B3 - C++ Pool

B-CPP-300

Rush 2

Santa Claus

2.0

Rush 2

repository name: cpp_rush2_*$ACADEMICYEAR*
repository rights: ramassage-tek
language: C++

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).

All your exercises will be compiled with `g++` and the `-Wall -Wextra -Werror` flags.

The `*alloc`, `free`, `*printf`, `open` and `fopen` functions, as well as the `using namespace` keyword, are forbidden in C++.

Santa Claus has made an official request for your school to systemize his gift wrapping chain.
He is tired of all these stupid elves and would like to make the whole process automatic.
He demands that you design a chain packaging simulator.

This subject is deliberately opaque.
You must reorganize things logically and deduce the implied instructions to meet all requirements.
Reading the subject ENTIRELY is necessary to understand all the interactions between the various elements.

Little Pony and Teddy

Santa has given you a list describing what was, up until now, done by the elves on the gift wrapping production chain.
Your project leader has taken care of the functional analysis, as well as most of the design for you.
He provides you with a list of indications and constraints to implement the different parts of the simulator:

- a `Teddy` is a `Toy`,
- a `LittlePony` is also a `Toy`,
- a `Teddy` screams *"gra hu"* when picked up (through the `isTaken` method),
- a `LittlePony` screams *"yo man"* when picked up (through the `isTaken` method),
- a `Toy` is an `Object`.

The project leader asks that you implement the `Object`, `Toy`, `Teddy` and `LittlePony` classes.
In addition to all this, he insists that you implement the following function:

```
Object **MyUnitTests();
```

This function must assert it is possible to create a `LittlePony` with *"happy pony"* as its title, and a `Teddy` titled *"cuddles"*.
These two objects must be returned in an array of 2 elements.

A Box Story

Now that you have toys, it's time to take care of wrapping gifts:

- a `Box` is a `Wrap`,
- a `GiftPaper` is also a `Wrap`,
- elves wrap gifts via a `Wrap`'s `wrapMeThat` method,
- elves must be able to put an `Object` (and only one!) into a `Box` by wrapping it into the `Box`, but only if the `Box` is open,
- a closed `Box` cannot wrap,
- a `GiftPaper` doesn't have to be open to wrap,
- a `Wrap` which already contains an `Object` cannot wrap anymore,
- elves can open a `Wrap` to access the contained `Object`, through the `Wrap`'s `openMe` method,
- once a `Wrap` is open and the object has been taken by an `Elf`, the `Wrap` is empty once again and can wrap a new `Object`,
- elves can close open `Boxes`, through the `Wrap`'s `closeMe` method,
- when elves take a `Wrap`, they say *"whistles while working"*,
- when elves wrap objects, the say *"tuuuut tuuut tuut"*.

The team leader demands that you implement the `Wrap`, `Box` and `GiftPaper` classes.
Implement also the following function:

```
Object *MyUnitTests(Object **);
```

The function takes as parameter a null-terminated array of 3 elements, respectively containing a `Teddy`, a `Box` and a `GiftPaper`.
It must put the `Teddy` into the `Box`, and the `Box` into the `GiftPaper`, and return the final present.

Error cases must generate explicit messages on the error output.

Of course, feel free to add any additional tests you need.

A ROLLING CARPET GATHERS NO MOSS

Time to add a workstation:

- elves have a `Table` in front of them, and a `ConveyorBelt` by their side,
- nothing can be put on the `ConveyorBelt` if there is already something on it,
- elves can `put` and `take` any `Object` on the `Table` or the `ConveyorBelt`,
- when there is no more room on the `Table`, it collapses.
  It can hold up to 10 `Objects`,
- elves receive `Wraps` by pressing the ***"IN"*** button of the `ConveyorBelt` using their `Hand`,
- elves send what's on the `ConveyorBelt` to Santa by pressing its ***OUT*** button,
- a `Wrap` sent to Santa vanishes, meaning there is once again space on the `ConveyorBelt`,
- elves can `look` at the `Table` to see what's on it. In return, they get an array of the titles of the various `Objects`. The last element in the array is `null`,
- pushing a button of the `ConveyorBelt` when the input/output hasn't been initialized yet is an error.

The project leader insists that you must write the `ITable` and `IConveyorBelt` interfaces, as well as the `PapaXmasTable` and `PapaXmasConveyorBelt` classes implementing those interfaces.
Santa's table and conveyor belt have everything required to make 2 gifts. The organization/distribution of wraps/toys between the conveyor belt and the table is up to you.

You must also provide the following functions:

```
ITable *createTable();
IConveyorBelt *createConveyorBelt();
```

These functions let clients reify (instantiate) the two objects.

The error cases must generate explicit messages on the error output.

A JOB FOR AN ELF

It is now time to simulate the actual job of the elves.

Create an `IElf` interface, describing all the actions elves may do.
These were all specified throughout the subject.
Once that's done, implement the `PapaXmasElf` class, implementing the interface.

When an elf is told to wrap a gift, it does so all by itself, using the `Table` and `ConveyorBelt` to wrap up a gift and send it by pressing the `ConveyorBelt`'s ***OUT*** button.

Here's the box

Let's complicate things a little.
Implement the `TableRand` and `ConveyorBeltRand` classes, that provide a random set of `Objects`.
Your `Elf` must be able to make as many presents as possible using these `Objects`, and send them to Santa.
When it can't do anything anymore, it yells:

```
o'pa ere's somin' wron' in da box!
```

Oh, by the way, there's something really weird with the `ConveyorBeltRand`.
It makes a lot of noise.
The goblin mechanic that delivered it said it talks.
More specifically, he said

```
e speex'n'XML
```

Ruddy goblin accents, I never understand 'em.

The fact of the matter is, your `ConveyorBeltRand` must "speex'n'XML": it must describe the gift it is sending, from the outside in, in XML.
The goblin says it's something called **"serialization"**, for what it's worth.

Is Santa quantum?

It is now time to implement Santa Claus, as a separate program called `santa`.
Santa is lazy, and simply waits for gifts to put them into the sleigh.
A gift is the result of an XML serialization of your objects.
These, stored in files, are passed as parameters to Santa on the command line:

```
./santa gift1.xml gift2.xml gift3.xml
```

You will have to show the inspector that your Santa loads each gift from XML into memory using what the ol' goblin called ''deserialization''.

To do so, you must take each gift, open it, take the toy inside it and put it back into the box.
Quality control is very important for Santa: to know wether Shrodinger's cat is still alive, he simply opens the box!
When he takes the toy out of the box, it yells (useful to know what's inside, right?).

The Warp Machine, or one more step towards dementia

Gifts are coming out of a device called the **Warp Machine**.
Any gift sent by the new `ConveyorBelt` with the ***OUT*** button is received by the warp machine, through one of these parts that look like a big magic sock.
This sacred machine expects a mystical number before it connects to the elves' magical conveyor belt, number provided by Santa when it is launch in warp mode with the "-w" flag (as in *"Warp Me Up Scotty"*).

There is an old saying among the dwarves, which goes something like this:

> *To connect the magical sock to the warp,*
> *this sock must be "figurated in 'u di pee multi-caste!?".*
> *The mystical number works by quadruplet.*
> *The first one always starts between 234 and 239.*
> *Ask my cousin gougle for more information.*

I didn't say it was a nice saying.
Just an old one.

Anyways.
Modify your Santa Claus program so that it takes this mystical number as parameter and properly configures the warp machine connected with the magic sock.
In addition, the goblin mechanic made a new version of `ConveyorBeltRand`: the `MagicalCarpet`.
Just like the `ConveyorBeltRand`, "e speex'n'XML", and the noise it produces is sent out into the warp.
The XML is then captured by Santa's sock.
This lets Santa know what's going on on the `ConveyorBelt`.
He can then get the gift out of the wrap, and proves everything worked correctly to the inspector by grabbing the toy inside it.

This means many elf production lines can all send their gifts to Santa at the same time!

If several Santas use the same mystical number, they will all receive the gifts destined to this number, which is very convenient for duplicating gifts.

Several Santa Claus?
We told you he was quantum!
Pay attention!

Bonus: get together with other groups and use the same XML elements.
This will make it possible for your elf production lines to work with their Santas, and vice versa!