

INF1005C – PROGRAMMATION PROCÉDURALE

Travail dirigé no. 5

Enregistrements et fichiers binaires

Objectif : Permettre à l'étudiant de manipuler des fichiers binaires et des enregistrements, ainsi que de maîtriser les concepts d'accès direct et séquentiel.

Durée : Deux séances de laboratoire.

Remise du travail : Avant 23h30 le jeudi 20 novembre 2014.

Travail préparatoire : Lecture des exercices et de la documentation fournie et rédaction des algorithmes.

Directives : N'oubliez pas les entêtes de fichiers ni, aux endroits appropriés, les commentaires dans le code. Vous devez ajouter un en-tête pour chaque fonction. Dans l'écriture de l'en-tête d'une fonction, ne pas oublier de donner la description IN/OUT pour chacun des paramètres. Il est interdit d'utiliser les variables globales, sauf celles en lecture seule (constantes).

Documents à remettre : Sur le site Moodle des travaux pratiques, vous remettrez l'ensemble des fichiers .cpp et .hpp compressés dans un fichier .zip en suivant la procédure de remise des TDs.

Mise en contexte

Vous faites partie de la société technique Smart Bird qui fait la conception, la fabrication et l'opération d'un drone (UAV) de reconnaissance à Polytechnique. Vous participez à une compétition aux États-Unis dans laquelle vous devez détecter des cibles au sol avec votre drone. Pour ce faire, votre avion survole de façon autonome la zone en prenant des photos avec la caméra embarquée. La détection des cibles à partir des images est faite par un logiciel de vision par ordinateur que vous avez aidé à développer.

Ce logiciel reçoit les images de la caméra et les analyse pour en extraire des cibles. Une cible est une feuille de bois d'une certaine forme géométrique et d'une certaine couleur avec une lettre d'une autre couleur écrite dessus. Pendant son fonctionnement, le logiciel sauvegarde les cibles qu'il a détectées dans un fichier binaire brut. Il peut ensuite les exporter dans un fichier texte de style CSV (character separated value) selon un format prescrit par les règles de la compétition. C'est le rapport final de détection remis aux évaluateurs.

On vous demande de compléter le module qui permet de produire ce rapport à partir des données brutes des cibles détectées par le logiciel. Il vous faut aussi écrire un petit programme qui interagit avec l'utilisateur.

Matériel fourni

Dans le fichier *CodeFourni.hpp* vous trouverez les déclarations des structures, constantes et variables qui vous sont fournies; leur implémentation se trouve dans *CodeFourni.cpp*. Vous devez écrire vos fonctions dans *CodeDemande.cpp* et mettre vos prototypes dans *CodeDemander.hpp*. Vous n'avez qu'à suivre les *TODO* dans chaque fonction, et cela veut aussi dire de déterminer le type approprié pour la valeur de retour ainsi que les paramètres. On vous donne aussi un fichier d'aide *Documentation.chm* qui contient toute la documentation du code fourni. Vous y retrouverez les descriptions détaillées des fonctions avec leur valeur de retour et paramètres ainsi que la description de ce que représentent les structures. Si jamais vous vous demandiez à quoi ça sert d'écrire des entêtes de fonctions, et bien en voici un très bon exemple!

On vous donne aussi un fichier texte *RapportFinalPartiel.txt* qui contient la moitié des cibles, dont certaines incomplètes, ainsi qu'un fichier binaire *Cibles.data* qui contient l'autre moitié des cibles.

Il y a deux structures d'enregistrement avec lesquelles vous devez travailler. L'enregistrement *Cible* représente une cible détectée qui possède différentes caractéristiques telles qu'un numéro unique, une position, une forme, etc. Deux cibles sont comparées à l'aide de leur numéro unique (*Cible::id*). La structure *ListeCibles* est une liste de plusieurs cibles avec une capacité (taille maximale) donnée par *MAX_NB_CIBLES* et une taille (nombre d'éléments présents) donnée par son membre *nbElems*. Consultez la documentation pour plus de détails sur les deux structures, les constantes et les fonctions qui leur sont associées.

```
struct Cible
{
    uint32_t    id;
    double      latitude;
    double      longitude;
    double      orientation;
    IndexForme  forme;
    IndexCouleur couleurForme;
    char        lettres[4];
    IndexCouleur couleurLettres;
    char        nomImage[TAILLE_NOM_IMAGE];
};
```

```
struct ListeCibles
{
    Cible  elems[MAX_NB_CIBLES];
    size_t nbElems;
};
```

Note : Le type `uint32_t` est un entier non-signé à 32 bits et le type `size_t` est un entier non-signé souvent utilisé pour représenter des tailles.

Travail à réaliser

Tout d'abord, lisez bien la documentation des fonctions fournies, car vous aurez à vous servir de quelques unes d'entre elles. On vous fournit entre autres des fonctions qui font la conversion d'une cible vers une string formatée selon le format strict de l'exercice et *vice-versa*. Les fonctions les plus importantes à comprendre sont `formaterCible()`, `convertirEnCible()`, `indexerForme()`, `indexerCouleur()`, et `trierListeCibles()`. Aussi, jetez un coup d'œil au fichier *RapportFinalPartiel.txt* pour voir ce que donne le formatage des cibles.

Vous avez six fonctions à implémenter. On vous donne les squelettes dans *CodeDemande.cpp* et vous devez ajouter les prototypes dans *CodeDemande.hpp*. Pour chaque fonction, vous devez choisir le bon type de valeur de retour et les bons paramètres selon ce qui est demandé, puis remplir les *TODO* qui vous sont donnés dans le code. Vous pouvez écrire plus de fonctions si vous voulez, mais ce n'est pas nécessaire. Vous devez ensuite écrire le `main` en suivant les directives présentes dans le squelette (les *TODO*) et plus loin dans l'énoncé.

Fonction `insérerCible()`

Prend en paramètre une liste de cibles et une cible.

Insère une cible dans une liste. Si la cible est déjà présente dans la liste (les cibles sont comparées à l'aide de leur ID), elle est remplacée. Si la cible n'est pas présente dans la liste et qu'il reste de la place dans la liste (constante `MAX_NB_CIBLES`), alors elle est insérée à la fin, puis la liste est triée avec la fonction fournie pour faire cela.

N.B. : À plusieurs endroits, on vous demande «d'insérer une cible dans une liste», et on vous a aussi demandé d'écrire une fonction «insérerCible». Coïncidence?

Fonction `importerCibles()`

Prend en paramètre une liste de cibles et un nom de fichier.

Importe les cibles formatées présentes dans un fichier texte et les insère dans une liste.

N'oubliez pas que la fonction `convertirEnCible()` permet d'extraire une cible (type `Cible`) à partir d'une ligne formatée selon le format de la compétition.

Fonction `chargerToutesCibles()`

Prend en paramètre une liste de cibles et un nom de fichier.

Charge les cibles brutes présentes dans un fichier binaire et les insère dans une liste.

Les cibles dans le fichier binaire ont exactement la même représentation que la structure `Cible` dans la mémoire du programme.

Fonction `sauvegarderCible()`

Prend en paramètre une cible et un nom de fichier.

Sauvegarde une cible dans un fichier binaire. Si la cible n'est pas déjà dans le fichier, elle doit être insérée à la bonne place, c'est-à-dire selon son numéro unique et en ordre croissant de numéro. Pour ce faire, les cibles suivantes sont décalées pour faire de la place à la cible à insérer, puis la cible est écrite dans l'espace libéré.

Vous devez seulement décaler les cibles qui se trouvent après le point d'insertion. Ne réécrivez pas le fichier au complet.

Fonction `produireRapport()`

Prend en paramètre une liste de cibles et un nom de fichier.

Exporte les cibles dans un rapport texte formaté selon les règles de la compétition. Chaque cible est sur une ligne.

On vous fournit la fonction qui traduit une cible en une string formatée.

Fonction demanderMiseAJourCible()

Prend en paramètre une cible.

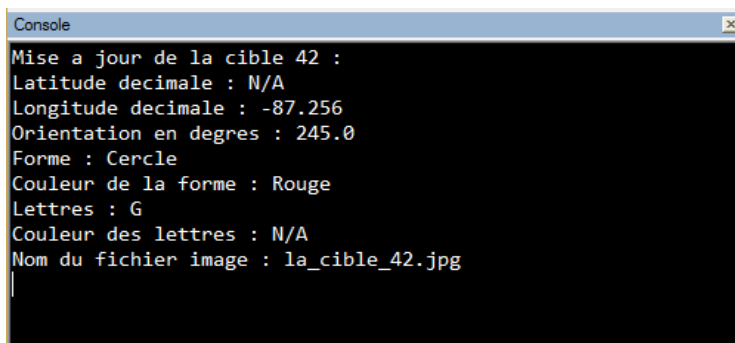
Affiche le numéro de la cible donnée et demande à l'utilisateur d'entrer toutes les autres valeurs. Si l'utilisateur entre la même chose que la valeur de `DONNEE_INCONNUE`, alors la valeur est ignorée, c'est-à-dire qu'elle n'est pas changée dans la cible.

`DONNEE_INCONNUE` est une string constante égale à "N/A" déclarée dans *CodeFourni.hpp*. Vous devez donc lire toutes les entrées de l'utilisateur comme des strings, même les données numériques, pour être capable de faire la comparaison avec `DONNEE_INCONNUE`.

On vous donne les fonction `indexerForme()` et `indexerCouleur()` pour trouver la valeur d'index d'une string représentant une forme ou une couleur. La string "Rouge" donnerait donc la valeur numérique `ROUGE` de l'énumération `IndexCouleur`. La string "Hexagone" donnerait la valeur numérique `FORME_INCONNUE` de l'énumération `IndexForme`, car l'hexagone n'est pas une des formes reconnues dans l'énumération `IndexForme` et dans le tableau `NOMS_FORMES`.

N.B. : La fonction `std::stoi()` présente dans `<string>` permet d'extraire la valeur de double représentée par une `std::string`. Il faut aussi utiliser `strcpy` ou `strncpy` dans `<cstring>` pour faire la copie des tableaux de caractères. Regardez dans vos notes de cours ou sur cplusplus.com pour voir comment s'en servir.

L'exécution de cette fonction devrait ressembler à ceci :



```
Console
Mise a jour de la cible 42 :
Latitude decimale : N/A
Longitude decimale : -87.256
Orientation en degres : 245.0
Forme : Cercle
Couleur de la forme : Rouge
Lettres : G
Couleur des lettres : N/A
Nom du fichier image : la_cible_42.jpg
|
```

Dans ce cas, la cible numéro 42 gardera sa latitude et la couleur de ses lettres, car l'utilisateur a entré N/A pour ces propriétés. Toutes les autres valeurs seront écrasées par ce que l'utilisateur a donné.

Fonction principale du programme

Le scénario (fictif) est le suivant : pendant la phase d'analyse après le vol, vous avez eu des problèmes avec le logiciel de détection et votre équipe se retrouve avec la moitié des cibles sauvegardée en binaire et l'autre moitié dans un rapport formaté. Les détecteurs du logiciel n'ont toutefois pas été capable d'extraire toutes les caractéristiques de certaines cibles. Il vous faut donc écrire un petit programme qui importe et charge les cibles déjà détectées et permet demande à l'utilisateur d'entrer manuellement les données manquantes.

On a déjà déclaré les noms de fichier que vous devez utilisés et la liste de cibles avec laquelle vous devez travailler. Les trois premiers *TODO* devraient être assez simples, vous n'avez qu'à utiliser les fonctions que vous avez écrites.

Le fichier *RapportFinalPartiel.txt* contient la moitié des cibles en format exporté et le fichier *Cibles.data* contient l'autre moitié des cibles. Il manque toutefois des informations pour les cibles 1 et 11. Dans le quatrième point, vous devez vous servir de votre fonction *demandeMisAJourCible()* pour entrer au clavier les informations manquantes. La mise à jour doit se faire dans des cibles temporaires, **PAS DIRECTEMENT DANS LA LISTE**.

Pour la cible 1 (`lesCibles.elems[0]`), il faut donner une orientation de `0.0` degrés et la lettre "H".

Pour la cible 11 (`lesCibles.elems[10]`), il faut donner une orientation de `45.0` degrés, la forme "Triangle" (la casse est importante) et la lettre "O".

Sauvegardez ces cibles temporaires dans le fichier binaire créé au troisième *TODO* et chargez-les dans la liste à partir de ce fichier binaire.

Enfin, exporter le tout dans le rapport final. Vous devriez obtenir 11 cibles complètes.

Vous pouvez comparer votre produit avec le fichier *RapportFinalCorrige.txt* pour savoir si vous faites la bonne chose. Ne portez pas attention au dernier chiffre significatif des nombres décimaux, il se peut qu'il soit différent, les nombres à points flottants étant ce qu'ils sont.

Les lettres des cibles forment un anagramme. Dans le cas de celles du TD (tirées de la compétition de 2013), le message secret était «FEAR THE GOAT».

Références

Facebook – Smart Bird Polytechnique

<https://www.facebook.com/smartbirdpolymtl>

Règles de la compétition

http://www.auvsi-seafarer.org/documents/2014Documents/2014_AUVSI_SUAS_Rules_Rev_1.0_FINAL_13_1024_1.pdf

Voir l'appendix E, page 53, pour le format du fichier texte.