

PROG 2

L3 RI

TD 1 : Introduction à C++

Surcharge des opérateurs : un même nom de fonction en désigne plusieurs selon le contexte (type des arguments par exemple).

Fonction amie qui accède aux champs **private** d'une autre classe.

static : lié à la classe et non à ses instances.

STL : Standard Template Library.

Itérateurs pour parcourir les différents conteneurs.

TD 2 : Introduction à la programmation C

Premières versions du système Unix.

Noyau pour d'autres langages (C++, Java...).

Utilisation de **gcc**.

sizeof pour connaître la taille d'un type.

Utilisation de **valgrind**;

Tous les paramètres sont passés par valeur (pas les tableaux : adresse du premier élément. Sauf dans une structure).

Fonction **ulimit** en Shell pour ajuster taille de la pile.

static : durée de vie globale, portée locale.

TD 3 : références et héritage en C++

Références : utilisation implicite. C'est un alias, on peut passer en paramètre et modifier l'objet lui-même.

Mot-clé **const** sur une référence si on ne veut pas que l'objet soit modifié.

Attention à ne pas retourner des références vers des objets supprimés.

Dans une classe : **public**, **private** pour ce qui est accessible depuis l'extérieur de la classe ou non. On peut déclarer des classes/méthodes amies avec **friend** qui ont accès au **private**.

On module la visibilité en faisant des héritages **public**, **private**, **protected**.

En cas d'héritage multiple, c'est la première occurrence de l'attribut qui masque les autres.

TD 4 : pointeurs, virtualisation et exceptions en C++

this : pointeur vers l'instance courante d'une classe.

Déconseillé d'utiliser pointeurs. Sont utilisés par STL. Ou dès qu'on utilise du C (**cstdlib**, **cstdio**...).

Copier une classe dont un attribut est un pointeur : on va copier la même adresse. Copie superficielle.

Résolution : choix de la méthode à appliquer (surcharge). Résolution statique : le compilateur cherche la première méthode possible dans l'héritage.

On peut spécifier la fonction pour utiliser la classe réelle de l'objet et non la classe spécifiée par la référence ou le pointeur : **virtual**.

Exceptions : **try**, **throw**, **catch**. Classes peuvent être lancées.

TD 5 : généricité en C++, conteneurs STL

template : déclarations paramétrées. Classes paramétrées : classes génériques.
Conteneurs associatifs : permet d'associer des valeurs à des clés. **mmap**. On peut ensuite faire des recherches de clés : **find**.
Différents types d'itérateurs pour les parcourir : **iterator**, **reverse_iterator**, **const_iterator**.
multimap : plusieurs paires pour une même clé. **set** et **multiset**.

TD 6 : Premiers pas en OpenGL/GLUT

L'interface est autonome par rapport au programme. Doit se reconfigurer à chaque événement : interactions clavier, X11, redimensionnement... Callback functions.

Quand on fait le programme, on n'a pas de contrôle sur l'environnement dans lequel il va être lancé. Les callback functions ne doivent dépendre que de choses sous contrôle.

TD 7 : À la découverte de Lisp

Un objet : soit un atome, soit une liste d'objets entre parenthèses.
Read Eval Print : REP.
Lier un symbole à une lambda-expression : déclaration de fonction.
cond : comme un **switch**. (**cond** (cond1 return1) (cond2 return2)...).
Liaison dynamique : récursivité native.
Les symbols dans une liste après un lambda ne seront évalués qu'au moment de l'application (liaison dynamique). On peut donc définir avec des objets encore non définis.
Protéger les noms de variables libres : **gensym**.

TD 8 : Évaluation statique et clôtures

Liaison lexicale : on fait un *snapshot* de l'environnement à chaque déclaration, que l'on restaure pour chaque évaluation.
Associer environnement et objet : clôture.
Il faut alors utiliser des *placeholders* pour les définitions récursives.
Continuation : doit utiliser la valeur des variables libres au moment de sa définition.

TD 9 : Scheme, aspects avancés

let équivalent à **lambda** (liaison locale pour évaluation).
letrec crée les liaisons avant évaluations : pour fonctions récursives.
define au toplevel pour créer variable.
set! : modification impérative d'une variable.
Environnement : liste chaînée de *frames* (tables d'association noms/valeurs).
Utilise lambda-expression : clôture.
call/cc call with current continuation. Permet de faire des coroutines entrelacées, qui peuvent se suspendre (**yield**), puis repartir.
delay et **force** on renvoie une clôture qui pourra être évaluée plus tard. Évaluation paresseuse.