

Lab 08 Classes

We have 3 guided exercises in this lab: Q1 to Q3 ↗

(Q3 is required for [Lecture09](#) and [Lab09](#))

Reminder: Do Assignment 1

Q1. The Date Class (Version 3) -- Complete your answers in [Page 2](#).

(i) Please add 2 constructors:

- The first constructor is the *Default constructor* (set date to Jan-1, 2018)
- The second constructor is a *constructor with 3 parameters for year, month, and day*

(ii) `main` is rewritten and run as follows:

Enter year, month and day as numbers: 2018 10 21
 You typed: year = 2018, month = 10, day = 21
 It is not the end of a month.
 The next day is: year = 2018, month = 10, day = 22
 Press any key to continue . . .

Enter year, month and day as numbers: 2018 12 31
 You typed: year = 2018, month = 12, day = 31
 It is the end of a month.
 The next day is: year = 2019, month = 1, day = 1
 Press any key to continue . . .

```
int main()
{
    Date d, dNext;
    d.input();
    cout << "You typed: ";
    d.output();

    if (d.isEndOfAMonth())
        cout << "It is the end of a month." << endl;
    else
        cout << "It is not the end of a month." << endl;

    dNext=d.next();
    cout << "The next day is: ";
    dNext.output();

    return 0;
}
```

You can see how the `main` function uses two member functions: `.isEndOfAMonth` and `.next`

Please finish these two member functions. Submit the completed program files to PASS.

Checkpoints (Make sure you fulfill the items below. Put a ✓ where appropriate. Answer questions marked with *.)

- ☐ I know how to write constructors, I know how they are called and how they run.
- they run when `main` declares the objects `d` and _____*,
 - and when the `next*` / `isEndOfAMonth*` function creates the resultant object !!
[choose one of them ^]
- ☐ I know how to write member functions,
- ☐ I know how they are called, eg. by the `main` function
 - ☐ I know they run.
- ☐ I notice that return type of the `next*` / `isEndOfAMonth*` member function is DATE!!
[choose one of them ^]
- ☐ I notice how the `next*` / `isEndOfAMonth*` / `valid*` member function calls the `next*` / `isEndOfAMonth*` / `valid*` member function, and `next*` / `isEndOfAMonth*` / `valid*` calls the `next*` / `isEndOfAMonth*` / `valid*` member function.
- ☐ I notice how calling objects are processed.
- The **calling object** (`Date d`; in the `main` function) is `*` / `is not` * accessed by `next`
 - The **calling object** (`Date d`; in the `main` function) is `*` / `is not` * accessed by `isEndOfAMonth`
 - The **calling object** (`Date d`; in the `main` function) is `*` / `is not` * accessed by `valid`.

[Lab08-Q1] Given framework for students

```
class Date {  
public:  
    // [TO-DO: Lab08-Q1 (i)] Add 2 constructors.
```

Date.h

Explanation:

The constructors will run when we have "Date d;", "Date d(2009,1,1);" etc.
(Ref: Lec08.pdf Page 10.)

```
_____  
_____  
  
//Checks whether the calling date object is the ending day of any month  
bool isEndOfAMonth();  
  
//Returns the "tomorrow" of the calling date object  
Date next();  
  
//I-O  
void input();  
void output();  
  
//Access functions  
void set(int new_y, int new_m, int new_d);  
int get_year();  
int get_month();  
int get_day();  
  
private:  
  
    // check if y,m,d are valid  
    bool valid(int y, int m, int d);  
  
    //check if y is a leap year  
    bool isLeapYear(int y);  
  
    //Date data  
    int year;  
    int month;  
    int day;  
};
```

```
#include <iostream>  
#include "Date.h"
```

Date.cpp

```
using namespace std;
```

```
// [TO-DO: Lab08-Q1 (i)] Add 2 constructors.
```

Explanation:

The constructors will run when we have "Date d;", "Date d(2009,1,1);" etc.
(Ref: Lec08.pdf Page 10.)

```
// Check whether the calling date object contains a date which is the ending day of any month  
bool Date::isEndOfAMonth()  
{
```

```
    // [TO-DO: Lab08-Q1 (ii)]
```

if year, month and day+1 is a valid date <== use the valid member function!!
then return false
otherwise
return true!!

```
}
```

```
// Return the "tomorrow" of the calling date object  
Date Date::next()  
{
```

```
    // [TO-DO: Lab08-Q1 (ii)]
```

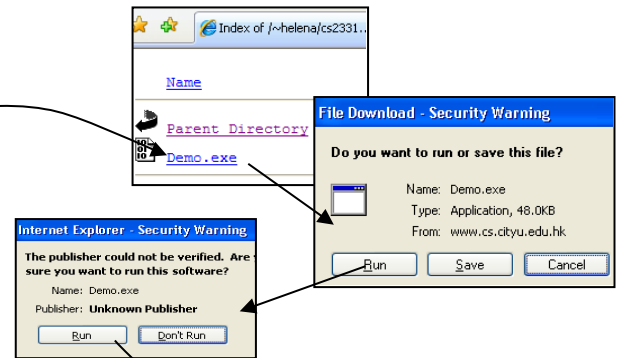
if (isEndOfAMonth()) <== is the end of a month
if it is not December, then
return Date(year,month+1,1); <== Construct and return a date object (1st day of next month)
else
return .. <== 1/1 of next year
else
return .. <== the next day in the same month

```
}
```

```
.. other functions are given already in the file.
```

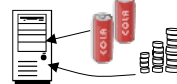
Q2. Vending Machine

Step 1. Locate Demo.exe for this question. **Run it** as shown below.



The opening:

The machine has \$100 and 2 cokes, awaiting...
(\$5 per can; only accepts \$1, \$2, \$5 coins)



Then YOU come  .

You may "press" the buttons: r, i, g, q (see below).

(underlined contents are input by user.)

- You type 'r' // want refund
Machine says: You haven't paid any, so refund \$0
- You type 'i 2' // insert \$2
Machine says: Alright, \$2 so far
- You type 'i 2' // insert \$2
Machine says: Alright, \$4 so far
- You type 'i 2' // insert \$2
Machine says: Alright, \$6 so far
- You type 'g' // want a coke
Machine says: OK, give you , refund \$1, see you next time.
- You type 'g' // want a coke
Machine says: You only paid \$0 so far, **No way!**
- You type 'i 5' // insert \$5
Machine says: Alright, \$5 so far
- You type 'i 3' // insert \$3 (a \$3 coin??)
Machine says: **Sorry! I only want \$1, \$2, \$5!!** Result: still \$5 so far
- You type 'g' // want a coke
Machine says: OK, give you , (no extra for refund), see you next time.
- You type 'i 5' // insert \$5
Machine says: Alright, \$5 so far
- You type 'g' // want a coke
Machine says: Sorry, though you've paid \$5, but I have no more coke.
- You type 'q' // quit: turn off the machine (**forgot to refund!! ☹**)
Machine says: Alright, at closing, I have \$115 and 0 coke.

r
\$0 refunded

i 2
\$2 paid

i 2
\$4 paid

i 2
\$6 paid

g
Here is your Coke, \$1 refunded

g
Not enough money

i 5
\$5 paid

i 3
\$5 paid

g
Here is your Coke

i 5
\$5 paid

g
Out of stock

q
\$115 and 0 coke(s)

Make sure you can run it like the above.
Put a ✓ here when finished ☐.

Step 2. Read the main function:

(Given at the course web.)

Note:

- (1) The **VendMach** class represents a vending machine.
- This line will run the **constructor** with 2 parameters.
- (2) We often use **switch-case** to handle user commands like these.
- (3) The member functions - insertCoin, getCoke, refund, status:
Note how they are called in response to user commands:

```

r: refund
i 1: insert $1 coin
i 2: insert $2 coin
i 5: insert $5 coin
g: get a coke
q: quit
  
```

→ m.status() runs finally at the end of main

```

#include <iostream>
using namespace std;

int main()
{
    VendMach m(100,2); // machine m has $100 and 2 Cokes initially

    char cmd;
    int amt;
    cin >> cmd;
    while (cmd!='q' && cmd!='Q')
    {
        switch (cmd)
        {
            case 'i': cin >> amt; m.insertCoin(amt); break;
            case 'g': m.getCoke(); break;
            case 'r': m.refund(); break;
        }
        cin >> cmd;
    }
    m.status();

    return 0;
}
  
```

Data to handle (3 data members only):

We keep the number of Cokes, the amount of money owned by the machine, and also the amount of money paid by a customer so far.

(For simplicity, we assume there are always enough coins to refund the customer.)

In *VendMach.h*, we will have these data members:

```

int totalMoney;
int totCokes;
int paid;
  
```

Now, please read the run-down in last page again: From the code of **main**, figure out how the 5 functions (constructor, insertCoin, getCoke, refund, status) use and update the 3 data members.

Put a ✓ here when finished ☐.

[CODING]

Step 3. Please fill in the code in Page 5.

Hint: Whenever you get lost, just think: *What should the function do?*

Usually it is simply one of the following:

- Update *totalMoney*? *totCokes*? *paid*?
- And / or show some output to the user?
- And / or pass back something to the caller?

Submit the completed program files to PASS.

Checkpoints (Make sure you fulfill the following items and put a ✓ where appropriate.)

- ☐ I have successfully finished the program in this part (100% correct in PASS).
- ☐ I understand how everything in the program works.

[Lab08-Q2] Given framework for students

```
class VendMach{
public:
    // Constructor and 4 public member functions
    VendMach(int tm, int tc); //Given: Constructor (Inside main, it runs at "VendMach m(100,2);")
    void insertCoin(int x);   //Given: insertCoin (Read how main calls it !!)

    _____ //Please add the getCoke, refund, status member functions (Read how main calls them !!)
    _____
    _____

private:
    _____ // Please add the 3 data members (totalMoney, totCokes, paid) (Read step 2 in the lab sheet.)
    _____
    _____
};
```

VendMach.h

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

//(1) Constructor - Simply set 3 data members (Inside main, it is called at "VendMach m(100,2);"
VendMach::VendMach(int tm, int tc)
{
    paid=0;
    _____ = tm;
    _____ = _____;
}

//(2) status - Show current status of the machine
void VendMach::status()
{
    cout << "$" << totalMoney+paid << " and " << totCokes << " coke(s)\n";
}

//(3) refund - Refund to user
_____refund_____
{
    _____

}

//(4) insertCoin - Accept user's coin x (if valid)
_____insertCoin(int x)
{
    _____

}

//(5) getCoke - Give the user a coke/Refund to user (if valid)
_____getCoke_____
{
    _____

}
```

VendMach.cpp

Refund the money inserted by the user so far (based on value of paid), i.e., output (cout) like "\$1 refunded" and reset the value of paid to zero. (2 lines only!!)

If x is valid (\$1, \$2, \$5), then update the value of paid. No matter x is accepted or not, display (cout) the amount paid so far, like "\$6 paid". (3 lines only!!)

If enough money and still has stock, update the 3 data members and say "Here is your Coke\n" (If excess was paid, refund it: "Here is your Coke, \$? refunded\n". <== see/call the refund function !) Otherwise output "Not enough money\n" or "Out of stock\n" accordingly

Q3. Fraction - Version 1 (Member Functions accessing the calling object plus another one)

(Q3 is required for [Lecture09](#) and [Lab09](#))

Step 1. Understanding Fraction operations

Your task: Read the answer for (a) below, then finish (b) on your own.

(a.) What is the sum of $\frac{5}{6} + \frac{2}{7}$?

Working:

=	$\frac{5 \times 7 + 2 \times 6}{6 \times 7}$	
=	$\frac{47}{42}$	
	(Find the HCF of 47 and 42: obtain 1)	HCF: Highest Common Factor
=	$\frac{47+1}{42+1}$	
=	$\frac{47}{42}$ (The answer)	← No need to turn it to "1 and 5/42"

(b.) What is the sum of $\frac{5}{6} + \frac{2}{3}$?

Working:

=	
=	
	(Find the HCF of and : obtain)
=	
=	(The answer)

Step 2. Design the Fraction class. It keeps 2 data members (numer, denom), like: $\frac{1}{2}$ <== nominator (**numer**)
2 <== denominator (**denom**)

where **numer** and **denom** are relatively prime (i.e., **numer** and **denom** have no common factors).

Your task: Read the code for the Fraction class below. Examples are given on next page.

DON'T HURRY TO START CODING – YOU HAVE TO FOLLOW THE GUIDELINE IN P.8

```
class Fraction {
public:
    Fraction();           // Initialize the fraction with numer=0, denom=1
    Fraction(int n, int d); // Initialize the fraction with numer=n, denom=d
    void display();       // Show (cout) the fraction like 1/2
    void add(Fraction num); // +, -, x, ÷ (See Examples on next page)
    void sub(Fraction num);
    void mul(Fraction num);
    void div(Fraction num);
private:
    void reduce();        // Simplify the fraction. Eg. if it is 27/18 (ie. numer=27 and denom=18),
                          // then change it to 3/2 (ie. numer becomes 3 and denom becomes 2).
    int numer, denom;
};
```

Example 1

```

Fraction num1(1,6), num2(1,2);
num1.display(); // display (cout) 1/6
num2.display(); // display (cout) 1/2
num1.add(num2); // add 1/2 to num1 => num1 becomes 2/3, but num2 doesn't change
num1.display(); // display (cout) 2/3

```

4 is the HCF (Highest
Common Factor)
of 8 and 12

$$\frac{1}{6} + \frac{1}{2} = \frac{1 \times 2 + 1 \times 6}{6 \times 2} = \frac{8}{12} = \frac{8 \div 4}{12 \div 4} = \frac{2}{3}$$

Example 2

```

Fraction num1(5,6), num2(2,3);
num1.add(num2); // add 2/3 to num1 => num1 will be 5/6+2/3, ie. 27/18, then simplified to 3/2; but num2 doesn't change
num1.display(); // display (cout) 3/2 (No need to turn it to "1 and 1/2")

```

Step 3. Get ready the driver program (main)

(A driver program is a simple program for testing your code.)

Sample input and output:

(Underlined contents are input by the user.)

```

Input a fraction (a/b): 1/2
Input another fraction (a/b): 2/3
1/2 + 2/3 = 7/6
1/2 - 2/3 = -1/6
1/2 * 2/3 = 1/3
1/2 / 2/3 = 3/4
Press any key to continue . . .

```

DO NOT
CHANGE
THIS FILE

```

int main()
{
    int numerator, denominator;
    Fraction num1, num2, ans;
    char dummy;

    cout << "Input a fraction (a/b): ";
    cin >> numerator >> dummy >> denominator;
    num1 = Fraction(numerator, denominator);

    cout << "Input another fraction (a/b): ";
    cin >> numerator >> dummy >> denominator;
    num2 = Fraction(numerator, denominator);

    // addition
    ans = num1;
    ans.add(num2);
    num1.display();
    cout << " + ";
    num2.display();
    cout << " = ";
    ans.display();
    cout << endl;

    // .. sub, mul, div ...
    ...

    return 0;
}

```

Step 4. Please fill in the code in Page 8: Fraction.h and Fraction.cpp

Note:

Two special cases should be handled by `.display()`

Case 1: **Denom** is 1, then show **numerator** only.
eg., Suppose `num1` contains 8/1,
Then `num1.display()` shows 8 only.

Case 2: **Numerator** is 0, then show 0 only.
eg., Suppose `num1` contains 0/1,
Then `num1.display()` shows 0 only.

Remember to call `reduce()` at the end of each operation to simplify the fraction.
(Reminder: in the constructor with 2 parameters, also call `reduce()`.)

Assume that no error case will happen (eg. no *division-by-zero*).

Submit the completed `Fraction.h` and `Fraction.cpp` to PASS (DO NOT USE OTHER FILE NAMES).
** NO NEED to upload the driver program (main function) to PASS.

[Lab08-Q3] Given framework for students

Fraction - Version 1 (Member Functions which access the calling object plus another one)

```
//The Fraction Class
class Fraction {
public:
    //Constructors
    Fraction();
    Fraction(int n, int d);

    void display();
    void add(Fraction num);

    _____ //sub, mul, div
    _____
    _____

private:
    void reduce();
    int numer, denom;
};
```

Fraction.h

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

//Default constructor
Fraction::Fraction() {

    set values for numer and denom as 0 and 1

}

//Constructor
Fraction::Fraction(int n, int d)
{

    set values for numer and denom as n and d

    reduce(); // "n/d" may not be in simplest form. So reduce it.

}

//Display a fraction
void Fraction::display() {

    Display fraction, eg. showing "1 / 2"
    Note also two special cases explained
    in Step 4 in the lab sheet.

}

// Find the HCF between 2 positive integers. E.g., say x1=12 and x2=8, their HCF is 4
int findHCF(int x1, int x2) {

    see Lec08_exercise Page 2

}

//Reduce the Fraction
void Fraction::reduce() {
    // call findHCF to get the factor. abs: take absolute values.
    int factor=findHCF(abs(numer),abs(denom));
    numer=numer/factor;
    denom=denom/factor;
}

//Add a fraction
void Fraction::add(Fraction num)
{
    int new_denom, new_numer;
    new_numer=numer*num.denom+num.numer*denom;
    new_denom=denom*num.denom;
    numer=new_numer;
    denom=new_denom;
    reduce();
}

//sub, mul, div
```

Fraction.cpp

IMPORTANT reminder:

For successful learning:
Redo Q1-Q3 (but don't read guideline)