C++ Pool - d14m

The Cast

Francois-Xavier "FX" BOURLET bourle_f@epitech.eu
Alex FOURNIER fourni_b@epitech.eu

Abstract: *This document is the subject of d14m*

Contents

Chapter I

GENERAL RULES

- READ THE GENERAL RULES CAREFULLY !!

    ○ You will have no possible excuse if you end up with a 0 because you didn't follow one of the general rules.

- GENERAL RULES :

    ○ If you do half the exercises because you have comprehension problems, it's okay, it happens. But if you do half the exercises because you're lazy, and leave at 2PM, you WILL have problems. Do NOT push your luck.

    ○ Every function implemented in a header, or unprotected header, means 0 to the exercise.

    ○ Every class MUST have a constructor and a destructor.

    ○ Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.

    ○ The imposed filenames must be PRECISELY respected, as well as class, function and method names.

    ○ Remember : You're coding in C++ now, and not in C. Therefore, the following functions are FORBIDDEN, and their use will be punished by a -42, no questions asked:

        * `*alloc`

        * `*printf`

        * `free`

        * `open, fopen, etc ...`

○ Files associated with a class will be CLASS_NAME.h and CLASS_NAME.cpp (If applicable), unless specified otherwise.

○ Turn-in directories are ex00, ex01, ..., exN

○ Any use of `friend` will be punished by a -42, no questions asked.

○ Read the examples CAREFULLY. They might require things the subject does not say...

○ These exercises require that you must create a lot of classes, but most of them are VERY short. So don't be lazy!

○ Read ENTIRELY the subject of an exercise before starting it!

○ THINK. Please.

○ THINK. By Odin !

○ T.H.I.N.K ! For Pony !

- COMPILATION OF THE EXERCISES :

  ○ The Koalinette compiles your code with the following flags : `-W -Wall -Werror`

  ○ To avoid compilation problems with the Koalinette, include every required headers in your headers.

  ○ Note that none of your files must contain a `main` function. We will use our own to compile and test your code.

  ○ This subject may be modified up to 4 hours before turn-in time. Refresh it regularly !

  ○ The turn-in dirs are (SVN REPOSITORY - `piscine_cpp_d14m-promotion-login_x/exN`) , N being the exercise number

Chapter II

Exercise 0

 KOALA	Exercise : 00	points : 3
	Exercise 0	
Turn-in directory: (SVN REPOSITORY - `piscine_cpp_d14m-promo-login_x`)/ex00		
Compiler: `g++`	Compilation flags: `-W -Wall -Werror`	
Makefile: `No`	Rules: `n/a`	
Files to turn in : `Fruit.h, Fruit.cpp, Lemon.h, Lemon.cpp, Banana.h, Banana.cpp, FruitBox.h, FruitBox.cpp, FruitNode.h`		
Remarks : `n/a`		
Forbidden functions : `Everything except new and delete`		

Fruits are good, eat them. They are full of little good vitamins which make a lot of good things into our small bodies tired by this hard pool.

But before having the possibility to taste a delicious vitaminized fruit juice, some work has to be done.

Start by implementing the following classes:
- `Fruit`
- `Lemon`
- `Banana`

Be sure to have a coherent inheritance tree and that the code below compiles:

```
$> cat main.cpp
int main(void)
{
    Lemon l;
    Banana b;
    std::cout << l.getVitamins() << std::endl;
    std::cout << b.getVitamins() << std::endl;
    std::cout << l.getName() << std::endl;
    std::cout << b.getName() << std::endl;
    Fruit& f = l;
```

```
11        std::cout << f.getVitamins() << std::endl;
12        std::cout << f.getName() << std::endl;
13        return 1337;
14 }
15 $>./a.out | cat -e
16 3$
17 5$
18 lemon$
19 banana$
20 3$
21 lemon$
```

All specializations of the class `Fruit` will have to initialize, during their construction, an attribute of type `int` declared in `Fruit` and containing the number of vitamins.

This attribute must be called _vitamins

`getName()` will return an instance of `std::string` containing the name of the fruit. Each fruit will have its own version of `getName()`.

We should not be able to reify the class `Fruit`.

Now, we need to build a `FruitBox` because we need a lot of vitamins which means many fruits. And with only two hands, it's not very convenient to carry lots of fruits.

Our `FruitBox` will be a container of `Fruit`, implemented as a linked list. I want a `FruitBox` which has the following member functions (`const` should be added where it's due):

- `FruitBox(int size);` // builds a `FruitBox` that can hold `size` fruits

- `int nbFruits();` // returns the number of `Fruit` currently into the `FruitBox`

- `bool putFruit(Fruit* f);` // adds a `Fruit` to the end of the `FruitBox`

- `Fruit* pickFruit();` // removes a `Fruit` from the `FruitBox` (the first that comes)

- `FruitNode* head();` // returns the head of the linked list

- `putFruit(Fruit *)` returns `false` if the `FruitBox` is full or if the `Fruit` instance is already present in the `FruitBox` .

- `pickFruit()` returns 0 (null pointer) if the `FruitBox` is empty.

- `head()` returns 0 (null pointer) if the `FruitBox` is empty.

In order to manipulate the `Fruit` as a linked list, we will have to encapsulate them into a `FruitNode` structure. You're on your own regarding the implementation, I want at least a `next` attribute.

I don't want to know how the `FruitBox` works, I just want to carry several `Fruit` , many fruit, to have more and more vitamins. Be careful, a `FruitBox` cannot be copied.

Chapter III

Exercise 1

		points : 4

Fruits
Turn-in directory: (SVN REPOSITORY - `piscine_cpp_d14m-promo-login_x`)/ex01

Compiler: `g++`	Compilation flags: `-Wextra -Werror -Wall`
Makefile: `No`	Rules: `n/a`

Files to turn in : `Fruit.h`, `Fruit.cpp`, `Lemon.h`, `Lemon.cpp`, `Banana.h`, `Banana.cpp`, `FruitBox.h`, `FruitBox.cpp`, `FruitNode.h`, `LittleHand.h`, `LittleHand.cpp`, `Lime.h`, `Lime.cpp`
Remarks : `n/a`
Forbidden functions : `Everything`

Good news, we have a big stock of `FruitBox` , enough to make a fucking Fruit'Party.

The problem is that all fruits are unsorted :'(, so we'll have to place all `Fruit` of the same kind together.

Be careful, we have a new type of `Fruit` , the `Lime` which inherits from the `Lemon` of course. It is poor in vitamins (only 2). And its little name is "lime".

We'll manually sort all `FruitBox` . The class `LittleHand` will implement all this boring work. It will own the following static member function:

```
void sortFruitBox(FruitBox& unsorted,
                  FruitBox& lemons,
                  FruitBox& bananas,
                  FruitBox& limes);
```

This function will move into the corresponding `FruitBox` all the `Fruit` of the disorganized `FruitBox` . All pulled fruits from the disorganized `FruitBox` which cannot be placed into any `FruitBox` (not the good type of `Fruit` or the `FruitBox` is full) will simply be put back into the disorganized `FruitBox` .

Chapter IV

Exercise 2

	Exercise : 02		points : 4

Eat my Coconut
Turn-in directory: (SVN REPOSITORY - `piscine_cpp_d14m-promo-login_x`)/ex02

Compiler: `g++`	Compilation flags: `-Wextra -Werror -Wall`
Makefile: `No`	Rules: `n/a`

Files to turn in : `Fruit.h`, `Fruit.cpp`, `Lemon.h`, `Lemon.cpp`, `Banana.h`, `Banana.cpp`, `FruitBox.h`, `FruitBox.cpp`, `FruitNode.h`, `LittleHand.h`, `LittleHand.cpp`, `Lime.h`, `Lime.cpp`, `Coconut.h`, `Coconut.cpp`
Remarks : `n/a`
Forbidden functions : `Everything except new and delete`

During your Fruit'Party, I took the opportunity to negociate some disorganized fruits from a really cheap wholesaler for you.

He also offered a new `Fruit` , the `Coconut` (which is a kind of `Fruit`). Its milk provides us with 15 wonderful vitamins, and with its pretty name "coconut", this nice fruit fit right in our fruit juices. I asked him to send us a large quantity of `Coconut` . However, he only delivers by packs so we'll have to receive `Coconut` and arrange them into `FruitBox` .

Once again your little hands will get down to work. `LittleHand` is now extended with a new static class member function taking a `Coconut` pack as parameter and returning an array of `FruitBox` , which can each contain 6 `Coconut` (it's a quite big thing). Here is the prototype of the function:

- `FruitBox * const * organizeCoconut(Coconut const * const * coconuts_packet);`

    - `coconuts_packet` is a `Coconut` array terminated by a null pointer.

    - `organizeCoconut` returns a (dynamically allocated) array of `FruitBox` (dynamically allocated), terminated by a null pointer.

Once again to be clear: if the little hands receive an array of 25 (pointers to) `Coconut`, they will return an array of 5 (pointers to) `FruitBox`. 4 of them will be full and the last one will contain only one `Coconut`.

Hurry up storing all this because vitamins fade away faster than we think!

Do not change the prototype of `organizeCoconut` for any reason, otherwise the delivery will not work anymore.

Chapter V

Exercise 3

	Exercise : 03	points : 5
For more vitamins, mix fruits!	

Turn-in directory: (SVN REPOSITORY - piscine_cpp_d14m-promo-login_x)/ex03	
Compiler: g++	Compilation flags: -Wextra -Werror -Wall
Makefile: No	Rules: n/a
Files to turn in : Fruit.h, Fruit.cpp, Lemon.h, Lemon.cpp, Banana.h, Banana.cpp, FruitBox.h, FruitBox.cpp, FruitNode.h, LittleHand.h, LittleHand.cpp, Lime.h, Lime.cpp, Coconut.h, Coconut.cpp, Mixer.h, Mixer.cpp	
Remarks : n/a	
Forbidden functions : Everything	

We finally have enough stuff to manage our vitaminized fruits. Let's mix all of that in order to obtain a quality fruit juice, so fresh and very energizing.

However, we'll need to mix our juice. To do this job, we need a... mixer.

All we have is an old mixer made of odds and ends. Its interface is not the easiest and we'll have to do some manual labour to connect it electrically.

So we have a `MixerBase`, base of our completely broken `Mixer`, which doesn't have any connecting cable and even less of a mixing blade... We'll have to deal with that.

This class is declared in `MixerBase.h` so you should not change it. (the correction script will use its own version anyway and we'll also use our own definition of all the member functions of this class)

Here is how the visible part of our poor `MixerBase` looks like:

```
class MixerBase
{
        public:
```

```
             MixerBase();
             int mix(FruitBox&) const;

     protected:
             bool _plugged;
             int (*_mixfunc)(FruitBox&);
};
```

We know that, by default, the mixer isn't plugged in and doesn't have any way to mix.

We'll have to specialize all of this so that, on one hand, we initialize the function pointer with a function that will handle the mixing of the `FruitBox` filled with fruits passed as parameter, and on the other hand, we provide a way to electrically connect the `Mixer`.

The `Mixer` must have a member function with the name of your choice which electrically connects the mixer. The mixing function itself will have to return the sum of all vitamins from `Fruit` passed as parameter for an unsurpassed cool and energizing juice.

Even if the mixer provides a way to be electrically connected, you'll have to plug it with you little hands. In `LittleHand`, add a static member function for the `MixerBase` passed as parameter:

```
void plugMixer(MixerBase& mixer);
```

Given that you have crafted only one type of `MixerBase`, you can be sure that `plugMixer` will take the real type `Mixer` as parameter.

I'm looking forward to all these wonderful and delicious juices that we'll be able to savor! I'm already feeling full of energy!

Chapter VI

Exercise 4

	Exercise : 04	points : 4
	A little help	
	Turn-in directory: (SVN REPOSITORY - `piscine_cpp_d14m-promo-login_x`)/ex04	
	Compiler: `g++`	Compilation flags: `-Wextra -Werror -Wall`
	Makefile: `No`	Rules: `n/a`
	Files to turn in : `Banana.cpp`, `Banana.h`, `FruitBox.cpp`, `FruitBox.h`, `Lemon.cpp`, `Lemon.h`, `Lime.cpp`, `Lime.h`, `Coconut.cpp`, `Coconut.h`, `Fruit.cpp`, `Fruit.h`, `FruitNode.h`, `LittleHand.cpp`, `LittleHand.h`, `Mixer.cpp`, `Mixer.h`	
	Remarks : `n/a`	
	Forbidden functions : `None`	

Doesn't it make you feel kinda weird to swallow so much vitamins? I don't know but for me, it gives me the wish to change the world, to create something new... But unlike the plasmidic genetic modifications as in the great game Bioshock, I want to do something real, I promise to never use them inappropriately and like Google's motto (don't be evil), I will make the world better, healthier, people will be happy. And the white rabbit has found you. I want to modify fruits, yes, to increase their vitaminic potential. I want more vitamins, more and more (in order to make the world better). We'll have so much good and pure vitamins, that our chakras will open, and then, like your mom who tucks your in your duvet, the world will slowly be covered by a sweet and peaceful layer of whipped cream (with strawberries on top).

So we'll find a way to directly modify fruits, we are going to dig into the heart of the matter, and we will conquer the world! (to make it better)

One time again, you'll use your little hands to do this delicate operation. Therefore you will add the following static member function to `LittleHand` :

```
void injectVitamin(Fruit& fruit, int quantity);
```

This function will inject (replace) into a `fruit` the `quantity` of vitamins passed as parameter. Try to use the following structure like in Matrix to do like Neo and modify

the matter:

```
struct InTheMatrixFruit
{
  virtual ~InTheMatrixFruit();
  int vitamins;
};
```

Now, we are almost equal with the great gods of C++, we have the power to modify the matter! The world and the future is ours!!