2015-2016

L'3 – Structures de données (SDD)

TP n°1 – Listes chaînées

Enoncé

Il s'agit de développer et de tester les fonctions spécifiées dans le fichier modèle *tp1.cpp* se trouvant sur Campus. Ces fonctions se divisent en deux catégories : (i) parcours de listes, (ii) modifications de listes. Pour chaque fonction, la spécification précise s'il faut traiter une liste simplement ou doublement chaînée, s'il faut écrire la fonction de façon itérative ou récursive, etc.

Le langage C++ est utilisé pour simplifier l'affichage (opérateur <<), l'allocation mémoire (opérateurs *new* et *delete*). Les aspects orienté objet du langage ne sont pas utilisés dans le TP.

Le fichier *tp1.cpp* contient des fonctions utilitaires vous permettant de construire et d'afficher des listes tests : *listet()* et *chainel()*. Un exemple d'utilisation de ces fonctions est donné au début du *main*.

Fonctions

Le fichier *tp1.cpp* définit le type d'une liste et d'un maillon de liste. Il définit également le prototype de chaque fonction à développer, précédé d'une spécification précise dans un cartouche. Vous devez respecter à la lettre ces types, prototypes et spécifications car vos fonctions doivent prétendre à pouvoir être testées de façon automatique par un programme externe.

Vos fonctions doivent être écrites de façon robuste. Elles doivent notamment vérifier que leurs paramètres d'appel sont valides avant d'effectuer tout traitement.

Programme de test

Vous devez concevoir la suite des tests démontrant le bon fonctionnement de chacune de vos fonctions. Ces tests doivent être placés dans la fonction main du fichier *tp1.cpp*. Ils doivent s'exécuter de façon automatique (sans saisie utilisateur), être le plus exhaustif possible (cas général, cas particuliers) et produire une trace compréhensible (nom de la fonction testée, valeurs de ses paramètres et de son code retour).

Un exemple de test est donné en début du *main* pour la fonction *longueur()*.

Compilation

Le compilateur ne devra émettre aucun avertissement (warning), car sous ce terme peuvent se cacher de vraies erreurs potentielles.

Notation

Votre travail est susceptible d'être noté en séance.

Un peu d'aide pour la traduction des algorithmes en C/C++

Symboles spéciaux

Symbole du pseudo-code	Signification	Traduction en C/C++	Remarque
←	Affectation	=	N <- N+1 est syntaxiquement correct en C/C++ mais signifie « N est-il plus petit que –N+1 » !!!!!
=	Test d'égalité	==	if (a = b) … est syntaxiquement correct en C/C++ mais est équivalent à : a = b ; if (a != 0) … !!!!!

Données modifiées

Une *donnée modifiée* en pseudo-code se traduit en C/C++ par un paramètre passé *par référence*. Le code appelant fournit au sous-programme *un pointeur* sur une variable définie chez l'appelant. Le sous-programme peut ainsi modifier la valeur de cette variable, puisqu'il en connaît l'adresse : c'est le concept de *donnée modifiée*.

Ce mécanisme a une conséquence dans le sous-programme : le sous-programme doit systématiquement déréférencer le pointeur (par l'opérateur *) pour accéder en lecture ou en écriture à la donnée modifiée. Attention, la précédence de l'opérateur * est plus faible que celle de l'opérateur -> : il faut donc parenthéser.

En C++ uniquement, une *donnée modifiée* peut être définie simplement comme alias du paramètre effectif transmis par l'appelant.

Pseudo-code	Traduction en C/C++	Traduction en C++

monAlgo(l : liste)
Donnée modifiée : liste l à modifier
Début
…
l ← l→succ
…
Fin
	void monAlgo(liste *l)
	{
	// blindage (non obligatoire)
	if (l == NULL) {
	return ;
	}
	…
	*l = (*l)->succ ;
	…
	}
		void monAlgo(liste& l)
		{
		…
		l = l->succ ;
		…
		}

Allocation mémoire

Les fonctions d'allocation et de libération mémoire en C sont *malloc()* et *free()* (fonctions définies dans *stdlib.h*). Les opérateurs équivalents en C++ sont *new* et *delete*. Dans un même programme C++ on peut utiliser une ou l'autre des façons de faire, mais sans mixité (il ne faut pas libérer avec *delete* un bloc alloué avec *malloc()*, ni libérer par *free()* un bloc alloué par *new*).

Pseudo-code	Traduction en C/C++	Traduction en C++
p : adresse maillon	maillon* p ;	maillon* p ;
p ← réserver maillon	p = (maillon*) malloc(sizeof(maillon)) ;	p = new maillon ;
libérer p	free(p) ;	delete p ;