2018-2 Computer Programming Final Exam (Programming Part)

You have access limited to (a) lecture notes, either at eTL or in printed form and (b) uploaded codes in eTL. You may not refer to any personal codes saved in your account and/or machine, but you can use your submitted lab and/or assignment and/or midterm codes. Otherwise, accessing the Internet is strictly prohibited.

Occasionally save your work. Proctors are not responsible for any damages that may incur to your written codes.

Write everything, including comments, in English. Using any other language than English might involve compilation errors. Note that you cannot get any points if your code does not compile.

Do not change the format of input and output. You will get severe penalties if you do not follow the input and/or output specification.

If a specific problem/task includes constraints on `import`s or `#include`s, you need to follow those.
For Java codes, remove any statements in your code that include `package`. In most cases, they will trigger compile errors.

Upload your work in eTL. For codes, you should submit only a single TAR or ZIP file containing these files:
[1] Problem 1: EncodeTree.java or EncodeTree.cpp . (You are able to submit your code in only one language: if you submit both, we will consider the one with lower grade.)
[2] Problem 2: DCPair.cpp .
[3] Problem 3: WordCount.java .
Do not include any subdirectories or any other files inside your archive, so that we can see your source codes right after untar/unzipping it. If you compress your files in MacOS Environment, please attach “mac-” into your zip file name. e.g.) “mac-[your zip name].zip”

Proctor will open a Q&A session about 10 minutes after the exam begins. Try to read and understand the questions before that time.
You need to finish your work no later than 3:45PM. Submission due is 3:50PM. No late submission is allowed.

Grading will be done in Linux environment using Java (OpenJDK) 11 and C++ (g++) 8.2.1, identical to that inside the lab machines. Keep that in mind when writing code in other environments.

You may not proceed before the proctor allows you to do so.

Problem 1 (C++ or Java)

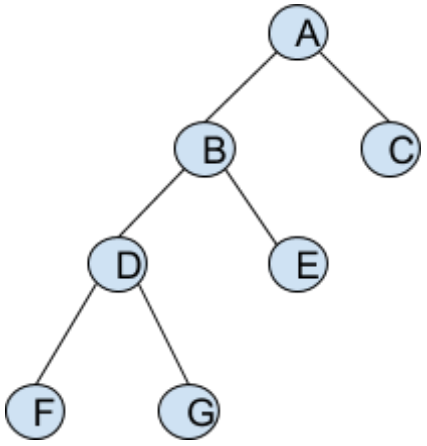
Complete a program EncodeTree.java or EncodeTree.cpp that encodes a binary tree.

- In this program, you need to implement a method `encode()` that encodes a binary tree in `std::string`/`String`.

Let's first assume that a node in a binary tree has either two children or no children (i.e., there are no nodes with only one child). Each node in the binary tree has a data field whose type is `char` (e.g., A, B, c, 5, …). The encoding scheme (used in the constructor) can be described using the following procedure:

[1] Begin from the root node.
[2] Output the label of the root node.
[3] Encode its left subtree.
[4] Encode its right subtree.
[5] Output label of the root node again.

When a tree is given as follows, the result of encoding is ABDFFGGDEEBCCA.



- Code for a single node is given as Node.java or Node.h, which are uploaded in eTL. You may not modify those files.
- You do not need to write your own `main()` method. Graders will write their own `main()` method, such as:

cat ./Test.java

```
public class Test
{
    public static void
main(String[] ar)
    {
        Node root = new
Node('A');
        root.setLeft(new
Node('B'));
        root.setRight(new
Node('C'));
        root.left.setLeft(new
Node('D'));
```

cat ./test.cpp

```
#include <iostream>
#include "Node.h"
#include "EncodeTree.cpp"
int main(int argc, char
*argv[])
{
    Node *root = new Node('A');
    root->setLeft(new
Node('B'));
    root->setRight(new
Node('C'));
    root->left->setLeft(new
Node('D'));
```

```
        root.left.setRight(new
Node('E'));

root.left.left.setLeft(new
Node('F'));

root.left.left.setRight(new
Node('G'));
        EncodeTree ent = new
EncodeTree(root);

System.out.println(ent.encode()
);
    }
}
```

javac Test.java
java Test
ABDFFGGDEEBCCA

```
    root->left->setRight(new
Node('E'));

root->left->left->setLeft(new
Node('F'));

root->left->left->setRight(new
Node('G'));
    std::cout << encode(root)
<< std::endl;
}
```

g++ ./test.cpp -o ./test
./test
ABDFFGGDEEBCCA

- You may assume that the labels of the nodes are unique.
- You may not import any external Java or C++ libraries.
- You may not add extra classes and/or methods other than provided. You may modify return type and parameters of the given method or just add a method with the same method name and different parameters or type.

Failing to following above will result in zero score.

Problem 2 (C++)

Write a C++ program DCPair.cpp that defines a pair of dollar and cent and implements four binary operators (+, −, < and the auxiliary operator <<).

- Your program should be implemented using operator overloading. Your program should be compatible with the given header file DCPair.h.

- Define a class `DCPair` representing a pair (`dollar`, `cent`) (`dollar`: total dollar in `int`, `cent`: total cent in `int`). A dollar is worth 100 cents, so range of `cent` is -99~99. In this problem, **sign** of `dollar` and `cent` **should be identical**.

- The '+' and '-' of two `DCPair`s are defined as follows. Given `DCPair`s `c1 = (a, b)` and `c2 = (x, y)`,

[1] `c1 + c2 = (a + x + (b + y) div 100, (b + y) mod 100)`

[2] `c1 - c2 = (a - x + (b - y) div 100, (b - y) mod 100)`

[3] `c1 < c2` if (a) `a < x` or (b) `a == x` and `b < y`.

A `DCPair (a, b)` has negative value when (a) `a < 0` or (b) `a == 0` and `b < 0`.

- Outputs should be done as `(-)dollar.cent` (e.g., `-1.50, -0.75`).

- The number of digits after "." must be two.

- Complete overloaded operator **<<** to generate an output to the console. You may add `print()` to support operator <<.

[4] `std::cout << c1` should print the value of `c1` to follow output specified above.

- You may add more variables and functions inside if you want.

- You do not need to write your own `main()` method. Graders will write their own `main()` method, such as:

cat ./example.cpp

```
#include <iostream>
#include "DCPair.h"
int main(int argc, char *argv[])
{
     int a = 1, b = 2, x = 50, y = 1;

     DCPair dca(a, x); //(1, 50).
     DCPair dcb(b, y); //(2, 1).

     DCPair addresult = dca + dcb; //(3, 51).
     DCPair subresult = dca - dcb; //(0, -51).

     if(dca < dcb) {
         std::cout << dca << " is smaller than " << dcb <<
std::endl;
     } else {
         std::cout << dca << " is greater than or equal to " <<
dcb << std::endl;
     }
```

```
      return 0;
}
```
g++ ./DCPair.cpp ./example.cpp -o example
./example
```
3.51
-0.51 /* -1.49 is a wrong output. */
1.50 is smaller than 2.01 /* 1.5 and 2.1 are wrong outputs. */
```

Problem 3 (Java)

Write a Java program WordCount.java that analyzes a sequence of words.

- Parameters will be given as command line inputs. First parameter is an input file, and the second parameter is an output file.
- An input file contains a single line of characters, with range of 'a'-'z' and 'A'-'Z', space, comma, and period. Any sequence of characters (without space, comma or period) is considered as a word. For instance, when the input line is:

```
I like banana, but apple   is good too. I hate fish.
```

valid words are:

I, like, banana, but, apple, is, good, too, I, hate, fish

. Note that there could be more than one occurrence of a single word.
- In `main(String[] ar)` method, you need to include:
[1] reading an input file.
[2] outputting the number of words, average length of a word, and how many words are above the average length, into an output file.
- When calculating average length of a word, round it to two decimal places.
- You need to follow exactly the same format as in the examples below. Do not include any extra characters on output.

cat ./input1.txt

```
I like banana, but apple is good too. I hate fish.
```

java WordCount ./input1.txt ./output1.txt

(Nothing is printed to the console.)

cat ./output1.txt

```
Number of words = 11
Average length of a word = 3.36
Number of words above the average length = 6
```

cat ./input2.txt

```
.Abra cada.bra alAk, . azam
```

java WordCount ./input2.txt ./output2.txt

(Nothing is printed to the console.)

cat ./output2.txt

```
Number of words = 5 /*Abra, cada, bra, alAk, azam*/
Average length of a word = 3.8
Number of words above the average length = 4
```

- You do not need to consider erroneous parameters and/or inputs.