Techniques d'implémentations

Benoit Pralong

28 septembre 2006

Le but de la session :

- Avoir quelques bases d'utilisation de LINUX.
- Savoir ce qu'est la STL : comment ça marche et comment l'utiliser.
- Faire un Makefile.
- Avoir quelques notions sur la génération d'une documentation.

Présentation de système d'exploitation de type LINUX

Système d'exploitation

On a Trois types de système d'explotations classiques :

- Windows (98, 2000, XP, Vista ...)
- Mac OS
- LINUX (Plusieurs distributions : Fedora, Debian, Suze, etc...)

LINUX : Un peu d'histoire

- GNU en 1985 (GNU is Not UNIX, GNU is Not UNIX is Not UNIX) Premiers logiciels libres, mais pas encore d'OS.
- LINUX, un projet de Linus Torval.
- Bill Clinton : utilisation de systèmes d'exploitations linux moins onéreux.
- Diffusion + Logiciel Libre (quelques exemples de logiciels libre connus ?)

FEDORA

Fedora est la version gratuite de Red Hat. Ici, à l'École Polytechnique la majorité des ordinateurs sous environnement LINUX utilisent une Fedora.

`http://fedoraproject.org/wiki/`

Terminal : la console

- Taper les commandes : donner les ordres à l'ordinateur
- Dernier recours en cas de bug complet de l'OS : permet de tout réparer (seulement quand on a de la pratique, sinon on risque d'aggraver son cas)

Les éditeurs de texte

- vi : éditeur de texte directement en ligne de commande : aucune érgonomie, mais fonctionne en toutes occasions.
- emacs : un des plus vieux, les premières versions datent du projet GNU.
- nedit, gedit, edit : un classique.
- kate : mon préféré, necessite KDE.

Commandes de bases : cd

cd : on se déplace dans l'arborescence.

- cd REP : on se déplace vers le répertoire REP
- cd .. : on recule vers le répertoire précedent.
- cd ou cd ~ : on va vers son répertoire HOME.

Commandes de bases : ls, pwd, rm, more et cat

- pwd : nous indique où nous sommes dans l'arborescence du projet (répertoire courant).
- ls : on liste les fichiers du répertoire courant.
- rm : efface un fichier.
- more : more fic. Affiche le contenu du fichier fic.
- cat : cat fic. Affiche le contenu du fichier fic.

Commandes de bases : mkdir, mv, cp, touch et ps

- mkdir : mkdir REP : crée un répertoire s'intitulant REP à partir du répertoire courant.
- mv : mv origine destination : déplace le fichier orgine vers le repertoire destination.
- cp : cp origine destination : copie le fichier origine vers la destination
- touch : touch fic : crée le fichier fic
- ps : affiche les processus ayant été lancé de la console. ps -aux affiche tout les processus lancés

Commandes de bases : kill, man, help, chmod, su et passwd

- kill -9 process : tue le processus ayant l'identifiant process
- man : man commande : affiche le manuel de la commande
- help : help commande : affiche l'aide (manuel) de la commande
- chmod : change les autorisations d'un fichier, répertoire, executable
- su : super utilisateur
- passwd : modifie son mot de passe

Commandes de bases : ssh et sftp

- ssh : permet de se connecter sur une machine distante.
- sftp : permet de se connecter sur une machine distante et de pouvoir transferer des fichiers. (put et get (mput et mget))

Pour en savoir plus

http://www.commentcamarche.net/linux/lincomm.php3
http://www.linux-france.org/article/debutant/debutant-linux.html
http://clx.anet.fr/spip/article.php3?id_article=104

Présentation de la STL

Introduction

- Motivation
  - Outils puissant : les Templates
  - Besoin d'avoir une bibliothèque puissante qui gère plusieurs SDD et algorithmes classiques sans avoir à les réimplementer chaque fois que l'on change de SDD.
- Histoire
  - 1979 Alexander Stepanov commence à mettre en forme ses idées de programmation générique
  - 1987 Bibliothèque pour le langage ADA
  - 1994 HP rend disponible gratuitement sa propre implémentation de la STL
  - Auhourd'hui, sert de base à la plus part des distributions STL

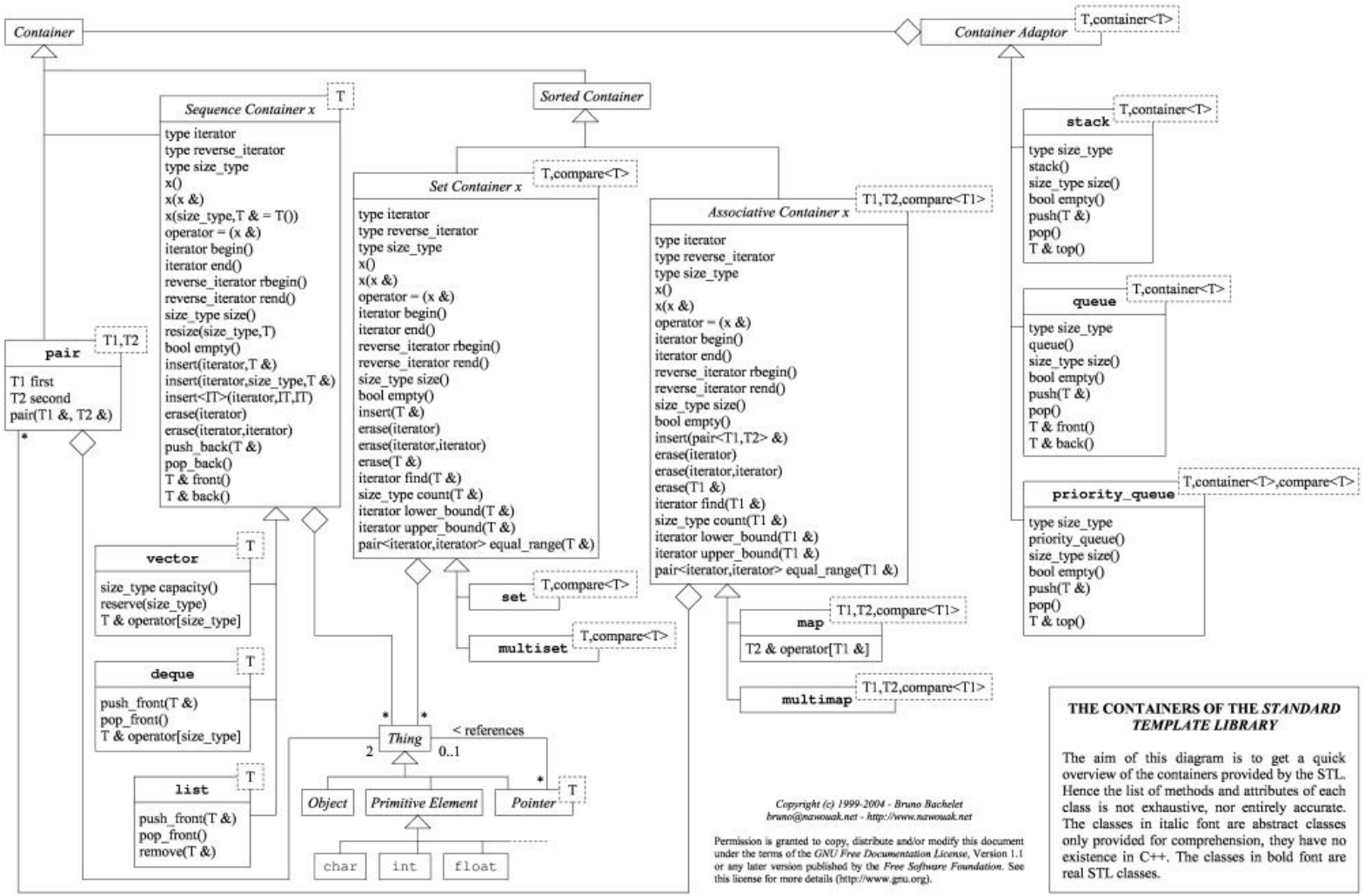
Programmation Générique et Template

- Template : permet de créer des fonctions indépendantes du type de données utilisées (exemple de min et max)
- Programmation Générique : indépendance vis à vis du type de données (utilisation des Templates), polymorphisme, signature des fonctions.

Contenu de la STL

- Ensemble de classes conteneurs
- Itérateurs
- Algorithme Générique
- Classe String

Diagramme UML de la STL



Vector - Présentation

- Tableau de valeurs : on peut accéder à n'importe quelle valeur en temps constant.
- Possède la capacité de modifier sa taille quand on insère ou enlève un élément.
- Insérer ou enlever un élément à la fin : temps constant par rapport à la taille du vecteur
- Insérer ou enlever un élément au début ou au milieu : temps linéaire par rapport à la taille du vecteur

List - Présentation

La classe list modélise la liste chainée normale. Elle ne possède pas d'opérateur d'indexation.
Ajout ou suppression d'un élément en temps constant.

dequeu - Présentation

- intermediaire entre list et vector
- possède un opérateur d'indexation de complexité o(log(n)) amortie.
- insertion en milieu et fin de séquence plus rapide que sur un vector sans atteindre la performace d'une liste

Opérateurs élémentaires

- size : renvoie le nombre d'éléments dans le conteneur
- max_size : renvoie le nombre maximum d'éléments
- empty : renvoir true si le conteneur est vide
- resize : redimensionne le conteneur

Opérateurs élémentaires

- front : premier élément
- back : dernièr élément
- push_back : ajoute un élément au fond de l'objet
- pop_back : retire un élément au fond de l'objet

Opérateurs élémentaires

- insert : insere un élément (ou plusieurs éléments)
- erase : on supprime un élément
- swap : on échange plusieurs valeurs entres elles

Autres conteneurs

- queues : files : premier arrivé, premier sorti (FIFO)
- stack : piles : dernier arrivé, premier sorti (LIFO)
- pair : paire : très utile
- conteneurs associatifs (clefs, valeurs)
  - map : clé - objet
  - set : objet - objet

Les itérateurs

- agissent comme des pointeurs sur les objets de la STL.
- Chaque structure de la STL définit son type d'itérateur.
- Chaque itérateur a des propriétés particulière.

Interêt et utilisation des itérateurs

Les itérateurs représentent la manière la plus sure de se promener dans les éléments d'un objet STL (list, vector, etc etc ...).
La plus part des algorithmes présent dans la STL nécessite des itérateurs (par ex, les algorithmes d'insertions).

Fonction classique présente pour les itérateurs

- begin() : renvoie au premier élément du conteneur
- end() : renvoie à la case après le dernier élément du conteneur (concept dépendant du conteneur)
- ++ : permet de se déplacer dans le conteneur d'un élément à son suivant
- -- : permet de se déplacer dans le conteneur d'un élément à son prédecesseur. Attention, pas toujours utilisable (depend du conteneur)

Mais...

La grande utilitée des itérateurs réside dans les algorithmes :

- de recherche
- de tri
- d'insertion
- de suppression

Certains algorithmes sont déjà présent dans la STL (Tri par exemple). Ils sont codés selon le type de structure de données, il dépendront de ce fait du type d'objet auxquel ils font face.
Il y a plus de 70 algorithmes différents.

Les algorithmes classiques

- accumulate(debut, fin, init) : renvoie la somme de tout les élements de la séquence, init=0 pour entier, 0.0 pour réels
- find(debut, fin, v) : renvoie un iterateur pointant sur la valeur v entre debut et fin.
- count(debut, fin, v, 0) : compte le nombre de fois que la valeur v apparait entre debut et fin
- for_each(begin, end, F) : applique F pour tout les éléments de la séquence.

Les algorithmes classiques

- max_element(begin, end) : renvoi un itérateur sur la valeur max
- min_element(begin, end) : renvoi un itérateur sur la valeur min
- replace(begin, end, old, new) : dans la séquence remplace chaque valeur de old par new
- reverse(begin, end) : renverse la séquence

Les algorithmes classiques

- sort(begin, end) : trie en ordre ascendant (du plus petit au plus grand)
- unique(begin, end) : remplace chaque valeurs consécutives de même valeur par une seule instance.

la classe String

Permet de gérer les chaînes de caractère de manière plus simple que les tableaux de char. Offre des opérateurs simple comme +, -.
Permet d'obtenir le i-ème membre d'une chaine de caractère par s[i].
Sans rentrer plus dans les détails, la classe string offre un outil facile d'utilisation pour gérer les chaînes de caractère.

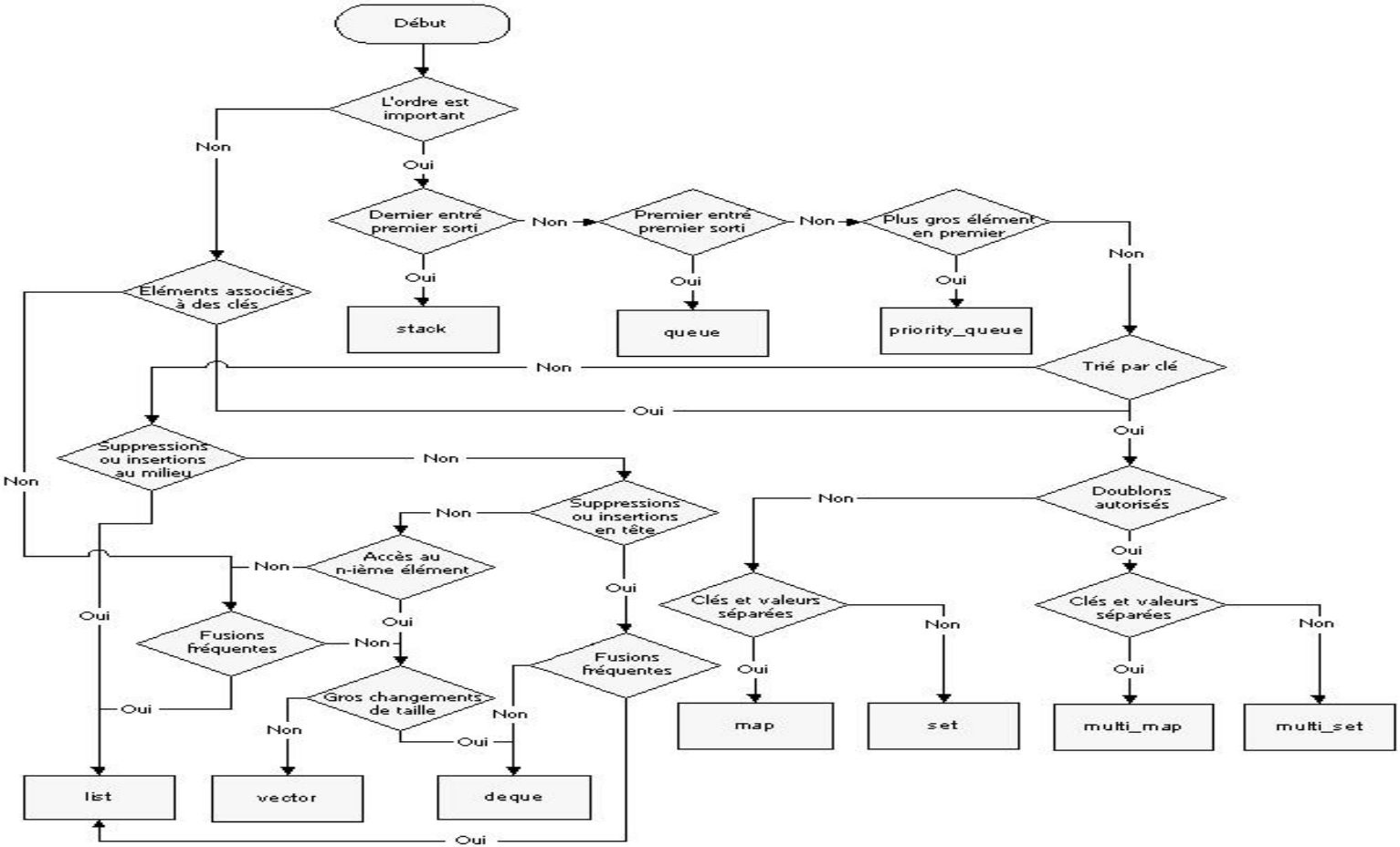
N'oubliez pas

Internet offre de nombreux tutoriaux :

- `http://www.msoe.edu/eecs/ce/courseinfo/stl/`
- `http://www.infosys.tuwien.ac.at/Research/Component/tutorial/prwmain.htm`

La littérature offre de nombreuse aide : il y a au moins un chapitre sur la STL dans n'importe quel livre de C++.
Utiliser soit même la STL dans ces projets, c'est en forgeant que l'on devient forgeron.
Savoir que ce que l'on cherche existe, cela vous fera gagner beaucoup de temps.

De quoi ai-je besoin ?



Comment faire un Makefile

Un fichier ?

- fichier qui contient plusieurs instructions de compilation, pour un projet donné.
- information sur le compilateur, les répèrtoire ou sont les fichiers, les différents flags de compilation
- liste une série d'ordres, indique quand les éxecuter

où un programme ?

- Quand on appelle make, on execute le makefile
- la commande *make* ordre execute un ordre particulier préalablement décrit dans le Makefile

Un exemple de Makefile

```
projet.exe  : arbre.o main.o
              g++ main.o arbre.o -o projet.exe
arbre.o     : arbre.h arbre.cpp
              g++ arbre.cpp -c
main.o      : main.cpp
              g++ main.cpp -c
clean       :
              rm -f *.o
              rm -f *
              rm -f *.exe
```

Les règles de compilation

Dans un Makefile, les règles de compilations s'écrivent de la manière suivante :

cible : dépendances
commandes

Si les dépendances renvoient à d'autres cibes, alors, on évaluera les autres cibles.

Et si on rajoutait des variables ?

On peut rajouter des variables, de manières à rendre le Makefile plus générique. Ainsi, si par exemple, on passe d'une architecture à une autre, il sera plus simple de juste modifier une variable du Makefile, que chaque occurence de cette variable dans le fichier Makefile.

Les variables

Une variable se définit NOM=VALEUR et se voir utiliser via $(NOM) dans le Makefile.

```
CC=g++
CFLAGS=-Wall -ansi-pedantic
LDFLAGS=-Wall -ansi
EXEC=tp1
```

Variables internes au Makefile

$@ : Le nom de la cible
$< : Le nom de la première dépendance
$^ : La liste des dépendances
$? : La liste des dépendances plus récentes que la cible
$* : Le nom du fichier sans suffixe

Règles d'inférences

On peut par exemples créer les .o à partir des .c. Pour cela, une solution possible, utiliser une règles d'inférences.

```
%.o  : %.c
    commandes
```

Ce qui peut nous donner :

```
%.o  : %.c
    $(CC) -o $@ -c $< $(CFLAGS)
```

Pour en savoir plus

taper man make en ligne de commande sous linux.
`http://ftp.traduc.org/doc-vf/gazette-linux/html/2002/083/lg83-B.html`
Il existe aussi de nombreux autre liens sur internet.

Quelques notions sur Doxygen

Présentation de Doxygen

- logiciel permettant de créer des documentations pour du code informatique.
- s'inspire beaucoup de javadoc.
- prend en compte la grammaire du langage utilisée
- prend en compte les commentaires (écrit dans un format reconnu par Doxygen).

Un exemple

```
/**
* La classe A représente une classe quelconque.
* @author Benoit Pralong
*/
class A {
/**
* Constructeur.
* Fixe l'objet A à une valeur précise .
* @param val valeur que l'on donne à A
*/
A(int val) {
...
}
}
```

Exemple de Tags

- @author
- @param
- @return
- @version

Commentaires Doxygen reconnu

-
  - /**
  - * commentaires
  - */
-
  - /*!
  - * commentaires
  - */
-
  - /**
  - commentaires
  - */

Commentaires Doxygen reconnu

-
  - ///
  - /// commentaires
  - ///
-
  - //!
  - //! commentaires
  - //!
-
  - //////////////////////
  - /// commentaires
  - //////////////////////

- site officiel : `http://www.stack.nl/~dimitri/doxygen/` ou `http://www.doxygen.org`
- exemple : `http://dbus.freedesktop.org/doc/api/html/index.html`
- un tutoriel : un exemple `http://www.stack.nl/~dimitri/doxygen/docblocks.html`