Fiche de TD/TP du chapitre II

Partie1: les pointeurs en pseudo-code

Exercice 1:

1a. type: ↓réel

1b. type: ↓entier

1c. type: ↓↓entier

1d. type: ↓caractère

exercice 2:

2a. affiche une valeur: 5

2b. affiche l'adresse de p

2c. affiche l'adresse de n

2d. affiche une valeur 2

2e. affiche une valeur 2

2a. affiche l'adresse de n

Exercice 3:

3a. En C, Le pointeur p a une valeur aléatoire du fait qu'il n'a pas été initialisé. L'adresse en question n'est pas une adresse à laquelle le programme devrait avoir accès donc une écriture de son contenu provoquera l'arrêt du programme (par raison de sécurité)

Début

p ← allouer(NbOctet(entier))

©p ← 2

Fin

3b. Le point ne point vers aucune zone mémoire; Il n'a donc pas d'espace où il peut stocker ces informations.

Début

p ← NIL

p ← allouer(NbOctet(entier))

©p 2

Fin

3c.@n est l'adresse la variable n, elle est donc constante (on peut pas lui affecter une autre valeur que l'initiale) : ce n'est pas une Ivalue

Début

p ← @n

n ← 2

Fin

3d.©p + i est une valeur (la valeur du contenu de p + celle de i), on ne peut donc lui affecter de valeur. Ce n'est pas l'espace mémoire que l'on a réservé à l'initialisation de p car l'opérateur unitaire © est prioritaire sur + .

Début

p ← allouer(10 * NbOctet(Entier))

pour i 0(1)9 faire

©(p + i) ← 0

fin Pour

Fin

Exercice 4:

Algorithme: valeurs_et_contenus_de_pointeurs

/***Objectif:** Cet algorithme va permettre d'affiche la valeur et le contenu de deux pointeurs connus*/

VARIABLE

↓réel: p1, p2

Début

p1 ← allouer(NbOctet(réel))

p2 ← allouer(NbOctet(réel))

©p1 ← 14,5

©p1 ← -5,75

écrire(p1)

écrire(p2)

écrire(©p1)

écrire(©p2)

Fin

Partie2 : Traduction, en pseudo-code, les programmes écrits en langage C suivants :

Exo1.pseudo-code

Algorithme : *Bref_manipulation_des_pointeurs*
*/****Objectif :*** *Bref manipulation des pointeurs pour se familiariser avec les notions d'adresse et de contenu et leur équivalents en C*/*

VARIABLE
entier : n
↓ entier : p
Début
n ← 3 // l'initialisation peut également se faire après celle de p.
p ← @n
écrire(@p, p, ©p, @n, n)/ affiche l'adresse de p, puis l'adresse de n, ensuite la valeur de n, suivit de l'adresse de n, et enfin affiche la valeur de n*/*
Fin

Exo2.pseudo-code

Algorithme : *Déclaration*
*/****Objectif :*** *Déclaration d'un variable pointeur sur les caractères et d'un variable caractère : la différence entre les deux.*/*

VARIABLE
↓ caractère : p1 (ou chaîne de caractère : p1)
caractère : p2
Début
//Rien à signaler.
Fin
/*On aurait pu s'attendre à ce que p2 soit un pointeur : il n'en est rien. En C, quand on colle l'étoile à un type, seul la variable qui le suit est un pointeur. Si on voulait que p1 et p2 soient des pointeurs, il aurait fallu écrire « char *p1, *p2 ; »*

Exo3.pseudo-code

Algorithme : *L'opérateur incrémentation*
/***Objectif :*** Nous allons voir comment utiliser l'opérateur incrément sur les pointeurs pour modifier la valeurs du nombre sur lequel il pointe.*/

VARIABLE
entier n
↓entier p

Début
n ← 5
p ← @n
n ← n + 1
©p ← ©p + 1 // incrémentation de n
écrire(n, ©p) // n = 7, ©p = 7 car n égale à ©p
Fin

Exo4.pseudo-code

Algorithme : Dérouler
/***Objectif :*** Cet algorithme vous permet de comprendre comment est-ce que l'on peut utiliser le contenu de p (qui est un pointeur) à la place de la variable sur laquelle il pointe.*/

VARIABLE
entier a, b, c
↓entier pa, pb, pc

Début
a = 3
b = 10
pa = @a
©pa = ©pa * 2 // a = 6
pb = @b
c = 3 * (©pb - ©pa) /*l'opérateur contenu est prioritaire sur tout opérateur arithmétique binaire et c = 3 * (b – a) = 3 * (10 – 6) = 12*/
pc = pb // Il s'agit de deux pointeurs pc = @b
pa = pb
pb = pc // pa = pb = pc = @b
Fin

Exo5.pseudo-code

Algorithme : Dérouler
/*Objectif : Il est question de vous montrer que l'allocation de l'espace à un pointeur en C n'est qu'une formalité.*/

VARIABLE
entier : i,
↓entier : p

Début
i ← 0
p ← @i // l'allocation est une formalité en C d'où l'absence de cette instruction
écrire(©p + 1) // affiche 1
Fin
//Le programme plante car p ne pointe pas sur un case mémoire utilisable par notre programme

Exo6.pseudo-code

Algorithme : Trois_utilisation_possible_de_la_multiplication*
/***Objectif :** Il est question de vous montrer quels sont les trois usages que l'on peut faire du symbole*/

VARIABLE
↓entier : p, q
entier : i, j // usage de * pour déclarer les pointeurs en C

Début
i = 10
j = 5
p = @i
q = @j
écrire(©p * ©q) /* usage de * pour la multiplication en C et en pseudo code. Et on utilise aussi * pour faire référence au contenu d'un pointeur en C*/
Fin

Exo7.pseudo-code

Algorithme : Double_indirection
/***Objectif :** Cet algorithme vous montrer comment déclarer et utiliser un pointeur sur un pointeur*/

VARIABLE
entier : i
↓entier : j
↓↓entier : k
Début
j ← @i
k ← @j //k pointeur sur le pointeur j
écrire(k, ©k, ©©k) // k = @j, ©k = @i, ©©k = i
Fin

Exo8.pseudo-code

Algorithme : Manipulation_des_affectations_composées

/*Voir comment examiner une expression qui comporte plusieurs affectation.*/

VARIABLE
entier : a, b
↓entier : p1, p2

Début
a ← 5
b ← a
p1 ← @a
a ← a + 1 // ©p1 = a = 6
p2 ← @b
a ← b // ©p1 = a = b = 5
b ← b + a // ©p2 = b = 10
écrire(' a=' a, ' b= ' b, '©p1=' ©p1, ' ©p2=' ©p2)// a=5, b=10, ©p1=5, ©p2=10
Fin
/* Il faut faire attention aux mots collés : intmain.*/

Exo9.pseudo-code

Algorithme : Taille_d'un_pointeur
/***Objectif :** Cet algorithme vous révélé la taille d'un pointeur*/

VARIABLE
↓entier p
entier n

Début
n = Taille(p) /*le contenu de p prend la la taille de p en octets soit 4 octets sur la plupart des compilateurs car p est un entier. La variable b n'est pas déclaré et non objective pour notre programme.*/
écrire(n) // n = 4 ceci est fonction de votre ordinateur.
Fin

Exo10.pseudo-code

Algorithme : Dérouler
/***Objectif :** Déférence de pointeurs*/

VARIABLE
entier : i, j
↓entier : p, q

Début
i ← 5
p ← @i
q ← @j
j ← 5
écrire(©p, ©q) // On a défère les pointeurs p et q ; ©p = 5 ©q = 5

Fin

Exo11.pseudo-code

Algorithme : Les_opérateurs_&_et_*
/***Objectif :** Cet algorithme vous permet de comprendre comment évaluer les expressions qui contiennent à la fois les opérateurs & et * sont évalués. */

VARIABLE
entier : i
↓entier : p

Début
i ← 5
p ←@i
écrire(©@p, @©p) /* On peut partir du contenu pour retrouver l'adresse et vice versa. Ce qui affiche @n, @n*/
Fin

Exo12.pseudo-code

Algorithme : Allocation
/***Objectif :** Montrer comment l'on peut allouer un espace mémoire dont l'on n'ignore l'identifiant pour contenir le ©p*/

VARIABLE
entier : i
↓entier : p

Début
p ← allouer(2 * Taille(entier))
©p ← 0
©(p+1) ← 0
écrire(©p, ©(p+1)) /* ©p concerne la première case mémoire réservée et ©(p+1) concerne la deuxième case mémoire réservée. La fonction calloc initialise les contenu des pointeurs à zéros.Donc on verra 0, 0*/
Fin

Exercice11

Pour compléter ce tableau vous devez avoir en tête la notion de priorité de l'incrémentation sur la multiplication, les notions d'incrémentation postfixée/préfixée

	A	B	C	P1	P2
Initialisation	1	2	3	/	/
P1=&A	1	2	3	&A	/
P2=&C	1	2	3	&A	&C

*P1=(*P2)++	4	2	4	&A	&C
P1=P2	4	2	4	&C	&C
P2=&B	4	2	4	&C	&B
*P1-=*P2	2	2	4	&C	&B
++*P2	2	3	4	&C	&B
P1=*P2	2	3	12	&C	&B
A=++*P2**P1	48	4	12	&C	&B
P1=&A	48	4	12	&A	&B
*P2=*P1/=*P2	12	12	12	&A	&B

Exercice12 : ***Extrait de l'examen de rattrapage 2013-2014***

a. var1 = p3 *type : ↓↓entier* *valeur :0x4B32*
b. var2 =&p3 *type : ↓↓↓entier* *valeur : 0x2C2F*
*c. var3 = *p2* *type : ↓entier* *valeur :0x1A40*
*d. var4 = **p2* *type : entier* *valeur : 20*
*e. var5 = *p3* *type : ↓entier* *valeur :0x1A40*
*f. var6 = **p3* *type : entier* *valeur : 20*

PARTIE3 : les sous-procedures

Exercice1 :

AlgorithmeExo1a

Exercice2 : Suite de Fibonacci

F(0)=0,
F(1)=1,
F(n)=F(n-1)+F(n-2).

1. *Écrivons une fonction pour calculer le terme de rang n de cette suite soit F(n)*

- *Fonction itérative pour calculer le terme de rang*

réel F(entier : n)
*/**Objectif :** Cette fonction vous permet de calculer le terme rang n de la suite de Fibonacci par itération.*/*

VARIABLE
entier : F, Fp, Fg, k

Début

```
    Cas où(n)
        (0) : F ← 0
        (1) : F ← 1
        sinon
            Fp ← 0
            Fg ←1
            Pour k ← 2 à n faire
                Fg ← Fg + Fp
                Fp ← Fg - Fp
            fin pour
            F ← Fg
    fin cas où
    rétourner(F)
Fin
```

- *Fonction récursive pour calculer le terme de rang n de la suite de Fibonacci*

réel F(entier : n)
*/****Objectif :*** *Cette fonction vous permet de calculer le terme de rang n de suite de Fibonacci en utilisant la récursivité.*/*

VARIABLE

Début

```
    cas où(n)
        0 : retourner 0
        1 : retourner 1
        si non
            retourner( F(n-2) + F(n-1) )
    fin cas où
```

Fin

Remarques :

- *La fonction récursive fait moins réfléchir que la fonction itérative. Dans certains cas donc la récursive vous permettra de simplifier les choses.*
- *De plus vous pourrez bien remarquer que les fonctions récursives sont bien plus gourmandes en espaces mémoires que la fonction itérative. Par contre elle peut être plus rapide que la fonction itération uniquement lorsque vous avez une bonne RAM ; C'est ce qui fait son avantage.*
- *Vous pourrez ressoudre ce problème en utilisant les sélections simples ; C'est-à-dire des si..alors, sinon….*

Nous allons maintenant traduit nos deux algorithme en C.

- *Après avoir traduit votre algorithme en C assez d'exécuter F(100) vous remarquerez que la fonction itérative est mieux adapter pour la plus part des machines que la fonction récursive*

2. *Pour le programme qui va vous permettre de calculer les N premiers termes de la fonction vous devez appeler la fonction F(k) N-1 fois pour k allant de 0 à N*

Algorithme : *Calcul_terme_de_rang_n_de_la_suite_de_Fibonacci*

*/*objectif : Nous allons écrire une fonction qui permet a partir d'un nombre n donné par l'utilisateur de calculer le terme de rang n de la suite de Fibonacci donné ci-haut.*/*

VARIABLE
entier : N, k
/ Écrire l'une des deux fonctions ci-haut ici*/*

*/*Notre programme proprement dit*/*
Début
écrire(« Entrez le nombre N et nous vous fournirons les N premiers termes de la suite de Fibonacci :)
lire(N)
pour k ← 0 à N-1 faire
écrire(« Le »k « -ieme terme est » F(k)) // ceci affiche le k-ième terme de notre suite
fin pour
Fin

Exercice3 :

1. *Écriture en C de la fonction récursive et itérative de factorielle :*

Nous vous conseillons de toujours commencé par faire l'algorithme avant de le traduit en C. Ceux-ci n'est pourtant par une règle obligatoire. Si vous trouvez le programme simple pour vous, vous pouvez directement passé au C mais attention aux erreur.

- *Fonction factorielle itérative*

```
Entier : fact( entier n)
    entier : k, N
    Début
        N ← 1
        si (n # 0) alors
            pour k ← 1 à n faire
                N ← N * k
            fin pour
        retourner N
    Fin
```

- *Fonction factorielle récursive*

```
Entier : fact(entier n)
    Début
        si(n = 0) alors
            retourner 1
        sinon
            retourner n * fact(n-1)
        fin si
    Fin
```

2. *Pour la fonction qui calcule la combinaison nous allons utilisé le fait que le factorielle d'un nombre fait intervenir trois factorielles. Ainsi vous ferons intervenir la fonction factorielle jadis*

écrite.

Exercice 4 :

Nous allons dérouler cette fonction à la main vérifier ce quel donne :

Lors de la déclaration : n1=5, n2=8, n3=10

après f1:n1=5, n2=8, n3=10 // il n'y aucune modification parce que le nombre n1 est passer en valeur

après f2 : n1=5, n2=8, n3=5 // n3 est modifier et prend la valeur de n1 cas elle est passé en paramètre à la fonction.

*Après f4 :*n1=5, n2=8, n3=8 // n3 est passé en paramètre a f4 puis à f3 avant d'être modifier dans celui-ci

Exercice 5 :

1. Écriture de la fonction : entier FONCTION pgcd(entier a, b) en fonction des critères spécifiés

- *J*
- *Pour cette deuxième partie on remarquer que nous avons à faire à la récursivité.*

Entier FONCTION pgcd(entier a, b)
/***Objectif :** Cette fonction permet de calculer le pgcd de deux nombres entiers */

VARIABLE
entier : N, c

Début
si(a < b)
c ← a
a ← b
b ← c
fin si // Je me rassure que a est plus grand que b.
si(a mod b = 0)
retourner b
sinon
retourner pgcd(b, a mod b)
retourner N
Fin

2. Nous allons déterminer si deux nombres entiers que vous allez entrer au clavier sont premiers ou pas en appelant la fonction pgcd écrite si-haut.

Algorithme : Nombres_premier_ou_pas
/*Objectif : Cette algorithme détermine si les deux nombres que vous allez entrez au clavier sont premiers entre eux ou pas*/

VARIABLE
entier : n, m

/*Mettre ici la fonction pgcd*/

Début
écrire("Entrez successivement deux nombres entiers positifs et nous vous dirons s'ils sont premiers ou pas")
lire(n)
lire(m)
si (pgcd(n, m) = 1)
écrire(n "et" m "sont premiers entre eux")
sinon
écrire(n "et" "ne sont pas premiers entre eux")
fin si
Fin

Voir la traduction en C.

Exercice 6 :

1. Écriture de la fonction : entier FONCTION puissance(entier : n, p)

a. *Calcule par itération :*

entier FONCTION puissance(entier : n, p)
/*Objectif : Cette fonction calcule la puissance n expo(p) par itération.*/

VARIABLE
entier : k, N

Début
N ← 1
si(p # 0) alors
pour k ← 1 à p faire
N ← N * n
fin pour
fin si
/* Par défaut le résultat est 1 c'est à dire que si l'on a n expo(0) le résultat serait 1*/
retourner N
Fin

b. Calcule par récursivité :

entier FONCTION puissance(entier : n, p)
/***Objectif :** Cette fonction calcule la puissance n expo(p) par récursivité.*/

VARIABLE
entier :

```
Début
    si(p = 0) alors
        retourner 1
    sinon
        retourner (n *  puissance(n, p-1))
    fin si
Fin
```

2. Pour l'algorithme suivant :

a. Traduction en C.

b.Si Nombre = 7 et D = 2, la valeur affichée est 111

Si nombre = 8 et D = 2, la valeur affichée est 1000

c. Objectif de l'algorithme : conversion du nombre N en base D

<u>Exercice 7 :</u> Programme qui calcule C qui calcule les solutions d'une équation du second dégré.

Algorithme : Solution_d'une_équation_du_second_degré

/***Objectif :** Cet algorithme va vous permettre de déterminer les solutions d'une équation du second degré a coefficients entiers.*/

```
VARIABLE
    entier : Det, a, b, c, x1, x2

Début
    écrire("Entrez les entiers a, b et c en sorte que l'on ait cette équation : ax2 + bx +c = 0 et nous allons vous proposez ces solutions.")
    écrire("Entrez a :")
    lire(a)
    écrire("Entrez b :")
    lire(b)
    écrire("Entrez c :")
    lire(c)
    si(a=0) alors
        écrire("Il ne s'agit pas d'un équation du second degré. Tout fois nous allons essayer de calculer la solution de cette équation si elle existe.")
        si(b=0) alors
            si(c=0) alors
                écrire(" Votre équation admet une infinité de solutions.")
            sinon
                écrire("Votre équation n'admet pas de solution")
            fin si
        sinon
            écrire("La solution de votre équation est :" - c/b)
        fin si
```

```
        sinon
                Det ← b2 – 4ac
                si(Det < 0)
                        écrire("Votre équation n'admet pas de solutions réelles")
                sinon
                        si( Det=0)
                                écrire("Votre équation admet une solution double")
                                écrire("L'unique solution de votre équation est :" -b/2a)
                        sinon
                                écrire("Votre équation admet deux solutions distinctes")
                                x1 ← (-b + racine(Det)) / 2a
                                x2 ← (-b - racine(Det)) / 2a
                                écrire("Les solutions de votre équation sont les suivantes :" x1, x2)
                        fin si
                fin si
        fin si
        écrire("Merci d'avoir exécuter ce programme")
Fin
```