ECUE «Prog 1» - CC3
5 janvier 2017 - Bruno Bouzy
sans document - durée 1 heure 30

Dans les exécutions de programmes, les entrées clavier sont en **gras** et les sorties en police normale.

Exercice 1 (4 points)

Donner la sortie du programme ci-dessous. Pour chacune des trois lignes avec commentaire, tenir compte de la couleur (Bleu, Rouge, Vert ou Jaune) de votre copie pour utiliser la valeur précisée dans le commentaire de la ligne à la place de la valeur donnée en gras dans la ligne.

```
#include <stdio.h>
int main() {
  int a = +2;                    // Bleu:-1, Rouge:+2, Vert:-3, Jaune:+2
  int * p = &a;
  int b = *p;
  printf("1: a = %d, b = %d, *p = %d.\n", a, b, *p);
  a *= -4;                       // Bleu:+5, Rouge:-2, Vert:+3, Jaune:-4
  printf("2: a = %d, b = %d, *p = %d.\n", a, b, *p);
  b += -1;                       // Bleu:+2, Rouge:-3, Vert:+4, Jaune:-1
  printf("3: a = %d, b = %d, *p = %d.\n", a, b, *p);
  int * q = &b;  printf("4: a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);
  *q += (*p)++;  printf("5: a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);
  *q *= ++(*p);  printf("6: a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);
  p = q;         printf("7: a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);
  q = &a;        printf("8: a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);
  return(0);
}
```

Exercice 2 (7 points)

1) Ecrire une procédure `void affiche` prenant un tableau `t` d'entiers et une taille `n` du tableau, et affichant les éléments du tableau en respectant les sorties du haut de la page suivante. **(1 point)**

2) Ecrire une procédure `void init` prenant en entrée un tableau `t` d'entiers et une taille `n` du tableau, et initialisant `t` avec des nombres entrés au clavier compris entre `0` et `9`. Tant qu'un nombre entré au clavier n'est pas compris entre `0` et `9`, la procédure redemande le nombre. La procédure respectera les entrées sorties du haut de la page suivante. **(2 points)**

3) Ecrire une procédure `void taprosca` prenant en entrée deux tableaux `u` et `v` d'entiers et la taille `n` des deux tableaux, n'affichant rien, et donnant en sortie les trois produits scalaires `u.u`, `v.v`, et `u.v`, où les tableaux `u` et `v` sont considérés comme des vecteurs. **(2 points)**
Rappel mathématique: `u.v = Σu[i]v[i]`

4) Ecrire un programme `main` déclarant deux tableaux `A` et `B` de taille `3`, initialisés avec `init`, affichés avec `affiche`, calculant avec `taprosca` les produits scalaires `A.A`, `B.B` et `A.B`, où les tableaux `A` et `B` sont considérés comme des vecteurs, et affichant les résultats. Le programme respectera les entrées-sorties du haut de la page suivante. **(2 points)**

```
A ?
v0 ? 10
v0 ? 3
v1 ? -1
v1 ? 4
```

```
v2 ? 7
A = { 3 4 7 }
B ?
v0 ? 5
v1 ? 6
```

```
v2 ? 2
B = { 5 6 2 }
A.A = 74,
B.B = 65,
A.B = 53.
```

Exercice 3 (9 points)

La fonction de Ackermann `A` est définie comme suit. Pour `m` et `n`, deux entiers positifs ou nuls,

Si `m=0`, alors	`A(0,n) = n+1`
sinon si `n=0`, alors	`A(m,0) = A(m-1,1)`
sinon	`A(m, n) = A(m-1, A(m, n-1))`

1° Calculer `A(1,0)`, `A(1,1)`, `A(2,0)`. **(1 point)**
2° Avec la définition, programmer la fonction `ackermann` de manière récursive. **(1 point)**

3° Programmer la fonction `ackermann_2` de manière récursive et analogue à la fonction `ackermann` de la question 2° en lui rajoutant un paramètre de sortie `int * count` donnant le nombre d'appels récursifs effectués. **(1 point)**
4° Calculer `A(1,2)` et `A(1,3)` et donner le nombre d'appels pour chaque calcul. **(1 point)**
5° Calculer `A(2,1)`. Pour ce calcul, `ackermann_2` effectue-t-elle deux fois certains calculs ? Lesquels ? **(1 point)**

Pour éviter de faire faire deux fois certains calculs, on va utiliser un tableau d'entiers `tab` déclaré dans le `main` et stockant les valeurs de Ackermann déjà calculées jusque là. Au début de la fonction (nommée `ackermann_3` et appelée avec les paramètres `m` et `n`, le compteur `cnt_3` et le tableau `tab`), la fonction teste si `tab` contient déjà la valeur `A(m, n)`. Si oui, elle retourne la valeur. Sinon, elle s'exécute et, avant de faire le `return`, elle stocke la valeur calculée dans le tableau `tab`. On stocke `A(m, n)` dans `tab[N_MAX*m+n]` où `N_MAX` est une constante fixée à `200000` avec un `#define`. Avant l'appel initial de la fonction, le tableau `tab` est initialisé avec des `0`. (Les valeurs de Ackermann étant strictement positives, une valeur de case du tableau strictement positive correspondra à une valeur déjà calculée, et une valeur nulle à une valeur non calculée.)

6° Programmer la fonction `ackermann_3` en rajoutant un paramètre en entrée `int * tab` où `tab` est le tableau contenant les valeurs déjà calculées jusque là. **(2 points)**

7° Programmer un `main` déclarant les entiers `m`, `n`, `a`, `i`, `cnt_2`, `cnt_3`, le tableau d'entiers `tab` de taille `1000000` (définie par `TAILLE_TAB`), demandant au clavier les valeurs de `m` entre `0` et `3` (définie par `M_SUP`) et de `n` entre `0` et `12` (définie par `N_SUP`), appelant `ackermann_2` et `ackermann_3` de manière adéquate et affichant les résultats : `A(m, n)`, le nombre d'appels avec ou sans utilisation d'un tableau. Le programme respectera les entrées-sorties suivantes : **(2 points)**

```
m ? 1
n ? 0
A(1, 0) = 2,
n appels sans = 2,
n appels avec = 2.
```

```
m ? 2
n ? 2
A(2, 2) = 7,
n appels sans = 27,
n appels avec = 15.
```

```
m ? 3
n ? 6
A(3, 6) = 509,
n appels sans = 172233,
n appels avec = 1277.
```