Université de Nice-Sophia Antipolis — POLYTECH
ELEC3 — 2016–2017

Examen de Harmonisation Info

Durée : 2h
Aucun document autorisé
Mobiles interdits Attention : de bien respecter les consignes données dans les questions

▶ 1. Qu'appelle-t-on « langage machine » ?

Question sur 1 pt : le langage machine d'un ordinateur est le seul langage que connaît un ordinateur. Il permet de coder, en notation binaire, les instructions des programmes que pourra exécuter l'ordinateur.

. . . . . .

▶ 2. Qu'appelle-t-on langage procédural ? Citez au moins deux langages de programmation procéduraux.

Question sur 2 pts : Les langage C ou Pascal sont des langages procéduraux, c'est-à-dire qu'un programme écrit dans ces langages est structuré autour des actions représentées par des procédures (ou des fonctions). En revanche, le langage Java est un langage à objets c'est-à-dire qu'un programme écrit dans ce langage est structuré autour des objets, représentés par des classes.

. . . . . .

▶ 3. Expliquez la différence entre sémantique statique et sémantique dynamique.

Question sur 2 pts : Un compilateur est un programme qui traduit un programme source écrit dans un langage de haut niveau (i.e. C) en un programme cible sémantiquement équivalent écrit en langage machine. Pour que la traduction puisse se faire, le programme source doit respecter les règles lexicales, syntaxiques et sémantiques du langage source. Le compilateur vérifie que le programme source respecte bien toutes ces règles.

. . . . . .

▶ 4. Expliquez comment on peut garantir la validité (la justesse) d'un programme.

Question sur 2 pts : La validité d'un programme ne peut se faire qu'analytiquement. L'axiomatique de Hoare, habituellement utilisée, est celle des affirmations, antécédent et conséquent, qui décrivent l'état du programme. L'antécédent et le conséquent doivent être vrais avant et après l'exécution d'un énoncé. Pour chaque énoncé d'un langage de programmation, il existe une règle de déduction qui permet de passer systématiquement de son antécédent à son conséquent (et réciproquement). L'application des règles à l'ensemble des énoncés du programme permet de vérifier la validité du programme.

On rappelle que tester l'exécution d'un programme ne peut que mettre en évidence des erreurs et jamais garantir la justesse à 100 % du programme.

. . . . . .

▶ 5. En quoi l'utilisation du type réel des langages de programmation peut poser des problèmes ?

Question sur 2 pts : En informatique, à cause de la représentation discontinue des réels, l'arithmétique réelles est inexacte, et les programmes doivent en tenir compte.

. . . . . .

▶ 6. À quelle valeur décimale correspond la configuration binaire (en complément à 2 sur 8 bits) suivante : 11011101.

-35

. . . . . .

▶ 7. Sur 8 bits, donnez les représentations binaires de 65, 127, −1, 128 et −128. Les nombres négatifs seront représentés en complément à 2.

Question sur 2 pts :

$$\begin{aligned} 65 &= 2^6 + 2^0 &= 01000001 \\ 127 &= 2^8 - 1 &= 01111111 \\ -1 & &= 11111111 \\ -128 & &= 10000000 \end{aligned}$$

sur 8 bits, 128 n'a pas de représentation, puisque l'intervalle de valeurs en complément à 2 est $[-2^{8-1}; +2^{8-1} - 1]$, c'est-à-dire $[-128; +127]$.

.....

▶ 8. Qu'est-ce que le type booléen ? Et comment est-il représenté en C ?

Question sur 2 pts : Le type booléen est un ensemble à deux valeurs $\{faux, vrai\}$. En C, il n'y a pas de type prédéfini booléen, c'est est un sous-type du type entier, avec comme convention $0 = faux$ et $\neq 0 = vrai$.

.....

▶ 9. L'appel de procédure `écrire(x)` permet d'écrire la valeur de `x` sur la sortie standard. Dans cet appel, `x` est-il un paramètre « donnée » ou « résultat » ?

Question sur 1 pt : `x` est un paramètre donnée, il désignera la donnée à écrire.

.....

▶ 10. Dans le fragment de code suivant, sous quelles conditions l'énoncé E3 est-il exécuté ?

```
si x=1 et y=2 alors E1
   sinon
        si x=2 ou z=4 alors E2
           sinon E3
        finsi
finsi
```

Question sur 1 pt :

```
si x=1 alors y ← 2
sinon { x≠1 }
   si x=2 alors y ← 3
   sinon { x≠1 et x≠2 }
         y ← 4
   finsi
finsi
```

.....

▶ 11. Expliquez à quoi correspond l'invariant de boucle de l'énoncé **tantque**.

Question sur 2 pts : Un invariant est une affirmation (donc vraie) vérifiée quel que soit le nombre d'itérations effectué par l'énoncé itératif. On appelle Invariant (avec un grand I) de boucle, l'affirmation située juste avant le prédicat d'achèvement ; il représente la sémantique de l'énoncé et doit être défini avant

l'écriture de l'énoncé.

Dans le cas de l'énoncé **tantque**, l'Invariant doit être vérifié avant et après l'énoncé.

.....

▶ 12. Ajoutez à l'algorithme suivant les affirmations qui prouvent sa validité, et expliquez à quoi correspond la valeur `d` calculée.

```
variables a, b, c, d : entiers;

lire(a,b,c)
{ ..................................... }
si a<b alors
    si b⩽c alors
        { ..................................... }
        d ← b
    sinon
        { ..................................... }
        si c>a alors
            { ..................................... }
            d ← c
        sinon
            { ..................................... }
            d ← a
        finsi
    finsi
sinon
    { ..................................... }
    si c>a alors
        { ..................................... }
        d ← a
    sinon
        { ..................................... }
        si c>b alors
            { ..................................... }
            d ← c
        sinon
            { ..................................... }
            d ← b
        finsi
    finsi
finsi
{ ..................................... }
écrire(d)
```

Question sur 2 pts :

```
variables a, b, c, d : entiers;

lire(a,b,c)
```

```
{ a, b et c désignent 3 entiers quelconques }
si a<b alors
    si b⩽c alors
        { a < b ⩽ c }
        d ← b
    sinon
        { a < b et c < b }
        si c>a alors
            { a < c < }
            d ← c
        sinon
            { c ⩽ a < b }
            d ← a
        finsi
    finsi
sinon
    { a ⩾ b }
    si c>a alors
        { b ⩽ a < c }
        d ← a
    sinon
        { a ⩾ b et c ⩽ a }
        si c>b alors
            { b < c ⩽ a }
            d ← c
        sinon
            { c ⩽ b ⩽ a }
            d ← b
        finsi
    finsi
finsi
{ d désigne la médiane des valeurs a, b et c }
écrire(d)
```

.....

▶ 13. En vous inspirant de l'algorithme (vu en cours) du produit de 2 entiers naturels x et y qui tient compte de la parité de y, écrivez l'algorithme qui calcule x^y ($x \times x ... \times x$, y fois), toujours en tenant compte de la parité de y. Vous prendrez soin d'indiquer et de prouver les invariants des boucles ainsi que leurs finitudes.

Question sur 4 pts :

```
{ Antécédent: x⩾0, y⩾0, x = a, y = b }
{ Conséquent: puissance(x,y) = x^y = a^b }
p ← 1;
{ a^b = p * x^y }
tantque y>0 faire
    { a^b = p * x^y et y>0 }
    tantque pair(y) faire
        { a^b = p * x^y et y = (y / 2) * 2 > 0 }
        y ← y / 2
        { a^b = p * x^(2*y) }
        x ← x * x
        { a^b = p * x^y }
    fintantque
    { a^b = p * x^y + y et y>0 et y impair }
    p ← p * x
    { a^b = p * x^(y-1) et y impair }
    y ← y - 1
    { a^b = p * x^y }
fintantque
{ y = 0 et a^b = x^y = p }
```


.....

▶ 14. Écrivez en C, l'algorithme précédent.

Question sur 2 pts :

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

int main(void) {
  int p, x, y;
  scanf("%d %d", &x, &y);
  assert(x>=0 && y>=0);
  /* soit x = a, et y = b */
  p = 0;
  /* a^b = p * x^y */
  while (y>0) {
    /* a^b = p * x^y et y>0 */
    while  ((y & 1) == 0) {
      /* a^b = p * x^y et y = (y / 2) * 2 > 0 */
      y >>= 1;
      x <<= 1;
    }
    /*  a^b = p * x^y + y et y>0 et y impair */
    p *= x;
    y--;
    /* a^b = p * x^y */
  }
  /* y = 0 et a^b = x^y = p */
  printf("x^y = %d\n", p);
  return EXIT_SUCCESS;
}
```

.....