Week 12

Remind

- 抄襲一律 0 分 （包含被抄襲者）
- 繳交期限: **12/6(Sun.) 11:59 p.m.**
- 繳交的檔案格式、名稱請符合以下規定
    - 請繳交 zip檔至 Ceiba作業區，名稱為 <student_id>.zip
    - 解壓縮後須符合格式、名稱
    - e.g. b12345678.zip
- 必須完成 Demo 才可以提早離開
- 若沒有完成 Demo 就中途早退，視同缺席

Problem 1 - Linked list (7%)

Introduction

Imagine you are a software engineer for a bank. You are asked to design a simple account tracking system. For simplicity, you decide to implement the system using the linked list data structure. Feel free to seek reference from the list.cpp file at week 8 from Ceiba. You will be implementing two classes: the `Node` class and the `Account` class. The purpose of this homework is to get you familiar with the class syntax.

Step 1.

Write a `class Node` that contains four fields:

1. Item (what you buy)
2. Value (how much is it)
3. Time (when you buy)
4. Pointer `next` pointing to the next node

First of all, you need to track down what you buy, how much you spend, and when you buy it. A node seems like an appropriate date structure to contain all these information. Each node is a record of a transaction, and by linking all of them we have derive a linked list for all historic transaction records. In short, the node list contains all information of thing you buy.

```
class Node
{
public:
  // your code here
};
```

Step 2.

Write the decorator for the `class Node`.

To write a class object, we have to perform initialization when instantiation. Here you will write two decorators using the function overload:

`Node(void)`

No argument, simply set `value` to `0` and `next` to `NULL`.

`Node(string _item, int _value, string _time)`

Initialize each field according to the three given arguments. Also, set `next` to `NULL`.

```
class Node
{
public:
  // your code here

  Node(void)
  {
    // your code here
  }

  Node(string _item, int _value, string _time)
  {
    // your code here
  }
};
```

Step 3.

Write an `ENUM Mode` where

- DEFAUIT → 0
- ITEM → 1
- VALUE → 2
- TIME → 3

Step 4.

Write a `class Account` that contains two fields:

1. Node* first (point to the head of the node list)
2. Mode mode (determine the insertion order)

```
class Account
{
public:
  Node* first;
  Mode mode;
}
```

The `class Account` will maintain the linked list, with the pointer `first` pointing to the first node of the list. The `Mode mode` specifies the insert mode and the sort mode. There are four modes in total, namely, DEFAULT, ITEM, VALUE, and TIME. For example, if you are in the ITEM mode, then you should insert a new transaction record based on the ASCII order of the item field. More information are explained in step 6.

Step 5.

Write the decorator for the `class Account`. Like step 2, initialize `first` to NULL and `mode` to DEFAULT.

```
Account()
{
  // your code here
}
```

Step 6.

`void insert(string _item, int _value, string _time)`

This is a member function of `class Account`.

- `_item`: what you buy
- `_value`: how much is it
- `_time`: when you buy

```
void insert(string _item, int _value, string _time)
{
  if (first == NULL)
  {
    first = new Node(_item, _value, _time);
  }
  else
  {
    // your code here
  }
}
```

You should insert data according to what mode you are in right now.

- If the current mode is DEFAULT, then insert at the front of the list.
- If the current mode is ITEM, then insert based on the ASCII order of the item.
- If the current mode is VALUE, then insert based on the price of the item.
- If the current mode is TIME, then insert based on the purchasing time. The format of time is `dd/mm/yy`, which is a string.

Note

- All the orders are from low to high.
- If two values are the same (e.g. `a == b`), view as `a > b`.

Step 7.

`void display()`

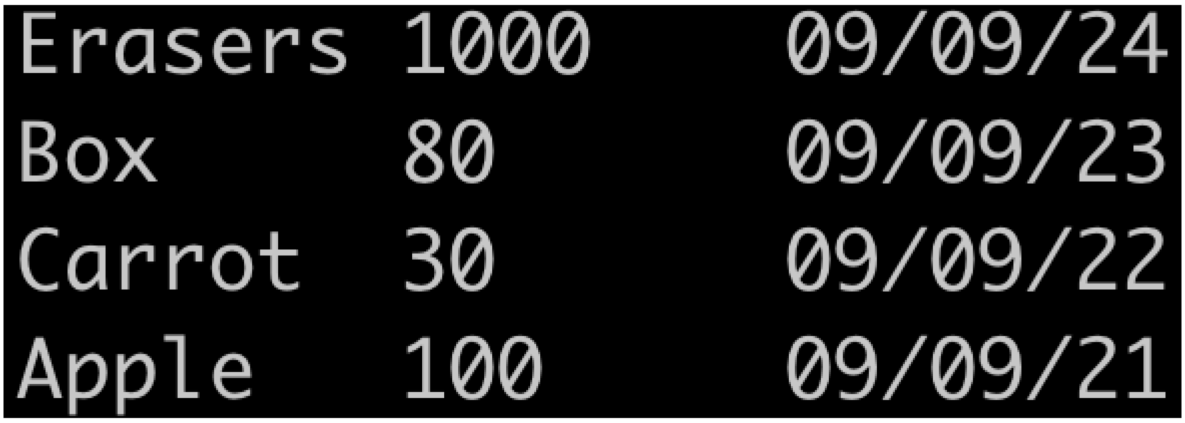
This is a member function of `class Account`.

Print out the list sequentially.

For example, if you call

```
int main()
{
  Account account;
  account.insert("Apple", 100, "09/09/21");
  account.insert("Carrot", 30, "09/09/22");
  account.insert("Box", 80, "09/09/23");
  account.insert("Erasers", 1000, "09/09/24");
  account.display();
  return 0;
}
```

The result will be



```
Erasers 1000        09/09/24
Box     80          09/09/23
Carrot  30          09/09/22
Apple   100         09/09/21
```

Step 8.

`void erase()`

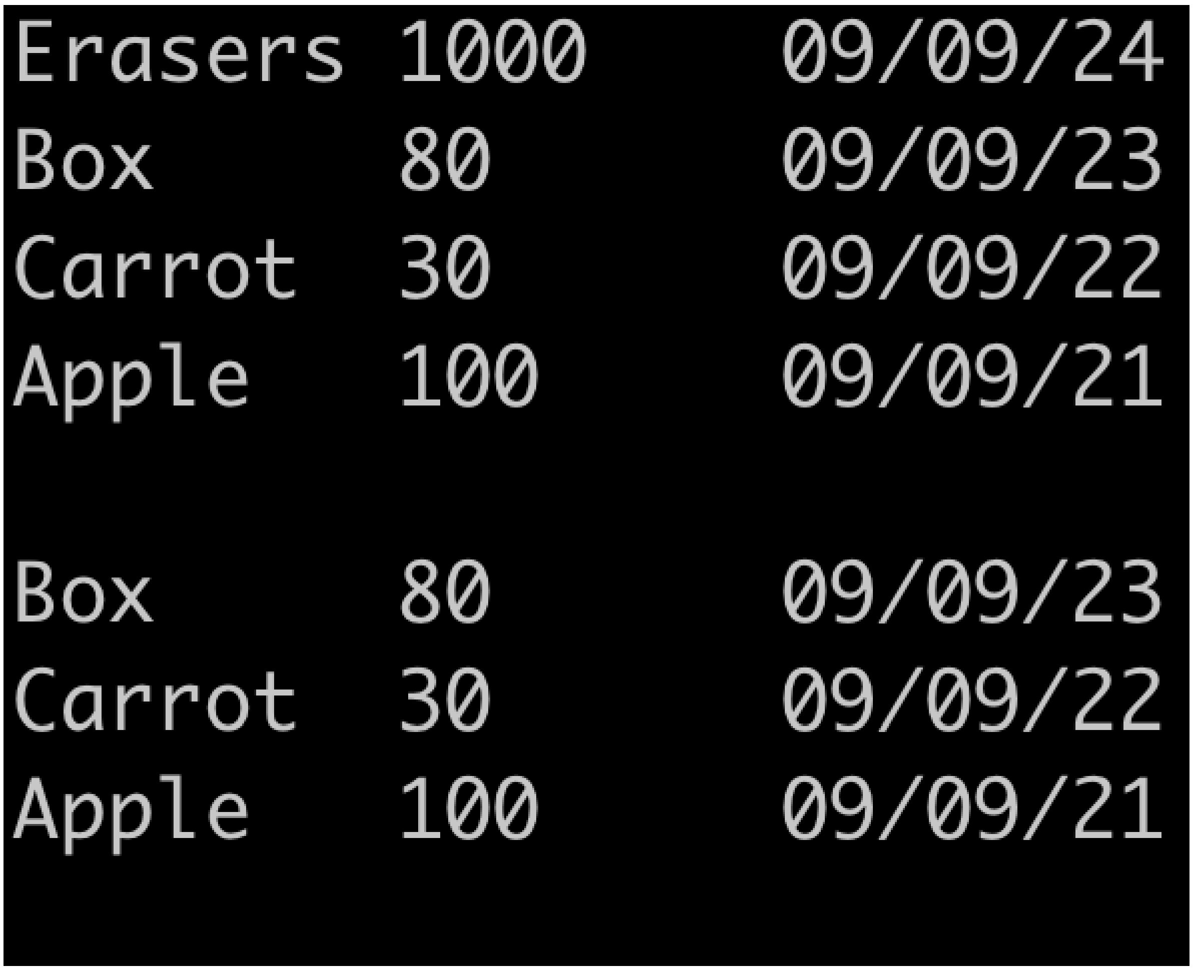
This is a member function of `class Account`.

Delete the last item of the list.

For example,

```
int main(int argc, char const *argv[])
{
  Account account;
  account.insert("Apple", 100, "09/09/21");
  account.insert("Carrot", 30, "09/09/22");
  account.insert("Box", 80, "09/09/23");
  account.insert("Erasers", 1000, "09/09/24");
  account.display(); cout << endl;
  account.erase();
  account.display(); cout << endl;
  return 0;
}
```

Then the result is



```
Erasers 1000     09/09/24
Box     80       09/09/23
Carrot  30       09/09/22
Apple   100      09/09/21

Box     80       09/09/23
Carrot  30       09/09/22
Apple   100      09/09/21
```

Step 9.

`void clearAll()`

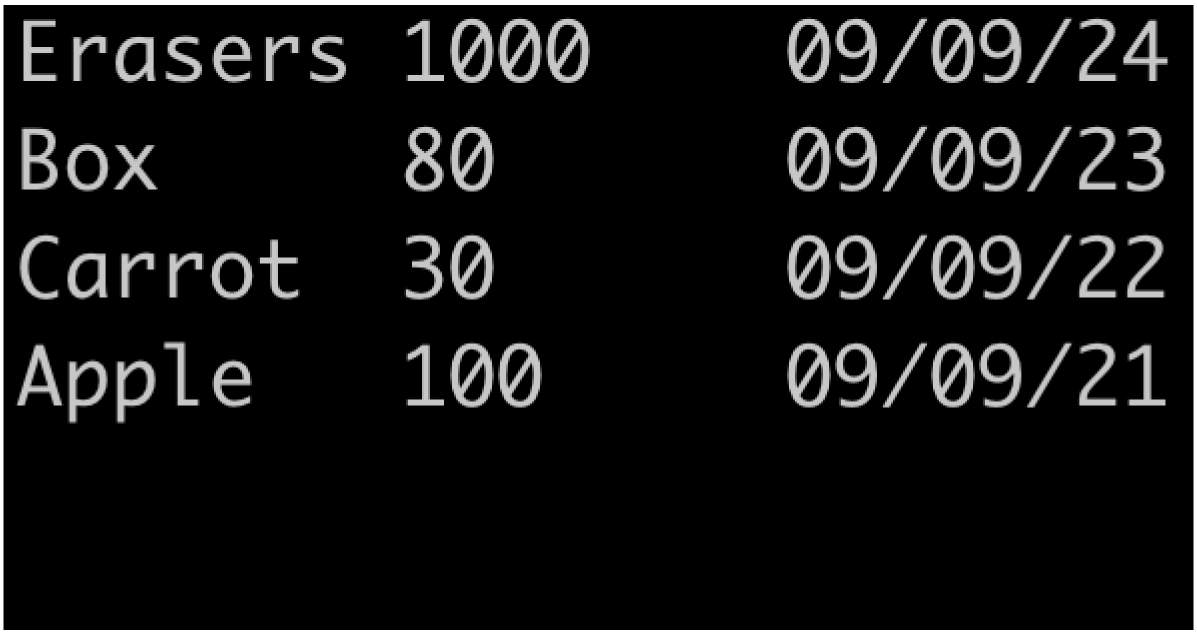
This is a member function of `class Account`.

Deallocate all the list.

For example,

```
int main(int argc, char const *argv[])
{
  Account account;
  account.insert("Apple", 100, "09/09/21");
  account.insert("Carrot", 30, "09/09/22");
  account.insert("Box", 80, "09/09/23");
  account.insert("Erasers", 1000, "09/09/24");
  account.display(); cout << endl;
  account.clearAll();
  account.display(); cout << endl;
  return 0;
}
```

Then



```
Erasers 1000     09/09/24
Box     80       09/09/23
Carrot  30       09/09/22
Apple   100      09/09/21
```

Step 10.

`void sort(Mode _mode = DEFAUlT)`

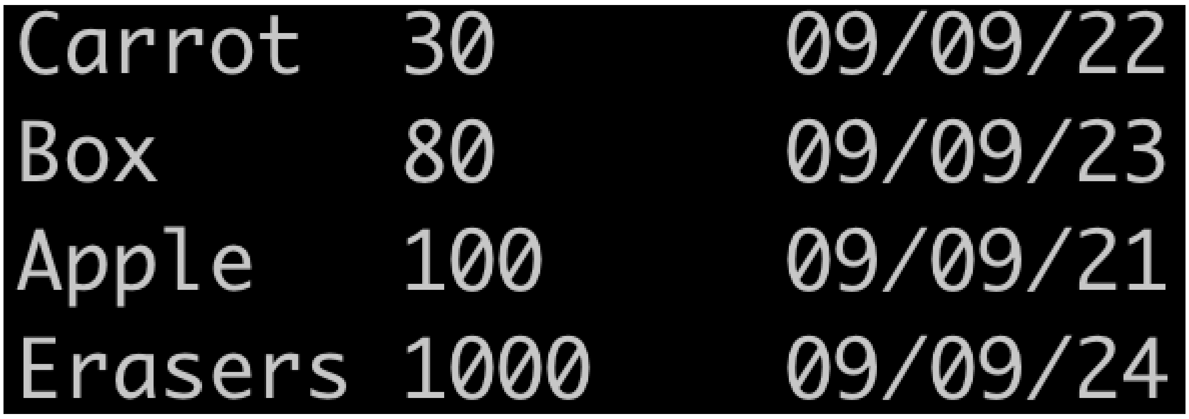
This is a member function of `class Account`.

Sort the list according to the mode.

For example, if you sort the list by VALUE,

```
int main(int argc, char const *argv[])
{
  Account account;
  account.insert("Apple", 100, "09/09/21");
  account.insert("Carrot", 30, "09/09/22");
  account.insert("Box", 80, "09/09/23");
  account.insert("Erasers", 1000, "09/09/24");
  account.sort(VALUE);
  account.display();
  return 0;
}
```

Then


```
Carrot  30       09/09/22
Box     80       09/09/23
Apple   100      09/09/21
Erasers 1000     09/09/24
```

Step 11.

`void summary()`

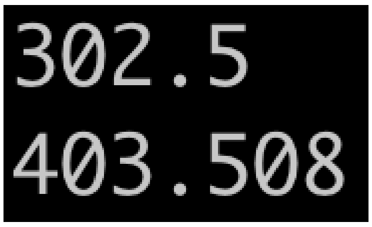
This is a member function of `class Account`.

- Print the **mean** of all values in the first line
- Print the **standard deviation** of all values in the second line

For example,

```
int main(int argc, char const *argv[])
{
  Account account;
  account.insert("Apple", 100, "09/09/21");
  account.insert("Carrot", 30, "09/09/22");
  account.insert("Box", 80, "09/09/23");
  account.insert("Erasers", 1000, "09/09/24");
  account.summary();
  return 0;
}
```

Then


```
302.5
403.508
```

Step 12.

Write the destructor for `class Account`. We'll do this part for you.

```
~Account()
{
    clearAll();
}
```

How your work should look like

```
#include <iostream>
#include <cmath>
#include "account.h"
using namespace std;

class Node
{
public:
  Node(void){}
  Node(string _item, int _value, string _time){}
};

class Account
{

public:
  Account(){...}
  ~Account(){...}

  void insert(string _item, int _value, string _time){...}
  void display(){...}
  void erase(){...}
  void clearAll(){...}
  void sort(Mode _mode = DEFAUlT){...}
  void summary(){...}

// you can write your public funtions if needed

private:
// you can write your private funtions if needed

}
```

How to submit your work

Just submit one .cpp file named `account.cpp`. No main function is needed. We will use your class for testing!