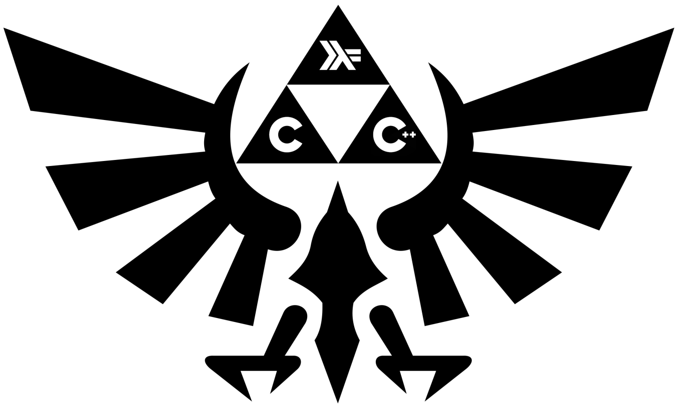
B3 - Paradigms Seminar

B-PDG-300

Day 09

The Legend of Gildas



3.0

Day 09

language: C++

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.

All your exercises will be compiled with `g++` **and the** `-std=c++20 -Wall -Wextra -Werror` **flags**, unless specified otherwise.

All output goes to the standard output, and must be ended by a newline, unless specified otherwise.

None of your files must contain a `main` function, unless specified otherwise.
We will use our own `main` functions to compile and test your code. It will include your header files.

There are no subdirectories to create for each exercice. Every file must be at the root of the repository.

Read the examples CAREFULLY. They might require things that weren't mentioned in the subject...

The `*alloc`, `free`, `*printf`, `open` and `fopen` functions, as well as the `using namespace` keyword, are forbidden in C++.
By the way, `friend` is forbidden too, as well as any library except the standard one.

Unit Tests

It is highly recommended to test your functions as you implement them. It is common practice to create and use what are called **unit tests**.

From now on, we expect you to write unit tests for your functions (when possible). To do so, please follow the instructions in the **"How to write Unit Tests"** document on the intranet, available here.
For them to be executed and evaluated, put a `Makefile` at the root of your directory with the `tests_run` rule as mentionned in the documentation linked above.

Exercise 0 - The Peasant

Turn in: `Peasant.hpp`, `Peasant.cpp`

Create a `Peasant` class representing a basic character.
A peasant has a `name` and `power` points.
Those two values are required to create a `Peasant`.
A peasant with no name or power points is like a pony that isn't pink: nonsense.
Anyone can ask a `Peasant` for their name and power points.
No matter what, `Peasant`'s health and power points are capped between 0 and 100.
When a `Peasant` is created, its health is set to its maximum values and power to the given value.

```
Peasant(const std::string &name, int power);
const std::string &getName() const;
int getPower() const;
int getHp() const;
```

A `Peasant` know some basic techniques.
Using a technique costs power points.
If a `Peasant` have 0 health points or doesn't have enough energy when using a technique, it prints:

```
[name] is out of combat.
```

or

```
[name] is out of power.
```

and the technique's effect is cancelled.

```
int attack()
```

Cost: 10 power points
Damage: 5
Output: *"[name] tosses a stone."*
Return: the number of damage points dealt by the attack

```
int special()
```

Cost: 0 power point
Damage: 0
Output: *"[name] doesn't know any special move."*
Return: the number of damage points dealt by the special move

```
void rest()
```

Cost: 0 power point
Effect: restores 30 power points
Output: *"[name] takes a nap."*

As strong as they are, heroes can also take damage:

```
void damage(int damage)
```

This functions outputs

```
[name] takes [damage] damage.
```

If a `Peasant` takes too much damage and its health points reach 0, the peasant screams

```
[name] is out of combat.
```

Upon creation, the peasant says:

```
[name] goes for an adventure.
```

Upon destruction, the peasant says:

```
[name] is back to his crops.
```

You're free to add as many private/protected methods as you want.

Here is a sample `main` function and its expected output:

```
int main(void)
{
    Peasant peasant("Gildas", 42);

    peasant.damage(50);
    peasant.damage(100);
    peasant.damage(200);
    peasant.rest();
}
```

Terminal

```
~/B-PDG-300> g++ -std=c++20 -Wall -Wextra -Werror *.cpp && ./a.out | cat -e
Gildas goes for an adventure.$
Gildas takes 50 damage.$
Gildas is out of combat.$
Gildas is out of combat.$
Gildas is out of combat.$
Gildas is back to his crops.$
```

Exercise 1 - The Knight

Turn in: `Peasant.hpp/cpp`, `Knight.hpp/cpp`

Create a `Knight` class that inherits from the `Peasant` class.
Just like its parent, a `Knight` must be constructed with a name and power points.
A `Knight` learned new techniques during its training:

```
int attack()
```

Cost: 10 power points
Damage: 20
Output: *“[name] strikes with his sword.”*
Return: the number of damage points dealt by the attack

```
int special()
```

Cost: 30 power point
Damage: 50
Output: *“[name] impales his enemy.”*
Return: the number of damage points dealt by the special move

```
void rest()
```

Cost: 0 power point
Effect: restores 50 power points
Output: *“[name] eats.”*

Upon creation, the knight says:

```
[name] vows to protect the kingdom.
```

Upon destruction, the knight says:

```
[name] takes off his armor.
```

Here is a sample `main` function and its expected output:

```
int main(void)
{
    Knight knight("Arthur", 20);

    knight.attack();
    knight.special();
    knight.rest();
    knight.special();
    knight.damage(50);
}
```

Terminal

```
~/B-PDG-300> g++ -std=c++20 -Wall -Wextra -Werror *.cpp && ./a.out | cat -e
Arthur goes for an adventure.$
Arthur vows to protect the kingdom.$
Arthur strikes with his sword.$
Arthur is out of power.$
Arthur eats.$
Arthur impales his enemy.$
Arthur takes 50 damage.$
Arthur takes off his armor.$
Arthur is back to his crops.$
```

Exercise 2 - The Enchanter

Turn in: `Peasant.hpp/cpp`, `Enchanter.hpp/cpp`

Create an `Enchanter` class that inherits from the `Peasant` class.
Just like its parent, an `Enchanter` must be constructed with a name and power points.
An `Enchanter` studies its spell book for year to learn new techniques:

```
int attack()
```

Cost: 0 power points
Damage: 0
Output: *"[name] doesn't know how to fight."*
Return: the number of damage points dealt by the attack

```
int special()
```

Cost: 50 power point
Damage: 99
Output: *"[name] casts a fireball."*
Return: the number of damage points dealt by the special move

```
void rest()
```

Cost: 0 power point
Effect: restores 100 power points
Output: *"[name] meditates."*

Upon creation, the enchanter says:

```
[name] learns magic from his spellbook.
```

Upon destruction, the enchanter says:

```
[name] closes his spellbook.
```

Here is a sample `main` function and its expected output:

```
int main(void)
{
    Enchanter enchanter("Merlin", 20);

    enchanter.attack();
    enchanter.special();
    enchanter.rest();
    enchanter.special();
    enchanter.damage(50);
}
```

Terminal

```
~/B-PDG-300> g++ -std=c++20 -Wall -Wextra -Werror *.cpp && ./a.out | cat -e
Merlin goes for an adventure.$
Merlin learns magic from his spellbook.$
Merlin doesn't know how to fight.$
Merlin is out of power.$
Merlin meditates.$
Merlin casts a fireball.$
Merlin takes 50 damage.$
Merlin closes his spellbook.$
Merlin is back to his crops.$
```

EXERCISE 3 - THE PRIEST

Turn in: `Peasant.hpp/cpp`, `Enchanter.hpp/cpp`, `Priest.hpp/cpp`

Create a `Priest` class that inherits from the `Enchanter` class.
Just like a `Peasant`, a `Priest` must be constructed with a name and power points.
A `Priest` is just like an `Enchanter`, except he can heal himself when resting.

```
void rest()
```

Cost: 0 energy point
Effect: restores 100 power points and 100 health points.
Output: *“[name] prays.”*

Upon creation, the priest says:

```
[name] enters in the order.
```

Upon destruction, the priest says:

```
[name] finds peace.
```

Here is a sample `main` function and its expected output:

```
int main(void)
{
    Priest priest("Trichelieu", 20);

    priest.attack();
    priest.special();
    priest.rest();
    priest.damage(50);
}
```

```
~/B-PDG-300> g++ -std=c++20 -Wall -Wextra -Werror *.cpp && ./a.out | cat -e
Trichelieu goes for an adventure.$
Trichelieu learns magic from his spellbook.$
Trichelieu enters in the order.$
Trichelieu doesn't know how to fight.$
Trichelieu is out of power.$
Trichelieu prays.$
Trichelieu takes 50 damage.$
Trichelieu finds peace.$
Trichelieu closes his spellbook.$
Trichelieu is back to his crops.$
```

EXERCISE 4 - THE PALADIN

Turn in: `Peasant.hpp/cpp`, `Knight.hpp/cpp`, `Enchanter.hpp/cpp`, `Priest.hpp/cpp`, `Paladin.hpp/cpp`

Create a `Paladin` class that inherits from both the `Knight` and `Priest` class.
Just like a `Peasant`, a `Paladin` must be constructed with a name and power points.
A `Paladin` uses technique from various classes:

- `attack` from the `Knight`
- `special` from the `Enchanter`
- `rest` from the `Priest`

Upon creation, the paladin says:

```
[name] fights for the light.
```

Upon destruction, the paladin says:

```
[name] is blessed.
```

Here is a sample `main` function and its expected output:

```cpp
int main(void)
{
    Paladin paladin("Uther", 99);

    paladin.attack();
    paladin.special();
    paladin.rest();
    paladin.damage(50);
}
```

Terminal

```
~/B-PDG-300> g++ -std=c++20 -Wall -Wextra -Werror *.cpp && ./a.out | cat -e
Uther goes for an adventure.$
Uther vows to protect the kingdom.$
Uther learns magic from his spellbook.$
Uther enters in the order.$
Uther fights for the light.$
Uther strikes with his sword.$
Uther casts a fireball.$
Uther prays.$
Uther takes 50 damage.$
Uther is blessed.$
Uther finds peace.$
Uther closes his spellbook.$
Uther takes off his armor.$
Uther is back to his crops.$
```

Exercise 5 - The Character Interface

Turn in: Peasant.hpp/cpp, Knight.hpp/cpp, Enchanter.hpp/cpp, Priest.hpp/cpp, Paladin.hpp/cpp, ICharacter.hpp

We really have a lot of classes for our characters.
We need something generic from which every character should derive at some point of their inheritance tree.
This way, we could control them all using a single type!

Create a ICharacter class.
ICharacter must contain only pure virtual methods, except its constructor/destructor.

Here is a sample main function and its expected output:

```
int main(void)
{
    ICharacter *peasant = new Peasant("Gildas", 42);
    ICharacter *knight = new Knight("Arthur", 20);
    ICharacter *enchanter = new Enchanter("Merlin", 20);
    ICharacter *priest = new Priest("Trichelieu", 20);
    ICharacter *paladin = new Paladin("Uther", 99);

    peasant->attack();
    knight->special();
    enchanter->rest();
    priest->damage(21);
    std::cout << paladin->getName() << ": "
              << paladin->getHp() << " health points, "
              << paladin->getPower() << " power points."
              << std::endl;

    delete peasant;
    delete knight;
    delete enchanter;
    delete priest;
    delete paladin;
}
```

Terminal

```
~/B-PDG-300> g++ -std=c++20 -Wall -Wextra -Werror *.cpp && ./a.out | cat -e
Gildas goes for an adventure.$
Arthur goes for an adventure.$
Arthur vows to protect the kingdom.$
Merlin goes for an adventure.$
Merlin learns magic from his spellbook.$
Trichelieu goes for an adventure.$
Trichelieu learns magic from his spellbook.$
Trichelieu enters in the order.$
Uther goes for an adventure.$
Uther vows to protect the kingdom.$
Uther learns magic from his spellbook.$
Uther enters in the order.$
Uther fights for the light.$
Gildas tosses a stone.$
Arthur is out of power.$
Merlin meditates.$
Trichelieu takes 21 damage.$
Uther: 100 health points, 99 power points.$
Gildas is back to his crops.$
Arthur takes off his armor.$
Arthur is back to his crops.$
Merlin closes his spellbook.$
Merlin is back to his crops.$
Trichelieu finds peace.$
Trichelieu closes his spellbook.$
Trichelieu is back to his crops.$
Uther is blessed.$
Uther finds peace.$
Uther closes his spellbook.$
Uther takes off his armor.$
Uther is back to his crops.$
```

Exercise 6 - The Potions

Turn in: Peasant.hpp/cpp, Knight.hpp/cpp, Enchanter.hpp/cpp, Priest.hpp/cpp, Paladin.hpp/cpp, ICharacter.hpp, IPotion.hpp, HealthPotion.hpp/cpp, PowerPotion.hpp/cpp, PoisonPotion.hpp/cpp

Characters sometime need a little pick-me-up.
Alchemists have studied for centuries to create potions of various effects.
Create a potion interface `IPotion`, you are free to puts the methods that you need inside.

Create the 3 following potions:

- `HealthPotion`: restores 50 health points
- `PowerPotion`: restores 50 power points
- `PoisonPotion`: remove 50 health points

Characters can drink potions using their `drink` methods:

```
void drink(const [...]& potion);
```

Characters can drink a potion even when out of combat. Potions can be used multiple times.

They display the following when drinking:

- Health potion: `[name] feels rejuvenated.`
- Power potion: `[name]'s power is restored.`
- Poison potion: `[name] has been poisoned.`

When the potion type is unknown, they print:

```
[name] drinks a mysterious potion.
```

Here is a sample `main` function and its expected output:

```cpp
int main(void)
{
    ICharacter *peasant = new Peasant("Gildas", 42);
    PoisonPotion poison_potion;
    HealthPotion health_potion;
    IPotion& potion = health_potion;

    std::cout << peasant->getName() << ": " << peasant->getHp() << "HP, "
              << peasant->getPower() << " PP." << std::endl;
    peasant->drink(poison_potion);
    std::cout << peasant->getName() << ": " << peasant->getHp() << "HP, "
              << peasant->getPower() << " PP." << std::endl;
    peasant->drink(potion);
    std::cout << peasant->getName() << ": " << peasant->getHp() << "HP, "
              << peasant->getPower() << " PP." << std::endl;

    delete peasant;
}
```

Terminal

```
~/B-PDG-300> g++ -std=c++20 -Wall -Wextra -Werror *.cpp && ./a.out | cat -e
Gildas goes for an adventure.$
Gildas: 100HP, 42 PP.$
Gildas has been poisoned.$
Gildas: 50HP, 42 PP.$
Gildas drinks a mysterious potion.$
Gildas: 100HP, 42 PP.$
Gildas is back to his crops.$
```