ENGG1340 / COMP2113, Assignment 3

Due Date: Apr 29, 2022 23:59

If you have any questions, please post to the Moodle discussion forum on Assignment 1.

- General Instructions
- Problem 1: Large Numbers Addition (50 marks)
- Problem 2: STL - Log Analyzer (45 marks)

Total marks: 100 marks

- 5 marks for proper code comments, indentation and use of functions
- 95 marks for program correctness

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 TWO problems and a tester would automatically test your submitted program. So if your submitted files and program outputs do not conform to our instructions given here, your programs cannot be evaluated and you will risk losing marks totally.

Sample test cases are provided with each problem in this document. 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 are able to design proper test cases to verify the correctness of your program. We will also use additional test cases when marking your submission.

Input and output format

Note carefully whether your C++ programs should read from the **standard input**, **command line input** or **file input**. Also, unless specified otherwise, your answer should be printed through the **standard output**. If you failed to follow the instructions, the tester may not be able to give a score for your program. Additionally, you should strictly follow the sample output format (including space, line breaker, etc.), otherwise, your answer might be considered as wrong.

How to use the sample test cases

Sample test cases in text file formats are made available for you to check against your work to avoid formatting errors which might fail the tester. Here's how you may use the sample test cases. Take Problem 2 test case 3 as an example. The sample input and the expected output are given in the files `input2_3.txt` and `output2_3.txt`, respectively. Suppose that your program is named "2", do the followings at the command prompt of the terminal to check if there is any difference between your output and the expected output.

```
./2 < input2_3.txt > myoutput.txt
diff myoutput.txt output2_3.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.

Coding environment

You must make sure that your program can compile, execute and generate the required outputs on our standard environment, namely, the gcc C++11 environment we have on the CS Linux servers (academy*).

Make sure the following compilation command is used to compile your programs:

```
g++ -pedantic-errors -std=c++11 [yourprogram].cpp
```

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.

Submission

Name your C++ programs as in the following table and put them together into one directory. Make sure that the folder contains only these source files (`*.cpp`) and no other files. **Compress this directory as a `[uid].zip` file where [uid] is your university number** and check carefully that the correct zip file have been submitted. We suggest you to download your submitted file from Moodle, extract them, and check for correctness. **You will risk receiving 0 marks for this assignment if you submit incorrect files.** Resubmission after the deadline is not allowed.

Problem	Code templates provided	Files to Submit
1	Create your own `1.cpp`	`1.cpp`
2	`2.cpp`	`2.cpp`

Late submission

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

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.

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.

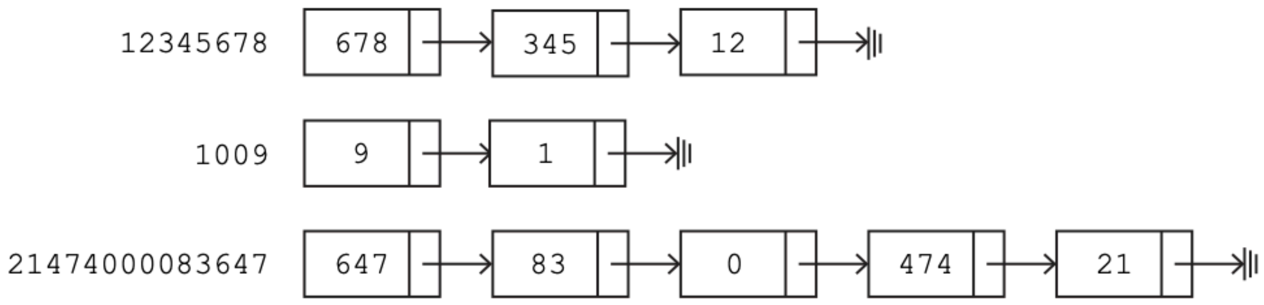
Please be careful not to post spoilers. Please don't post any code that is directly related to the assignments to the discussion forums or share your work to any public domain. However you are welcome and encouraged to discuss general ideas on the discussion forums.

Problem 1: Large Numbers Addition

This problem is an extension of the tutorial problem in Module 8.3 on large number comparisons.

The largest integer that can be stored using a 32-bit int data type is 2,147,483,647. In this question, you are going to implement addition of two arbitrarily large numbers using linked lists.

A large integer is to be stored in your program as follows. Starting from the least significant digit, an integer n is segmented into **chunks of 3 digits**. The **value** of the chunk with the least significant digits is stored in the first node and that of the most significant digits is stored in the last node, so that **the linked list in reverse** will give the original integer n. Here are some examples:



Input:

- Your program should accept an expression `a + b` from the user, which is an addition of two large numbers `a` and `b`.

Output:

- First two lines of output print the linked list storing the two input large numbers to be added.
- The third line of output prints the linked linked list storing the sum of the two numbers.
- The fourth line of output prints the sum in its conventional form.

Requirements:

- You should assume that the input integers are of arbitrarily number of digits. In other words, you cannot simply store the input numbers in any integer or string types.
- You should start with the solution program `largenum.cpp` of the tutorial problem and modify the code from there. Then the use of dynamic array to handle the arbitrary long input would have been done for you.
- Remember to name your submission as `1.cpp`.
- Modify the code so it first creates two linked lists for the input numbers, then performs the addition which creates a third linked list storing the sum.
- Addition is done by adding the values stored in the nodes of the two linked lists that correspond to the same decimal places, and propagating the carry digit (if there is any) to the addition of the next nodes storing the value of the more significant digits.
- You are not allowed to use STL containers (e.g., vectors) or other external libraries.

- You may add your own functions wherever appropriate for better program modularity.
- You are allowed to reuse/modify functions in the sample programs `build_list_forward.cpp`, `build_list_backward.cpp`, `build_list_sorted.cpp`, `build_list_reverse.cpp` and `largenum.cpp` of Module 8.3 whenever appropriate.

Sample Test Cases

User inputs are shown in blue.

1_1

```
12345678 + 1009
678 -> 345 -> 12 -> NULL
9 -> 1 -> NULL
687 -> 346 -> 12 -> NULL
12346687
```

1_2

```
999999999 + 99
999 -> 999 -> 999 -> NULL
99 -> NULL
98 -> 0 -> 0 -> 1 -> NULL
1000000098
```

1_3

```
12345678 + 21474000083647
678 -> 345 -> 12 -> NULL
647 -> 83 -> 0 -> 474 -> 21 -> NULL
325 -> 429 -> 12 -> 474 -> 21 -> NULL
21474012429325
```

Problem 2: STL - Log Analyzer

You are provided with a template program `2.cpp`. Complete the program and the program will read and process a web log data file from user input (`cin`). The program will then print out the 5 most popular pages, and the 5 most active users and the corresponding pages they have visited.

The log file consists of records of three types, each record occupies exactly one line. Here is the format of these three types of record:

Page record: `PAGE <page id> <page url>`
User record: `USER <user id>`
Visiting record: `VISIT <page id>`

A **page** record represents a page on the web server. A **user** record represents a user that accesses the system. A **visiting** record represents a visit by the user indicated by the most recent user record. Here is a sample data file:

```
PAGE 1288 /library
PAGE 1282 /home
USER 20686
VISIT 1288
VISIT 1282
USER 20687
VISIT 1288
```

In this case, we have 2 pages (`/library` and `/home`) on the server. Two users have accessed the server, one (#20686) visiting 2 pages (`/library` and `/home`) and the other (#20687) visiting 1 page (`/library`).

You can assume the followings regarding the data file:

- The data file always consists of the three types of records only
- The page ids are unique across all PAGE records
- The user ids are unique across all USER records
- The page ids are unique across all VISIT records of a user
- VISIT records will only appear after the first USER record
- The page id in a VISIT record appears only after its corresponding PAGE record

- There will be at least 5 users and 5 pages

The followings have been implemented for you:

- A `Page` structure, each `Page` object consists of the id, path and a counter to count the number times it is being visited

```
struct Page {
    int id;
    string path;
    int counter;
    Page(int id, string path) {
        this->id = id;
        this->path = path;
        counter = 0;
    };
};
```

- A STL vector for organizing the `Page` objects

```
vector<Page> pages;
```

- A `User` structure, each `User` object consists of the id and the pages the user visited

```
struct User {
    int id;
    vector<string> visits;
    User(int id) {
        this->id = id;
    };
    void add_visit(int page_id) {
        ...
    };
    int size() const {
        return visits.size();
    };
    void print_visits() {
        ...
    }
};
```

- A STL vector for organizing the `User` objects

```
vector<User> users;
```

- A function to overload `<` operator for the comparison of 2 pages based on their ids

```
bool operator<(const Page & a, const Page & b) {
    return (a.id < b.id);
};
```

You are required to complete the missing functions in the template program to make it work. *Note*: The functions `Page()` and `User()` in the structs `Page` and `User` above are called **struct constructors**. Refer to Module 10 "C Programming (Part 3)" for the discussion on the struct constructor functions. It works the same in C++.

Sample Test Cases

You are provided with 4 sample test cases. We will use the first test case and detail the test run below.

Assume we have `input2_1.txt` in the current directory. If we run the program like this (assuming that we are working on Linux):

```
$ g++ -pedantic-errors -std=c++11 2.cpp -o 2
$ ./2 < input2_1.txt
```

The program will print:

```
*** 5 most popular pages ***
36:/msdownload
32:/ie
17:/products
17:/search
13:/sitebuilder
*** 5 most active users ***
23:10068
- /activeplatform
- /activex
- /automap
- /frontpage
- /gallery
- /ie
- /intdev
- /isapi
- /java
- /msdownload
- /musicproducer
- /office
- /promo
- /sbnmember
- /search
- /sitebuilder
- /spain
- /vbasic
...
[snipped]
...
11:10019
- /athome
- /clipgallerylive
- /games
- /isapi
- /kb
- /logostore
- /msdownload
- /msoffice
- /ntserver
- /products
- /search
```

The output format is explained as the below:

- Each line of the "5 most popular pages" is in the format of "visit_counter:page_path". E.g., `36:/msdownload`.
- A line of an active user is in the format of "number_of_pages_visisted:user_id". E.g., `23:10068`.
- A line of an active user's visit is in the format of "- page_path". E.g., `- /activeplatform`.

Note:

- If two pages are equally popular, the pages are ordered lexicographically by the path.
- If two users visit the same number of pages, the users are ordered by their id ascendingly.
- The list of pages a user visit must be printed in lexicographical ascending order.
- You can choose not to use the provided template program and implement in your own way. However, your program must contain the provided Page and User structures and use STL vectors for storage.
- Once again, you MUST use STL vectors in your program. Your program will be checked and your score will be 0 if we find that you use traditional arrays to store pages and users in your implementation.