Algorithmique et C++

Déclarations C++ et Exceptions

Jean-Pierre Fournier

Les déclarations en C

- permettent de spécifier les types des variables (dans le corps) et paramètres (dans l'entête)
- int unInt; *pour déclarer une variable simple de type* `int`*...*
- int fct(int param, short * tab); *pour déclarer une fonction retournant un* `int`*, et deux paramètres, un* `int` *et un pointeur sur un* `short` *(ou sur plusieurs* `shorts` *consécutifs en mémoire)*

Les déclarations en C++

- Objet unObjet; *pour déclarer une instance de la classe Objet*
- Objet * unPointeur; *pour déclarer un pointeur vers un objet de la classe Objet*
- Objet & uneReference=…; *pour déclarer une nouvelle référence d'objet de la classe Objet.*
- Les références simplifient l'usage des objets anonymes :
  - `uneListe.ancre->suivante->valeur=3;`
    `uneListe.ancre->suivante->suivante = NULL;`
  - `Cellule & laCellule=*(uneListe.ancre->suivante);`
  - `laCellule.valeur = 3; laCellule.suivante = NULL;`

Les références en C++

- permettent de récupérer sans le copier et sans passer par un pointeur l'objet retourné par une fonction, puis de travailler dessus,
  - `Objet & leResultat=recherche(…, …, …);`
- permettent de spécifier le type d'objet retourné par une fonction qui n'a pas besoin de copier son résultat,
  - `Objet & recherche(Objet *, int);`
  - `const Objet & recherche(const Objet *, int);`
- permettent de déclarer le paramètre qui recevra un objet envoyé lors de l'appel de sous-programme, sans que l'objet ne soit copié au passage.
  - boolean presence(const Objet &, int);

Les paramètres en C++

- Un paramètre de sous-programme C++ peut être :
  - en entrée (donnée)
  - en entrée et en sortie (donnée et résultat)
  - en sortie (résultat)
- Si c'est un résultat, le sous-programme ne doit pas travailler sur une copie => il doit recevoir la référence de l'original ou un pointeur vers l'original
  - void ssp(Objet * adresse);
  - void ssp(Objet & unObjet);

plus simple à utiliser, plus lisible…

Les paramètres en C++ (suite)

- Si c'est une donnée, il peut travailler sur une copie (coûteux) ou directement sur l'original, soit par référence, soit par pointeur, mais l'original **doit** être préservé :
  - void ssp(const Objet & unObjet);
  - void ssp(const Objet * uneAdresse);

plus simple à utiliser, plus lisible…

Déclarations et constructeurs

- Il est possible, au cours d'une déclaration, de transmettre une donnée au "constructeur" qui sera éventuellement employé
  - `Objet monObjet(23, "un texte");`
  - `Objet & unObjet=*new Objet(132);`
- Il est généralement interdit de déclarer une référence sans l'initialiser, et NULL n'est pas une valeur de référence acceptable…

Un algorithme

- s'appuie sur des pré-requis
  - un algorithme qui calcule une racine carrée suppose que la donnée est positive ou nulle
  - un algorithme qui cherche dans un tableau trié suppose que l'ordonnancement du tableau est juste
- doit aussi vérifier les post-requis
  - après une mise à jour de données, leur cohérence doit être totale

Que faire si le traitement demandé se révèle impossible ?

- Donner à l'un des résultats une valeur spéciale ?
  - Cela suppose qu'une telle valeur existe.
  - C'est reporter le problème à un autre algorithme.
  - C'est éventuellement obliger d'autres algorithmes à transmettre une valeur spéciale.
- Afficher un message ?
  - Cela suppose qu'un humain surveille le programme et sait quoi faire en cas de problème…
- Stopper l'exécution ?
  - C'est brutal.
  - Ce n'est pas "professionnel".

La gestion des exceptions

- Quand un algorithme est bloqué, il lève l'exception appropriée
  - if (. . .) ***throw*** …

Le fait que la condition soit vraie rend la suite impossible

On indique le nom d'une exception

La gestion des exceptions

- Un algorithme susceptible de ***propager une exception*** doit l'annoncer
  - cela fait partie de ses spécifications
    - void xxx(a, b, …) throw ***(x, y, …);***
    - Objet & yyy(a, b &, …) throw ***(x, y, …);***
- ***Une exception qui n'est pas propagée <u>doit</u> être traitée localement.***

La gestion des exceptions

- Pour un programme écrit en langage évolué, deux modes de déroulement :
  - Mode normal, les instructions s'enchaînent sans rupture
  - Mode exceptionnel, à la suite d'une levée d'exception :
    - Recherche du traitement approprié, localement d'abord, puis dans le bloc appelant…
    - Branchement direct dans la suite d'instructions prévue pour traiter l'exception, (généralement sans retour possible)

En C++

- pour lever une exception
  - throw (unObjet)
  - throw (15);
  - throw ("au secours");
- pour traiter une exception
  - try { *instructions susceptibles de lever des exceptions* } catch(*type et mode param.*){traitement};

En C++

try {*instruction; instruction; appel ssp propageant; instruction...*}

catch (Exception & exc) {*instruction; instruction ...*}

catch (int arg) {*instruction; instruction...*}

catch (…){*instruction; instruction…*};

- Les catch sont traités dans l'ordre !
- Ils peuvent aussi lever des exceptions !

Spécifications C++

- void ssp(int, const Obj &) throw(int);
  - Cette procédure ne pourra propager qu'une exception avec un **`int`**
- int fct(int, const Obj &) throw(int, char *);
  - Cette fonction ne pourra propager que ces deux types d'exceptions
- void Obj::mproc(int, const Obj &) const throw();
  - Cette méthode ne pourra pas propager d'exception.