info719 : Rappels d'algorithmique et programmation C
Examen
SOLUTIONS

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 22, poste : 94 22
email : Pierre.Hyvernat@univ-savoie.fr
www : http://www.lama.univ-savoie.fr/~hyvernat/
wiki : http://www.lama.univ-savoie.fr/wiki

Durée : 2h ; documents et calculatrice non-autorisés.
Un barème provisoire est donné dans la marge.
1 point sera réservé à la présentation de vos réponses et de vos calculs...

Partie 1 : langage C

(2) *Question 1.* Qu'affiche le morceau de programme suivant ?

```
{
  int m, n, p;
  n = 3; m = n+1;
  {
    int n, p;
    p = m+7;
    n = 4;
    printf("(1) m=%i, n=%i et p=%i\n",m,n,p);
  }
  printf("(2) m=%i, n=%i et p=%i\n",m,n,p);
}
```

Solution : le programme affiche

```
(1) m=4, n=4 et p=11
(2) m=4, n=3 et p=-1077735528
```

(La valeur de `p` sur la deuxième ligne ne correspond à rien...)

(3) *Question 2.* Même question que précédemment :

```
{
  int m, n;
  int *p, *q;
  int **x;
  m = 42;
  p = &m;
  q = &n;
  *q = m / 2;
  q = p;
  printf("(1) m=%i, n=%i, *p=%i et *q=%i\n",m,n,*p,*q);

  x = &q;
```

```
  m = n+1;
  **x = 17;
  *p = **x;
  *p = *q;
  printf("(2) m=%i, n=%i, *p=%i, *q=%i et **x=%i\n",m,n,*p,*q,**x);
}
```

Solution : le programme affiche

```
(1) m=42, n=21, *p=42 et *q=42
(2) m=17, n=21, *p=17, *q=17 et **x=17
```

(2) *Question 3.* En C, nous avons vu deux manières de déclarer un tableau de taille `100` :

- `int T[100] ;`
- `int *T = malloc(100*sizeof(int)) ;` .

Quelles sont les différences entre ces deux déclarations ? Laquelle préférez-vous, et dans quelles circonstances ?

Solution :

- La déclaration "`int T[100]`" permet de déclarer un tableau sur la pile. L'accès à la pile est très rapide, mais la durée de vie du tableau est alors locale au bloc où il a été déclaré. (En particulier, on ne peut pas faire un `return(T)` si se tableau est déclaré dans une fonction.) On ne peut pas modifier la taille du tableau.
- La déclaration "`int *T = malloc(100*sizeof(100))`" permet de déclarer un tableau dans le tas. Un tel tableau a une durée de vie illimitée (jusqu'à la fin du programme, ou jusqu'à un "`free(p)`"). On peut donc renvoyer la valeur de `T` si on est dans une fonction. De plus, il est possible de modifier la taille du tableau. (Le système devra peut-être changer le tableau d'emplacement, mais c'est le système qui gérera ça.) Par contre, l'accès ou tas est un peu plus lent que l'accès vers la pile.

Partie 2 : complexité

(3) *Question 1.* Parmi les affirmations suivantes, lesquelles sont vraies, et lesquelles sont fausses ? (Justifiez brièvement vos réponses.)

- $n^3 + n^2 = O(n^3)$,
- $n^3 - n^2 = \Omega(n^3)$,
- $(n+2)^{k+1} = O\big((n+1)^{k+2}\big)$ pour une *constante* entière $k > 0$,
- $\log(n) = O\big(\log\big(\log(n)\big)\big)$,
- $\sqrt{n} = \Omega\big(\log(n)\big)$,
- $\sin(\frac{n\pi}{3}) = O\big(\ln(n)\big)$.

Solution : on rappelle que $f = O(g)$ si $\frac{|f(n)|}{|g(n)|}$ est *borné* par une constante, et que $f = \Omega(g)$ si $g = O(f)$.

- Vrai : $\frac{n^3+n^2}{n^3} = 1 + \frac{1}{n} < 3$.
- Vrai : $\frac{n^3}{n^3-n^2} = 1 + \frac{n^2}{n^3-n^2} = 1 + \frac{1}{n-1} < 3$ si $n \geq 2$.
- Vrai car $(n+2)^{k+1}$ est un polynôme de degré $k+1$ et $(n+1)^{k+2}$ est un polynôme de degré $k+2$.
- Faux, car $\log(n) \ll n$ et donc $\log(\log(n)) \ll \log(n)$.
- Vrai : car $\log(n) \ll \sqrt{n}$.
- Vrai : car $\sin(\frac{n\pi}{3}) \leq 1$.

Partie 3 : programmation

Remarque : écrivez vos programmes le plus efficacement possible.

(3) *Question 1.* Écrivez, dans une syntaxe aussi proche du C que possible, une fonction de prototype : "`cherche(int T[], int t, int e);`"

Les arguments de cette fonction sont :
- un tableau d'entiers `T`,
- la taille `t` de ce tableau,
- un entier `e`.

On suppose que le tableau est trié par ordre croissant, et la fonction devra renvoyer :
- `-1` si l'entier `e` n'apparaît pas dans le tableau `T`,
- la valeur d'un indice où l'entier `e` apparaît dans `T` sinon.

Estimez la complexité de votre fonction en fonction de la taille `t`.

Solution : on va faire une recherche dichotomique :

```
int cherche(int T[], int t, int e)
{
  int debut = 0;      // le début du sous-tableau qui nous intéresse
  int fin = t;        // la fin du sous-tableau qui nous intéresse
  int milieu;         // le milieu du sous-tableau qui nous intéresse
  int indice = -1;    // l'indice de l'élément que l'on cherche

  // on regarde au milieu du tableau, et si on trouve l'élément e, on~
  // s'arrête.
  // Sinon, on regarde dans la partie (droite ou gauche) pertinente.
  // On s'arrête quand on obtient un tableau vide.
  while ((indice == -1) && (fin - debut) > 1) {
    milieu = debut + (fin - debut) / 2;
    if (e == T[milieu]) {
      indice = milieu;
    } else {
      if (e < T[milieu]) {
        fin = milieu;
      } else {
        debut = milieu;
      }
    }
  }
  return (indice);
}
```

(À tester...)

Comme on divise la taille du tableau par 2 à chaque étape, on fait au plus $\log \mathtt{t}$) étape, et la complexité est donc en $O\big(\log(t)\big)$.

(3) *Question 2.* Le développement de la fonction sinus au voisinage de zéro est donné par :

$$\sin(x) \quad = \quad x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!} + o(x^{2n+1})$$

Pour comparer les approximations obtenues, on souhaite écrire une fonction

```
double devSinus(float x, int n);
```

En n'utilisant que les fonctions arithmétiques habituelles, expliquez comment vous programmeriez cette fonction.

Solution : Pour éviter de dupliquer les calculs, on garde les valeurs de $(2n+1)!$ et de $(-1)^n x^{2n+1}$ dans des variables tampon.

```
double devSinus(float x, int n)
{
  double numerateur = (double)x;       // la valeur de (-1)^n x^(2n+1)
  double X = (double)(-x * x);
  long int fact = 1;                   // la valeur de (2n+1)!
  int i;
  double resultat = x;

  for (i = 1; i < n + 1; i = i + 1) {
    numerateur = numerateur * X;
    fact = fact * (2 * i) * (2 * i + 1);
    resultat = resultat + (numerateur / fact);
  }

  return (resultat);
}
```

La complexité de ce programme est $O(n)$. (Tous les programmes que vous avez écrit étaient en $O(n^2)$...)

(4) *Question 3.* Le résidu d'un nombre n en base b est la somme itérée de ses chiffres, quand il est écrit en base b.

Par exemple, le résidu de 13762 en base 10 est 1 car
- $1+3+7+6+2=19$,
- $1+9=10$,
- $1+0=1$.

Écrivez un petit programme complet (avec la fonction `main`), dans une syntaxe aussi proche du C que possible, qui demande un entier n et un entier b à l'utilisateur ; et qui affiche le résidu de n en base b.

N'oubliez pas de commenter votre programme et de justifier ce que vous faites...

Pouvez-vous estimer la complexité de votre programme en fonction de n et de b ?

> *Remarque :* avec la bibliothèque "`math.h`", vous pouvez utiliser toutes les fonctions mathématiques habituelles (exponentielle, logarithme, etc.)

Solution : La solution la plus simple est la suivante :

```
int residuSimple(int n, int b)
{
  int residu = n % (b - 1);
  if (residu == 0) {
    residu = b - 1;
  }
  return (residu);
}
```

(Si si, ça marche...)

On peut faire une version assez simple en la remarque suivante : le résidu de 13762, c'est la même chose que le résidu de $1376+2$, càd, en base b, le résidu de n, c'est la même chose que le résidu de $\lfloor \frac{n}{b} \rfloor + (n \bmod b)$.

```
int residuMoinsSimple(int n, int b)
{
  int residu = n;
   while (residu >= b) {
    residu = (residu / b) + (residu % b);
  }
```

```
  return (residu);
}
```

Finalement, si on veut faire comme dans l'énoncé, on commence par définir une fonction qui fait une étape du résidu en ajoutant tous les chiffre de n. Comme le chiffre de droite en base b est $n \bmod b$ et que les chiffres de gauche (sans le chiffre de droite) forment le nombre $\lfloor \frac{s}{b} \rfloor$, cette fonction peut s'écrire :

```
int residuUneEtape(int n, int b)
{
  int residuPartiel = 0;
  while (n > 0) {
    residuPartiel = residuPartiel + (n % b);
    n = n / b;
  }
  return (residuPartiel);
}
```

La fonction finale devient donc :

```
int residuEtapes(int n, int b)
{
  int residu = n;
   while (residu >= b) {
    residu = residuUneEtape(residu, b);
  }
  return (residu);
}
```

Voici un exemple de fonction principale pour tester tout ça :

```
int main()
{
  int n, b;
  printf("Entrez un nombre : \n  ");
  scanf("%i", &n);

  printf("Entrez une base : \n  ");
  scanf("%i", &b);

  printf("Calcul du résidu de %i en base %i :\n", n, b);
  printf("  - méthode simple : %i\n", residuSimple(n, b));
  printf("  - méthode moins simple : %i\n", residuSimple(n, b));
  printf("  - méthode par étapes : %i\n", residuSimple(n, b));

  return (0);
}
```