TD 4 : RPG / JdR

Exercice 1 : Pile et Tas

Écrire un programme qui alloue
- 1 kilobyte sur la pile (mémoire `stack`),
- 1 kilobyte sur le tas (mémoire `heap`).
- Tester.

Modifier le programme pour allouer en plus
- 1 megabyte sur la pile,
- 1 megabyte sur le tas.
- Tester.

Modifier le programme pour allouer en plus
- 1 gigabyte sur la pile,
- 1 gigabyte sur le tas.
- Tester.

Modifier le programme pour allouer en plus
- 10 gigabyte sur la pile,
- 10 gigabyte sur le tas.
- Tester.

Exercice 2 : A

Écrire une classe A ayant

- un attribut value de type double
- deux constructeurs :
  - Un constructeur avec un paramètre double initialisant l'attribut value.
  - Un constructeur de copie

    ```
    A(const A&)
    ```

  - Chaque constructeur doit écrire dans `cout` la description de son appel.
- Un destructeur
  - Le destructeur doit écrire dans `cout` la description de son appel.

Écrire un `main` qui

- instancie `a1` de type A dans la pile
- instancie `a2`, un autre A qui prend comme valeur la valeur de `a1` : `A a2 = a1;`
- instancie `a3` de type A dans le tas
- libère `a3` du tas

Écrire une fonction `f()` qui

- instancie `af1` de type A dans la pile
- instancie `af2` de type A dans le tas
- libère `af2` du tas

Ajouter l'appel de f dans le `main`.

Écrire le résultat attendu.

Faire tourner le programme et comparer le résultat attendu avec le résultat retourné.

Exercice 3 : String

Créer un classe String gérant une chaine de caractère sous forme d'une **zone mémoire allouée dans le tas**.

Écrire une **fonction** réservée à String qui permet de calculer une longueur de chaine de caractères C.

La classe String possède deux constructeurs :

- Un constructeur à partir d'une chaine de caractères.
- Un constructeur de copie.

Cette classe possède un opérateur de « cast » (transtypage) qui retourne un pointer de caractères constants.

```
Operator const char * () const {
    ...
}
```

Ne pas oublier de libérer la mémoire au moment opportun.

Exercice 4 : `Damageable`

- Écrire la classe `Damageable` ayant
  - comme attributs :
    - un `double` nommé `m_hitPoints` (points de vie).
      - `m_hitPoints` possède un accesseur et pas de mutateur.
    - un `String` nommé `m_name` (nom).
      - `m_name` possède un accesseur et un mutateur.
  - comme méthodes :
    - `void damage(int damage)` qui permet d'indiquer que des dégâts ont été infligés. Ces dégâts sont retirés des points de vie. Les points de vie ne peuvent pas descendre en dessous de 0.
    - `bool isDead()` qui indique si le « quelque chose » endommageable est mort ou cassé. Le « quelque chose » est mort ou cassé si ces points de vie sont égaux à 0.

    - `void healRepair(int heal)` qui soigne ou répare, si et seulement si le quelque chose n'est pas mort ou cassé. Dans ce dernier cas, il reste mort ou cassé.

- Écrire un fichier `TestDamageable`

    1. Créer un `Damageable` nommé « Knight » ayant 12 pdv (points de vie).
    2. Le chevalier subit 8 pts de dommage. Vérifier qu'il est vivant et que ses points de vie sont de 4.
    3. Un prêtre-médecin tente de le soigner le chevalier. Le chevalier récupère 6 pts de vie. Vérifier qu'il est vivant et que ses points de vie sont de 10.
    4. Le chevalier rencontre un dragon qui lui inflige 52 pts de dommage. Vérifier que le chevalier est mort et que ses points de vie sont de 0.
    5. Le prêtre-médecin tente de le soigner à nouveau. Il tente de lui octroyer 8 pts de vie. Vérifier que le chevalier est toujours mort et que ses points de vie sont toujours de 0.

Exercice 5 : NameSpace

- Réécrire le programme en déclarant la classe `Damageable` dans l'espace de nom `rpg`.
- Réécrire le test en utilisant la classe `Damageable` du le namespace `rpg`.