TD n° 5 (Correction)

Les structures

Le but de ce TD est de manipuler des structures (type `struct` en C++) qui est un type composé permettant de regrouper plusieurs données en une seule. L'accès aux composants (ou champs) d'une structure se fait en utilisant la "notation pointée".

Exercice 1. Le créateur d'un jeu souhaite pouvoir modéliser des personnages participants à des combats. Un personnage aura un nom, des points de vie, et un bouclier lui permettant d'arrêter les coups. La force des coups qu'il portera à son adversaire sera égale à son nombre de points de vie - 1 et il sera blessé s'il reçoit un coup d'une force supérieure à la force du bouclier qu'il porte. Un personnage blessé se verra retirer un nombre de point de vie égal à la différence entre la force du coup reçu et la force de son bouclier. La structure de données choisie pour modéliser un personnage est donc la suivante :

```
struct guerrier {
     string nom;
     int ptVie;
     int bouclier;
};
```

1. Écrire la procédure **créeGuerrier** qui permet d'initialiser un personnage, sachant qu'il doit disposer au départ d'exactement 10 points à répartir entre les points de vie du personnage (au moins deux points) et la force de son bouclier. Par exemple, on devra pouvoir saisir le nom du personnage, par exemple, "Elisa", et lui donner 7 points de vie. Dans ce cas, la procédure **créeGuerrier** devra lui affecter automatiquement un bouclier d'une force égale à 3 (10 - 7).

   Vous pouvez réutiliser, sans la redéfinir ni la réaliser, la fonction ci-dessous, qui reprend et étend avec un paramètre supplémentaire, celle vue dans le TD1,

   ```
   int litValeurBornee(string texteAEcrire, int min, int max);
   ```

   Cette fonction permet, après avoir affiché le `texteAEcrire` explicitant la valeur attendue. de saisir une valeur auprès de l'utilisateur, de vérifier qu'elle est bien comprise entre un `Min` et un `Max`, puis de la renvoyer quand elle respecte les bornes données.

   Correction :

   ```
   void creeGuerrier(guerrier &g) {
     cout << "donner le nom de votre personnage";
     cin >> g.nom ;
     g.ptVie = litValeurBornee("nbre de point de vie", 2, 10);
     g.bouclier = 10 - g.ptVie;
   }
   ```

2. Spécifier et réaliser la procédure **afficheGuerrier** permettant d'afficher l'état d'un guerrier, i.e. son nom, et ses différentes propriétés.

   Correction :

   ```
   void afficheGuerrier(guerrier g) {
     cout << "personnage : " << g.nom << endl;
     cout << "ptdeVie : " << g.ptVie << endl;
   ```

```
   cout << "bouclier : " << g.bouclier << endl;
}
```

3. Spécifier et réaliser la fonction **forceCoup** qui permet de retourner la force du coup porté par un guerrier vivant, calculée comme égale à son nombre de points de vie - 1.
   Correction :

```
int forceCoup(guerrier g) {
   return g.ptVie - 1;
}
```

4. Spécifier et réaliser la procédure **reçoitCoup** qui met à jour l'état d'un guerrier qui reçoit un coup d'une force donnée.
   Correction :

```
void recoitCoup(guerrier &g, int f) {
  int temp;
  if (f > g.bouclier) {
    temp = g.ptVie + g.bouclier - f;
    if (temp >= 0) G.ptVie = temp;
    else g.ptVie = 0;
  }
}
```

5. On suppose déjà réalisée la procédure

   ```
   void initialiseGuerrierAutomatique(guerrier G1; guerrier &G2);
   ```

   qui permet au concepteur de créer le guerrier G_2 qui combattra le guerrier G_1 créé par l'utilisateur du jeu. Réaliser le programme principal qui crée les deux guerriers, les affiche, enchaîne les coups qu'ils se portent jusqu'à ce que l'un des deux ne soit plus en état de se battre puis affiche le nom du gagnant et ses points de vie.

Correction :

```
int main() {
  guerrrier g1, g2;
  creeGuerrier(g1);
  initialiseGuerrierAutomatique(g1, g2);
  while (g1.ptVie > 1 and g2.ptVie > 1) {
    afficheGuerrier(g1);
    afficheGuerrier(g2);
    recoitCoup(g2, ForceCoup(g1));
    if (g2.ptVie >= 1) alors recoitCoup(g1, ForceCoup(g2));
  }
  cout << "Gagnant : ";
  if (g1.ptVie > g2.ptVie) afficheGuerrier(g1);
  else                     afficheGuerrier(g2);
}
```

Exercice 2. Une entreprise vend des fournitures de bureau (crayon, gomme, ciseaux,...). Chaque fourniture stockée dans la réserve est identifiée par un code numérique unique (ex : 345), un libellé exprimé sous la forme d'une chaîne de caractères (ex. stylo feutre pointe fine marque Boc), un prix d'achat unitaire au fournisseur (ex. 1,5 euro), le nombre d'items achetés au fournisseur et le nombre d'items déjà vendus.

Chaque fin d'année, l'entreprise effectue son inventaire, i.e. décompte pour chaque fourniture, le nombre d'items théoriquement en stock (c.à.d. le nombre d'items achetés au fournisseur moins le nombre d'items vendus) puis calcule le montant total du stock (somme des prix d'achat unitaire multipliés par les nombres d'items théoriquement en stock).

On vous demande de programmer les fonctions et procédures suivantes. À chaque question, vous devrez également programmer une petite procédure de test quand c'est possible ou de démonstration sinon.

1. une structure `Fourniture` adapté à la représentation des informations apparaissant dans l'énoncé ci-dessus ;
2. la procédure permettant de saisir (lire au clavier) les information relatives à une fourniture et de les transmettre en résultat ;
3. la procédure permettant d'afficher les informations relatives à une fourniture donnée ;
4. la procédure qui met à jour une fourniture f donnée, quand une vente de n unités de cette fourniture vient d'être effectuée ;
5. la procédure qui pour une fourniture f donnée, calcule et transmet en résultat le nombre d'items en stock et le montant correspondant ;
6. une structure `Catalogue` permettant de représenter l'ensemble des fournitures (au plus 500) vendues par l'entreprise.
7. la procédure permettant de rajouter une nouvelle fourniture f donnée dans un catalogue c donné et supposé non plein.
8. la procédure ou fonction `bilan` qui étant donné un catalogue c, affiche pour chaque fourniture, son libellé, le nombre d'items en stock et le montant correspondant, puis enfin, affiche la valeur totale du stock ;
9. la procédure permettant de retirer une fourniture de code donné d'un catalogue c donné.

Correction :

```
#include <iostream>
#include <cmath>
#include <string.h>
using namespace std;

#define ASSERT(test) if (!(test)) cout << "Test failed in file " << __FILE__ \
                << " line " << __LINE__ << ": " #test << endl

struct fourniture{
  string libelle;
  float prix;
  int code, nbAchat, nbVendu;
};

void litFourniture(fourniture &f){
  cout << "entrez le code, le libelle et le prix "<< endl;
  cin >> f.code >> f.libelle >> f.prix ;
  do {
    cout <<  "nb achetes et nb vendus " << endl;
```

```
    cin >> f.nbAchat >> f.nbVendu ;
  } while ((f.nbAchat < 0) or (f.nbVendu > f.nbAchat) );
}

void ecritFourniture(struct fourniture f){
  cout << "la fourniture " << f.libelle << " de code " << f.code << " commandee
       << f.nbAchat <<" fois, a ete vendue " << f.nbVendu <<" fois" <<endl;
}

// se rappeler l ordre dans lequel les elements ont ete introduits
// ces deux variables sont des variables globales pour les demos

fourniture f1= {"gomme", 1.10, 101, 20, 4};
fourniture f2= {"crayon", 0.50,1024, 20, 7};

void demoAffiche(){ ecritFourniture(f1); }

void maj(fourniture &f, int n){f.nbVendu+=n;}

void demoMaj(){
  maj(f1,2); ecrireFourniture(f1);}

void stock(fourniture  f, int &n, float &montant){
   n = f.nbAchat-f.nbVendu;
   montant = n * f.prix;}

void demoStock(){
  float m;int n;
  stock(f1,n,m);
  cout< <"stock de f1:" << n << " en euro, ca represente " << m << endl;}

const int TAILLE=500;
struct catalogue{
        fourniture tab[TAILLE];
        int nb;
};

void ajouteAuCatalogue(catalogue &c, fourniture f){
        c.tab[c.nb]=f;  //  marquer les deux etapes
        c.nb = c.nb+1;
}

void bilan(catalogue c){ // ou (const catalogue &c)
        int n; float m, sum = 0; fourniture f;
        for (int i=0; i < c.nb; i++){
                f = c.tab[i];
                stock(f,n,m);
                cout << "il y a " << n << " "<< f.libelle
                 << " en euro, ca represente " << m << endl;
                sum += m;
        }
        cout << "le stock total vaut " << sum << endl;
}

void demoBilan(){
        catalogue c; c.nb=0;
```

```
        ajouteAuCatalogue(c, f1);
        ajouteAuCatalogue(c, f2);
        bilan(c);
}

void retireCatalogue(catalogue &c, int code){
   int i=0;
   while (i <c.nb and c.tab[i].code != code) {i++;}
   if (i==c.nb) {
     cout << "la fourniture de code " << code << "n existe pas ";
   } else { // on supprime la i-ieme donc on decale le reste du tableau
            for(int j=i; j<c.nb; j++) c.tab[j] = c.tab[j+1];
            c.nb = c.nb -1;
   }
}

void demoRetire(){
        catalogue c; c.nb=0;
        ajouteAuCatalogue(c, f1);
        ajouteAuCatalogue(c, f2);
        bilan(c);
        retireCatalogue(c, 1024);
        bilan(c);
}

int main(){demoAffiche();        demoMaj() ;
        demoStock() ;
        demoRetire();
        return 0;
}
```