**Langage de programmation : bases**

**HAUTE ÉCOLE DE NAMUR-LIÈGE-LUXEMBOURG**

**DA-IA – bloc 1**

Exercices 3A – Pointeurs

Objectif

* Apprendre à gérer les pointeurs à partir d'exercices de "drill"

# Exercice 1 : bases

Soient les instructions suivantes :

#define NB\_REELS **25**

void main(void){

double centiemes[NB\_REELS];

int i;

for(i = 0 ; i < NB\_REELS ; i++){

centiemes[i] = (double)i/100;

}

}

Déclarez un pointeur pCentieme destiné à contenir l'adresse d'un réel.

Faites-en sorte qu'il pointe vers le 3e élément du tableau.

Affichez, via printf et pCentieme, la valeur de la 3e cellule.   
(Vous devez voir 0.02)

À l'aide d'une opération arithmétique sur pCentieme, faites-en sorte qu'il pointe vers le 6e élément du tableau et ensuite affichez le.  
(Vous devez voir 0.05)

Via pCentieme toujours, ajoutez 1 à la valeur pointée et réaffichez le contenu de cette 6e cellule.  
(Vous devez voir 1.05)

Via pCentieme toujours, ajoutez 4 à la valeur pointée et réaffichez le contenu de cette 6e cellule.  
(Vous devez voir 5.05)

# Exercice 2 : opérations arithmétiques

Qu'affichent les instructions successives suivantes ?  
Répondez d'abord sans coder ni exécuter le code. Vérifiez ensuite vos réponses via l'exécution du code correspondant.

int entiers[] = {12, 23, 34, 45, 56, 67, 78};

int \*pEntier = entiers;

printf("%d", \*pEntier); // 12

printf("%d", \*++pEntier); // 23

printf("%d", \*(pEntier+3)); // 56

printf("%d", \*pEntier--); // 23

printf("%d", --\*pEntier); // 11

# Exercice 3 : opérations arithmétiques

Qu'affichent les instructions successives suivantes ?  
Répondez d'abord sans coder ni exécuter le code. Vérifiez ensuite vos réponses via l'exécution du code correspondant.

int multiples100[ ] = { 100, 200, 300, 400, 500, 600 };  
int \*pMultiples100 = multiples100 + 2;  
  
printf("%d\n", \*pMultiples100); // 300  
printf("%p\n", pMultiples100); // L’adresse de la 3ème cell   
printf("%d\n", \*++pMultiples100); // 400  
printf("%d\n", \*(pMultiples100 - 1)); // 300  
printf("%d\n", \*pMultiples100 + 70); // 470  
printf("%d\n", --\*pMultiples100); // 399

printf("%d\n", \*--pMultiples100); // 300

# Exercice 4 : les chaines de caractères

Quelle est la valeur de caract après chacune des instructions consécutives suivantes ?

Répondez d'abord sans coder ni exécuter le code. Vérifiez ensuite vos réponses via l'exécution du code correspondant.

char option[] = "Informatique";

char \*pOption = option;

char caract;

caract = \*(pOption + 3); // ‘o’

caract = \*++pOption; // ‘n’

caract = \*pOption++; // ‘n’

caract = ++(\*pOption); //‘g’

puts(option);

# Exercice 5 : passage par référence/adresse

Remplacez les pointillés pour que l'affichage soit :

*nombre1 vaut 20.3, nombre2 vaut 100.5*

void echangeDeuxReels (………………………………………………………………………………)

{

double temp;  
 temp = ……………;  
 ……………………………………………;  
 ……………………………………………;  
}

void main(void)

{

double nombre1 = 100.5;

double nombre2 = 20.3;  
 echangeDeuxReels( ……………………………………………………………………………………………);  
 printf("nombre1 vaut %.1f, nombre2 vaut %.1f", nombre1, nombre2);

}

# Exercice 6 : allocation dynamique 🡪 chaine de caractères

1. Déclarez un pointeur pSection destiné à contenir l'adresse d'une chaine de caractères
2. Allouez la place mémoire nécessaire pour "Informatique de gestion" et stockez l'adresse dans pSection
3. Affectez ce littéral à la valeur pointée par pSection
4. Affichez la chaine de caractères (via pSection)
5. Affichez l'initiale de la chaine de caractères (via pSection)
6. Libérez la place mémoire allouée et vérifiez la valeur de pSection

# Exercice 7 : allocation dynamique 🡪 listes chainées

Soit une liste chainée simple où chaque chainon concerne un étudiant et contient son nom (50 caractères tout compris), sa date de naissance (AAAAMMJJ) et un pointeur vers le chainon suivant.

1. Déclarez la structure d'un chainon et donnez-lui un synonyme (Etudiant)
2. Déclarez pDebEtudiants, pointeur début de liste et initialisez-le à NULL
3. Allouez de la place mémoire pour un nouveau chainon pNouvEtudiant
4. Demandez à l'utilisateur le nom et la date de naissance d'un étudiant et affectez ces valeurs aux données du nouveau chainon
5. Ajoutez le chainon en début de liste
6. Affichez les étudiants (un seul) contenus dans la liste, à partir de pDebEtudiants

# Exercice 8 : passage par adresse d'un pointeur

Fonction d'ajout d'un chainon en début de liste.

Reprenez la liste chainée de l'exercice précédent (7). Le but est d'ajouter plusieurs chainons, chaque fois en début de liste.

Traduisez le DA suivant où chaque module est une fonction :

o────────────────o

│ listeEtudiants │

o────────────────o

┌─── \*

│ pDebEtudiants = NULL

│ codeErreur = PAS\_D\_ERREUR

│ obtenir nom

│╔══ while (codeErreur == PAS\_D\_ERREUR AND nom ≠ "")

│║ // allocation mémoire pour un nouveau chainon

│║ o────────────────────────o

│║ │ nouveauChainonEtudiant │

│║ o────────────────────────o ↓ codeErreur, pNouvEtudiant

│║┌── if (codeErreur ≠ ERREUR\_MEMOIRE)

│║│ obtenir dateNaissance

│║│ // Ajout en début de liste

│║│ o────────────────o ↓ pDebEtudiants, nom, dateNaissance, pNouvEtudiant

│║│ │ ajouteEtudiant │

│║│ o────────────────o ↓ pDebEtudiants

│║│ obtenir nom

│║└──

│╙──

│ o──────────────────o ↓ pDebEtudiant

│ │ afficheEtudiants │

│ o──────────────────o

│ libérer pDebEtudiants

└──────────

Remarques

* La fonction nouveauChainon ne peut renvoyer 2 valeurs. Il faut donc que l'une des variables (codeErreur ou pNouvEtudiant) soit passée par adresse à la fonction. Celle-ci reçoit donc l'adresse du pointeur pNouvEtudiant et renvoie la valeur de codeErreur.

CodeErreur nouveauChainon( ... pNouvEtudiant);

* La fonction ajouteEtudiant reçoit et renvoie pDebEtudiants (vu qu'on ajoute le nouveau chainon en début de liste).   
  Pour permettre de renvoyer un code d’erreur par la suite (on ne le gère pas encore ici…), elle reçoit l'adresse du pointeur pDebEtudiants.

void ajouterEtudiant( ... pDebEtudiants, … );