TD6 Listes chaînées, files et piles

L'objectif de ce sujet est de se familiariser avec la manipulation des structures en langage C. Celles-ci permettent en effet la manipulation de collections de données dont les types sont différents, contrairement aux tableaux. L'illustration se fera avec les listes chaînées, les files et les piles.

Introduction

Ce sujet introduit la notion de structures (précisons du langage C) qui sont très utiles pour regrouper des données de types différents. C'est une notion très utilisée en microprocesseurs et qui est aussi à l'origine de la notion d'objet dans la programmation. L'illustration sera faite avec des *structures de données linéaires* (le mot structure ici ne désigne absolument pas la structure du langage C, mais équivaut à "organisation") comme les listes chaînées, files et piles qui permettent un certain "rangement" des éléments tout en facilitant leur manipulation (recherche, insertion, suppression).

Préparation

1. Regarder les vidéos sur les structures de données. Celles-ci ont été faites sans Eclipse, mais vous pouvez à votre convenance utiliser Eclipse ou non.

— Fin de la préparation —

TD6.1 Structures en langage C

Les structures appartiennent à la famille des types composés. A l'inverse des tableaux qui contiennent obligatoirement des données de même type, les structures sont des collections de données qui peuvent être de types identiques ou différents. Les différentes données constituent les champs de la structure. En réalité, peu importe le type, l'important est de comprendre que chaque champ a un rôle important et unique à jouer au sein de la structure. La construction d'une structure se fait selon la syntaxe suivante :

```
struct nom_structure{
        type champ1 ;
        type champ2 ;
};
```

Tip : Ne pas oublier le ; après l'accolade de fermeture de la structure.

TODO 01 : Créer une structure `struct ninja`.	
Dans un fichier `structures.h`, définir une étiquette NBLETTRES égale à 20.	
Définir une structure avec le mot-clé `struct` appelée `ninja` comportant : — un champ appelé nom, de type tableau de char de taille NBLETTRES+1 — un champ appelé chakra, de type entier	
Utiliser un `typedef` pou définir un nouveau type. Le nom du type ainsi défini s'écrit généralement en MAJUSCULES.	`typedef struct ninja NINJA;`

La déclaration d'une structure se fait de façon identique à celle de n'importe quel autre type de données :

```
struct ninja nin1;
NINJA nin2;
```

L'initialisation de la structure peut se faire en même temps ou après la déclaration de la structure.

```
strcpy(nin1.nom, "Shikamaru") ; //ne pas oublier d'inclure <string.h>
nin1.chakra=1000 ;

NINJA nin2 = {"Choji", 2000} ;
```

Tip : Dans le cas d'une structure déclarée de façon statique (sans pointeur), le symbole "." permet d'accéder aux différents champs de la structure. Ce sera le cas très souvent en microprocesseurs. Dans le cas de structures de données, ce sera la notation pointeur "->" qui sera privilégiée.

Tip : Dans le cas d'une initialisation se déroulant en même temps que la déclaration, l'ordre des champs doit être respecté !

TODO 02 : Initialiser une structure `struct ninja`.	
Dans un fichier `main.c`, inclure `structures.h` et `stdio.h`.	
Créer une fonction principale.	Déclarer et initialiser deux structures `struct ninja` en utilisant les deux méthodes d'initialisation.
Afficher les champs sur la console.	
Sauvegarder, compiler, exécuter.	

TD6.2 Listes chaînées

Une liste chaînée est une collection d'éléments où chaque maillon est "relié" au suivant (éventuellement au précédent) par un pointeur. Pour faire ce tour de force, il faut au préalable avoir défini la notion de structure récursive.

TD6.2.1 Structure auto-référentielle ou récursive

Une structure récursive est une structure dont au moins un des champs est un pointeur vers une structure de même type.

```
struct nom_structure{
        type champ1 ;
        type champ2 ;
        struct nom_structure* champ3 ;
};
```

Tip : Attention ! il n'est pas possible d'utiliser le type défini en majuscules par le typedef dans la structure.

TODO 03 : Créer une structure récursive `struct ninja`.	
Modifier la structure NINJA en structure récursive en rajoutant deux pointeurs vers une structure NINJA nommés previous et next.	

La notion de pointeurs est fondamentale ici, car c'est elle qui permet de créer les listes chaînées. Cependant, la façon d'accéder aux champs d'une structure définie par un pointeur utilise un symbole "->" différent du ".".

```
NINJA* pnin ;
strcpy(pnin->nom,  "Sakura") ;
pnin->chakra=2000 ;
pnin->previous=NULL ;
pnin->next=NULL ;
```

Quelle erreur grossière est présente dans le code ci-dessus ?

TODO 04 : Créer une fonction d'initialisation de la structure `struct ninja`.	
Créer un fichier `structures.c`.	Y inclure `structures.h`.
Ecrire une fonction d'initialisation d'un pointeur sur une structure de type `NINJA`. Elle renvoie un pointeur alloué dynamiquement, initialisé grâce aux arguments d'entrée (`name`, `chakra`, `p` et `n`).	`NINJA* createNINJA(char[],int, NINJA*, NINJA*);`
Dans la fonction principale	Créer 3 pointeurs initialisés de type `NINJA*` grâce à la fonction `createNINJA`. Les arguments `p` et `n` seront mis à `NULL` pour chaque appel de la fonction.
Dans la fonction principale	Chaîner les 3 éléments précédents. La première (la tête) gardera son champ `p` à NULL avec `n` initialisé, la seconde aura `p` et `p` initialisés, la dernière (la queue) gardera son champ `n` à NULL avec `p` initialisé.
Sauvegarder, compiler et exécuter avec le debugger. En cliquant sur les pointeurs `previous` et `next`, vous devriez voir apparaître toute la liste chaînée.	

Les maillons d'une liste chaînée sont donc connectés les uns aux autres. La liste chaînée peut alors être entièrement définie par sa tête et sa queue qu'on peut rassembler dans une nouvelle structure en C qui sera alors la "liste".

TODO 05 : Créer et manipuler une liste chaînée.	
Dans le fichier `structures.h`.	Créer une structure `struct clan` qui intègre : — `nom`, tableau de char de taille `NB_LETTRES+1` — `nombre`, entier `int` — `hokage` (tête de liste), pointeur sur une structure `NINJA` — `genin` (queue de liste), pointeur sur une structure `NINJA`
Pour vous éviter de coder toutes les fonctions nécessaires à la manipulation des listes chainées, on vous les donne. Télécharger le fichier `fonctions_unnamed_vNinja.c` sous Moodle.	Pour chaque fonction, — Etudier précisément ce qu'elle fait en faisant des dessins sur le chainage des maillons. — La recopier dans votre fichier `structures.c` et lui donner un nom plus explicite. — Insérer le prototype correspondant dans votre fichier `structures.h`. — Mettre un commentaire en haut de chaque prototype.
Dans la fonction principale	— Créer 5 pointeurs sur `NINJA` avec les champs `previous` et `next` initialisée à NULL. — Créer une variable de type `CLAN` en initialisant le champ `hokage` et `genin` avec deux des 5 pointeurs NINJA. — Chaîner les pointeurs NINJA (en les ajoutant ainsi au CLAN) en utilisant les fonctions déchiffrées. — "Tuer" trois (pointeurs sur) NINJA en utilisant les fonctions déchiffrées. — "Annihiler" entièrement le CLAN.
Sauvegarder, Compiler et exécuter avec le debugger.	Suivre l'évolution de la liste chaînée.

TD6.3 Autres structures de données : pile et file

Vous allez maintenant vous-même créer une structure de pile et de file à l'aide d'une liste chaînée.

Une pile (LIFO : Last In First Out) est une structure de données dont l'insertion et la suppression se font toujours en haut (tête) de liste (pensez pile d'assiettes : je pose une assiette en haut d'une pile existante, et je reprends l'assiette en haut de la pile pour l'utiliser)

Une file (FIFO : First In First Out) est une structure de données dont l'insertion se fait toujours en queue de liste et la suppression en en tête de liste (pensez queue à la Poste : premier arrivé, premier servi. Le reste : FAITES LA QUEUE !)

TODO 06 : Créer une pile avec une liste chaînée.	
Créer les nouveaux fichiers `pile.c` et `pile.h`.	Dans `pile.c`, inclure uniquement les fonctions nécessaires au bon fonctionnement de la pile en piochant parmi les fonctions de la liste chaînée. Copier dans `pile.h` les structures et prototypes utiles.
Sauvegarder, Compiler et exécuter avec le debugger.	Suivre l'évolution de la pile.

TODO 07 : Créer une file avec une liste chaînée.	
Créer les nouveaux fichiers `file.c` et `file.h`.	Dans `file.c`, inclure uniquement les fonctions nécessaires au bon fonctionnement de la file en piochant parmi les fonctions de la liste chaînée. Copier dans `file.h` les structures et prototypes utiles.
Sauvegarder, Compiler et exécuter avec le debugger.	Suivre l'évolution de la file.

Bilan des compétences

— Utilisation des structures en langage C
— Utilisation des pointeurs sur les structures
— Compréhension des structures de données (listes chaînées, pile, file)