Développement Objet

TP 4

Exercice 1 Créer le programme ci-dessous (`exo1.cpp`) pour voir la différence entre le passage de paramètre par référence et le passage de paramètre par valeur.

```
1  void par_ref(string &s) {
2      int x;
3      cout << s.size() << endl;
4      for (int i = 0; i < s.size(); ++i) {
5          s[i] = 'b';
6      }
7      cout << "Pause par_ref :";
8      cin >> x; // Faire une pause.
9  }
10
11 void par_val(string s) {
12     int x;
13     cout << s.size() << endl;
14     for (int i = 0; i < s.size(); ++i) {
15         s[i] = 'b';
16     }
17     cout << "Pause par_val :";
18     cin >> x; // Faire une pause.
19 }
20
21 int main() {
22     int x;
23     string s;
24     for (int i = 0; i < 1024 * 1024 * 1024; ++i) {
25         s += 'a';
26     }
27     cout << "Pause 1 :";
28     cin >> x; // Faire une pause.
29     par_ref(s);
30     cout << "Pause 2 :";
31     cin >> x; // Faire une pause.
32     par_val(s);
33 }
```

Exercice 2 Répéter le programme (`exo2.cpp`) dans l'exemple sur la page 35 du support de cours. Vérifier l'utilisation de la mémoire avec ou sans `delete[]`.

Exercice 3 Le programme suivant concerne principalement l'allocation dynamique de la mémoire pour gérer un tableau de `std::string` : on va remplir ce tableau par les objets de type `std::string`, une fois que le nombre d'objets remplis (`size`) atteint la capacité `real` du tableau, on double la capacité de ce tableau.

La fonction `read()` est utilisée pour lire des mots. Cette fonction retourne le mot entré s'il n'est pas « //FIN » qui signifie la fin d'entrée. Si l'entrée est « //FIN », alors cette fonction retourne un mot vide. Réalisez cette fonction à la place (1).

Dans la fonction `main()`, on utilise la fonction `read()` pour lire le premier mot (ligne 31), puis continue à utiliser une boucle (ligne 32–35) pour lire les mots afin de remplir le tableau `words`. Plus précisément, dans cette boucle, la fonction `add(words, word)` ajoute un mot `word` au tableau `words`. En effet, cette fonction va d'abord tenter de grandir le

tableau (le premier paramètre) par l'appel de la fonction `resize(words)` puis ajouter le mot (le deuxième paramètre) au tableau en incrémentant le compteur `size`.

La fonction `resize()` joue le rôle essentiel de ce programme. Initialement, réalisez à la place (2), si aucun espace de mémoire a été alloué pour le tableau comme le paramètre `p`, dans le cas, la valeur de `p` est `nullptr` (pointeur null) dont `!p` est vrai, on crée un tableau de `std::string` avec 2 places par l'opérateur `new` puis affecte la capacité du tableau `real` par 2. Durant l'exécution du programme, on vérifie la capacité du tableau et le nombre d'éléments déjà remplis. Si `size == real`, on fait les étapes ci-dessous à la place (3) :

1. déclarer un nouveau pointeur de type `std::string`, par exemple, nommé `x` ;
2. allouer l'espace `real * 2` à `x` par l'opérateur `new` ;
3. multiplier la valeur de `real` par 2 ;
4. copier tous les éléments de `p` à `x` ;
5. libérer l'espace alloué à `p` par l'opérateur `delete[]` ;
6. stocker l'adresse mémoire du nouveau tableau (la valeur de `x`) dans `p`.

Pour vérifier, réaliser une boucle pour afficher chaque élément dans le tableau `words` à la place (4), et enfin, libérer l'espace mémoire alloué à la place (5).

Question (y répondez à la place (6)) : à quoi sert le type `*&` dans la déclaration de la fonction `resize(string *&p)` ? Peut-on le remplacé par `*` ou `&` ? Pourquoi ?

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int real = 0;
6  int size = 0;
7
8  string read() {
9      string word;
10     cout << "Entrer un mot (//FIN pour finir) : ";
11     // (1)
12     return word;
13 }
14
15 void resize(string *&p) {
16     if (!p) {
17         // (2)
18     }
19     if (size == real) {
20         // (3)
21     }
22 }
23
24 void add(string *&p, const string &s) {
25     resize(p);
26     p[size++] = s;
27 }
28
29 int main() {
30     string *words = nullptr;
```

```
31      string word = read();
32      while (word != "") {
33          add(words, word);
34          word = read();
35      }
36      // (4)
37      // (5)
38      return 0;
39  }
```