Each word will only consist of letters in lower case.
- The last input line will always be a single `!`.

Output:

- The program should find the most frequently occurring character bigram, then output all words that consists of that character bigram.
- The words must be listed in alphabetical order.

Requirements and assumptions:

- There is no character bigram for words with only one character.
- When there are more than one frequently occurring character bigrams, choose the character bigram that is first encountered as the most frequently occurring character bigram.
- You must utilize the Standard Template Library in your program.

Sample Test Cases

1_1

Input:

```
standard
template
library
!
```

Output (bigram is `ar`):

```
library
standard
```

1_2

Input:

```
she
sells
seashells
on
the
seashore
!
```

Output (bigram is `sh`):

```
seashells
seashore
she
```

1_3

Input:

```
a
e
i
o
u
!
```

Output (It's empty):

```

```

Problem 2: Run-length encoding

Run-length encoding encodes a string by replacing consecutive repeating characters with a `(character, count)` pair. For example, the string `aabbbcccc` can be encoded as `a2b3c4` (i.e., `a` repeated 2 times, `b` repeated 3 times, and `c` repeated 4 times).

The compression rate of the encoding can be calculated by $\frac{encoded\ length}{original\ length}$.

Write a **C program** that reads one single word from user. It then outputs the run-length encoded string of the input word, then outputs the compression rate of the encoding.

IMPORTANT: For this question, you must submit a C program. No mark would be given if you submit a C++ program.

Input:

- The program should read one single word from user.

Output:

- The program should produce the run-length encoded string of the input word, then outputs the compression rate of the encoding in three decimal places. Refer to the sample output for the required format.
- You need to use **double** data type to calculate the compression rate, then use `printf` with the `%f` specifier (with an appropriate format options) to output the compression rate in three decimal places.

Assumptions:

- The input word will only consist of letters. It has a maximum length of 500.

Sample Test Cases

2_1

Input:

```
aabbbcccc
```

Output:

```
a2b3c4
0.667
```

2_2

Input:

```
ab
```

Output:

```
a1b1
2.000
```

2_3

Input:

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Output:

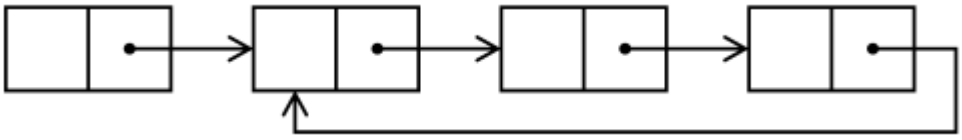
```
a80
0.037
```

Problem 3: Linked list cycle length

The following C struct and typedef defines type `ListNode`, which could be used as a node for a linked list implementation in C.

```c
struct listNode {
    int data;               // Data stored in the node
    struct listNode* next;  // Pointer to the next node
};
typedef struct listNode ListNode;
```

A cycle would be formed if the `next` pointer of a node points to an earlier node in the linked list. The figure below shows an example linked list with a cycle.



Write a **C program** that provides the function `findAverageCycleLength()` that analyze **an array of linked lists** and find the average length of all cycles found in the linked lists.

Do not include the `main()` function in your program. Please read the following specifications carefully.

IMPORTANT: For this question, you must submit a C program. No mark would be given if you submit a C++ program.

Input:

- Your program must include the provided header file `3.h`, which provides the definition of type `ListNode`. Your program should then provides the definition of the function `findAverageCycleLength()`. The function takes 2 input parameters, `listArray` and `n`:
    - Parameter `listArray` is an array of `ListNode`, each corresponding to the head of a linked list.
    - Parameter `n` indicates the number of elements in array `listArray`.

Output:

- The function must return the average length of all cycles found in the linked lists. If there are no cycle found in the array of linked list, the function should return 0.

Algorithm:

- You can use the address of the pointer to a node to identify a linked list node. Do not use the data to identify a node as there could be multiple nodes in the list that hold the same data.
- The Floyd's cycle-finding algorithm may be one of the most easiest methods for the task. You may use any other method.

Assumptions:

- There is no limit for size of the array. `n` would be 0 if the array is empty.
- There is no limit for the number of nodes in each of the linked lists.

Note:

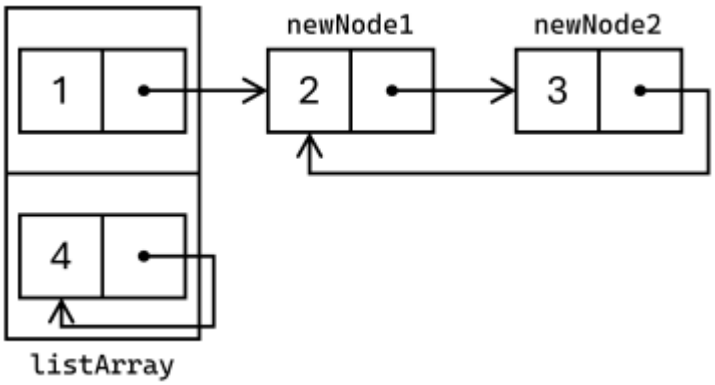
- Your function must not attempt to free any memory allocated for the input array and pointers. Doing so may cause segmentation fault when your function is being tested. We may not be able to grade your program if your program produce segmentation fault while grading.

Testing and compilation:

- Your program will be tested with tester programs. These tester programs will be compiled together with your `3.c` for testing. For example the tester `main3_1.c` has the following `main()` function (slightly modified for formatting). **No mark would be given if your program doesn't work with the sample tester programs.**

```
int main() {
  int n = 2;
  ListNode * listArray = (ListNode *)malloc(n * sizeof(ListNode));

  ListNode * listPtr = &listArray[0];
  ListNode * newNode1 = (ListNode *)malloc(sizeof(ListNode));
  ListNode * newNode2 = (ListNode *)malloc(sizeof(ListNode));
  listPtr->data = 1;   listPtr->next = newNode1;
  newNode1->data = 2;  newNode1->next = newNode2;
  newNode2->data = 3;  newNode2->next = newNode1;

  listPtr = &listArray[1];
  listPtr->data = 4;   listPtr->next = listPtr;

  double average = findAverageCycleLength(listArray, n);

  printf("%f\n", average);

  free(newNode1);
  free(newNode2);
  free(listArray);
}
```

The figure below shows the array and linked lists created in the main function above.



The tester program will be compiled with the following command:

```
gcc -pedantic-errors -std=c11 -o main3_1 main3_1.c 3.c
```

The corresponding sample output when running `./main3_1` could be found in `output3_1.txt`. You are expected to use the same `diff` test as explained in the section "Sample test cases" above to check the output.

Do not include any `main()` function in `3.c`. We may not be able to grade your work if you do so. Instead, please implement your own tester programs for testing.

Sample Test Cases

3_1

Tester program: `main3_1.c` (See above)

Output:

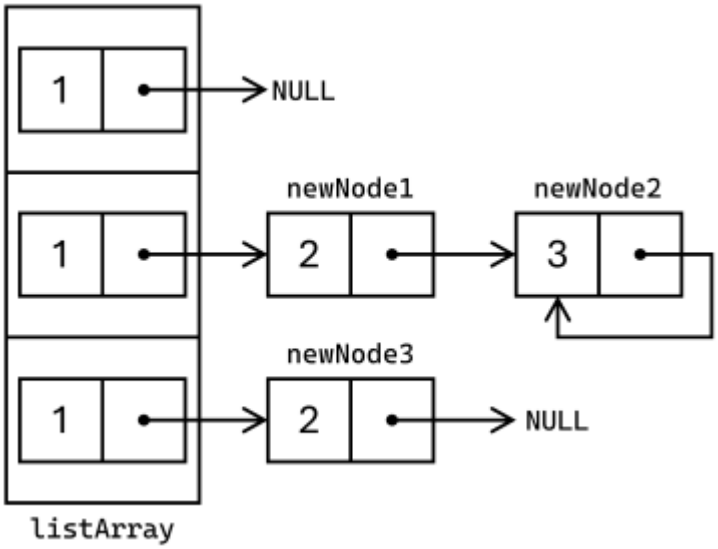
```
1.500000
```

3_2

Tester program: `main3_2.c`

```
int main() {
  int n = 3;
  ListNode * listArray = (ListNode *)malloc(n * sizeof(ListNode));

  ListNode * listPtr = &listArray[0];
  listPtr->data = 1;
  listPtr->next = NULL;

  listPtr = &listArray[1];
  ListNode * newNode1 = (ListNode *)malloc(sizeof(ListNode));
  ListNode * newNode2 = (ListNode *)malloc(sizeof(ListNode));
  listPtr->data = 1;   listPtr->next = newNode1;
  newNode1->data = 2;  newNode1->next = newNode2;
  newNode2->data = 3;  newNode2->next = newNode2;

  listPtr = &listArray[2];
  ListNode * newNode3 = (ListNode *)malloc(sizeof(ListNode));
  listPtr->data = 1;   listPtr->next = newNode3;
  newNode3->data = 2;  newNode3->next = NULL;

  double average = findAverageCycleLength(listArray, n);

  printf("%f\n", average);

  free(newNode1);
  free(newNode2);
  free(newNode3);
  free(listArray);
}
```

Visualized:



Output:

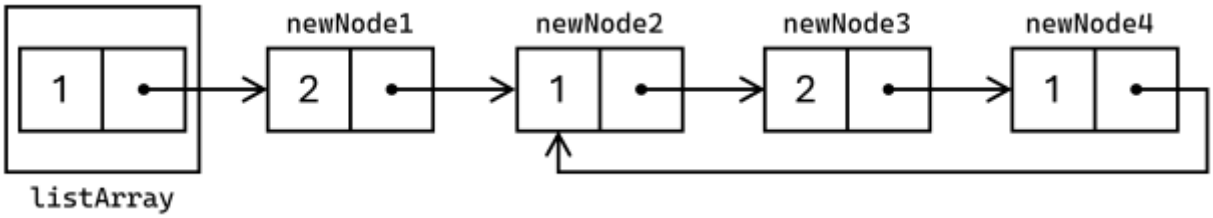
```
0.333333
```

3_3

Tester program: `main3_3.c`

```
int main() {
  int n = 1;
  ListNode * listArray = (ListNode *)malloc(n * sizeof(ListNode));

  ListNode * listPtr = &listArray[0];
  ListNode * newNode1 = (ListNode *)malloc(sizeof(ListNode));
  ListNode * newNode2 = (ListNode *)malloc(sizeof(ListNode));
  ListNode * newNode3 = (ListNode *)malloc(sizeof(ListNode));
  ListNode * newNode4 = (ListNode *)malloc(sizeof(ListNode));
  listPtr->data = 1;   listPtr->next = newNode1;
  newNode1->data = 2;  newNode1->next = newNode2;
  newNode2->data = 1;  newNode2->next = newNode3;
  newNode3->data = 2;  newNode3->next = newNode4;
  newNode4->data = 1;  newNode4->next = newNode2;

  double average = findAverageCycleLength(listArray, n);

  printf("%f\n", average);

  free(newNode1);
  free(newNode2);
  free(newNode3);
  free(newNode4);
  free(listArray);
}
```

Visualized:



Output:

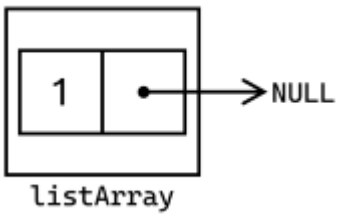
```
3.000000
```

3_4

Tester program: `main3_4.c`

```
int main() {
  int n = 1;
  ListNode * listArray = (ListNode *)malloc(n * sizeof(ListNode));

  ListNode * listPtr = &listArray[0];
  listPtr->data = 1;   listPtr->next = NULL;

  double average = findAverageCycleLength(listArray, n);

  printf("%f\n", average);

  free(listArray);
}
```

Visualized:



Output:

```
0.000000
```