TP STL

(2,5 scéances)

Yannick Kergosien

Objectif du TP : *se familiariser avec les structures et fonctions de bases de la STL.*
Lien utile : `http://www.cplusplus.com/reference/stl/`.

1 Manipulation d'un *vector* de *int*

Ecrivez un programme qui :
— Commence par créer un vecteur d'entier vide.
— Permet à l'utilisateur de saisir 5 entiers ($cin >> maVariableInt$;).
— Affiche dans l'ordre de saisie les 5 entiers grâce à l'opérateur [ ].
— Affiche dans l'ordre de saisie les 5 entiers grâce aux itérateurs.
— Demande à l'utilisateur un entier A compris entre 0 et 4 ; et supprime la A'ième case du vecteur.
— Affiche les 4 entiers du vecteur.
— Demande à l'utilisateur un entier A compris entre 0 et 3 et un autre entier B ; et insère à la A'ième case du vecteur l'entier B.
— Affiche les 5 entiers du vecteur.
— Et enfin vide le vecteur.

2 Manipulation d'un *set* de structure

Ecrivez un programme qui :
— Crée une structure "S_Coord" qui contient deux *double* "d_X" et "d_Y".
— Ajoute dans un ensemble *set* une dizaine de coordonnées. Cependant, pour ajouter des structures dans ce conteneur, il faut lui indiquer comment comparer deux structures "S_Coord" :

```
bool operator<(const S_Coord & s1, const S_Coord & s2)
{
   return s1.d_X < s2.d_X;
}
```

— Affiche toutes les coordonnées.
— Crée un tableau à deux dimensions (vecteur de vecteur de double).
— Calcul toutes les distances Euclidiennes entre chaque point de l'ensemble et stocke les résultats dans le tableau à deux dimensions.
— Affiche la distance entre deux points demandés par l'utilisateur en saisissant deux entiers a et b (pour connaître la distance entre le a'ième point et le b'ième point).

3 Manipulation d'une *multimap* de *string*

Ecrivez un programme qui :
— Crée une multimap de *string* avec des clés de type *int*.
— Ajoute des clés et des chaînes de caractères (ces deux éléments étant saisis à chaque fois par l'utilisateur) jusqu'à la saisie du mot "end".
— Affiche tous les élements dans la map.
— Affiche la valeur associée à une clé saisie par l'utilisateur (ou si cette clé n'existe pas, affiche la valeur dont la clé est la plus proche et supérieure au nombre saisi) jusqu'à la saisie du nombre 0.

— Affiche la clé associée à une valeur saisie par l'utilisateur jusqu'à la saisie du mot "end".
— Supprime tous les éléments dont la valeur contient une des lettres que saisis l'utilisateur (ex : si l'utilisateur saisi "azer", toutes les paires dont la valeur contient un a ou z ou e ou r seront surpprimées).

4 Un début du projet...

Ecrivez un programme dont vous reprendrez certaines parties du code pour faire le projet, qui :
— Crée un vecteur de class client, cette classe contient les attributs suivants :
  — Des *string* : un nom, un prénom, une adresse, une ville et des commentaires sur sa maladie.
  — Des *int* : un numéro de téléphone, un code postale, une durée estimée de consultation et une priorité.
  — Un vecteur de *int* qui représentera des identifiants de ressource dont aura besoin le client.
— Ecrit grâce à la fonction *for_each* vers un flux donné en paramètre, le nom et le prénom, et permet de connaître (grâce au retour de *for_each*) la priorité moyenne et la somme totale des durées de consultation.
— Effectue un pré-traitement grâce à la fonction *transform* : trie dans l'ordre croissant l'attribut vecteur.

5 Un algorithme simple à implémenter grâce à la STL

L'objectif de l'algorithme est de trouver une heure de rendez-vous pour une consultation pour tous les clients contenus dans le vecteur précédent. Afin de simplifier la planification des rendez-vous, nous considérerons que :
— les clients sont tous à traiter le même jour à partir de 8h.
— le nombre de ressources (assureur, banquier, etc.) est connu à l'avance (données en entrée).
— les ressources sont disponibles toute la journée à partir de 8h.
— une ressource ne traite qu'un seul client à la fois.
— les rendez-vous peuvent être pris qu'au quart d'heure (Xh00 ; Xh15 ; Xh30 ; Xh45), le pas de temps sera donc de 15 minutes.
— un client qui nécessite plusieurs ressources doit passer un "examen" sur chacune des ressources dans l'ordre du vecteur des ressources de la classe Client.
— la durée estimée de consultation représente la durée total pour tous les examens, nous supposerons que la durée d'un examen est égale à la durée total divisée par le nombre de ressources nécessaires (par exemple, un client dont la durée de consultation est de 60 minutes et nécessite 3 ressources, passera 20 minutes par examen/ressource).

L'idée de l'algorithme, un simple glouton, est de :

1. Sélectionner les clients un à un par ordre de priorité puis du plus contraignant au moins contraignant en fonction du nombre de ressources nécessaires et de la durée estimée de consultation. Vous pouvez utiliser un indice de priorité égale à [la priorité * 100 + le nombre de ressources nécessaires * 10 + la durée totale nécessaire pour la consultation] pour trier vos clients. Il faudra donc traiter ces clients par ordre décroissant de cette liste.
2. Affecter les clients un à un aux ressources :
   — En suivant la séquence des ressources du client, l'heure de début de chaque examen d'un client est choisi au plus tôt : maximum entre l'heure de fin du précédent examen (ou 8h) et la prochaine heure où la ressource est disponible sur toute la durée nécessaire.
   — Les affectations doivent se faire au quart d'heure.

A vous de trouver quels conteneurs seraient pratiques pour résoudre simplement ce problème.

Voici un exemple avec 2 ressources et 4 clients :

- Le premier nécessite un seul examen sur la ressource 2 pour une durée de 35 minutes. Il sera donc inséré au plus tôt à 8h (toutes les ressources étant libres).
- Le deuxième nécessite deux examens : sur la ressource 1 pour une durée de 26 minutes puis sur la ressource 2 pour une durée de 23 minutes. Le premier examen sera inséré à 8h et devra attendre la libération de la ressource 2 pour le deuxième examen (“arrondi” au quart d’heure).
- Le troisième nécessite deux examens : sur la ressource 1 pour une durée de 53 minutes puis sur la ressource 2 pour une durée de 27 minutes. Même chose que pour le client 2 cependant l’heure du deuxième examen est contrainte par la fin de son premier examen.
- Le quatrième nécessite un seul examen sur la ressource 2 pour une durée de 24 minutes. Cet examen aurait pu être inséré entre le client 2 et 3 sur la ressource 2 si il durerait moins que 15 minutes, mais comme ce n’est pas le cas il sera inséré à la fin.

Vous retrouverez donc les Gantts suivants :