Tous les documents papier et tous les dispositifs électroniques sont interdits. Le barème est donné à titre indicatif.

Les règles de la logique de Hoare et la syntaxe de la logique du langage micro-C sont rappelées dans l'annexe à la fin de ce document.

Exercice 1 : logique de Hoare (*5 points*)

Pour chacun des triplets de Hoare suivants, indiquer s'il est valide ou non. Si le triplet n'est pas valide, donner un état qui satisfait la précondition mais ne satisfait pas la postcondition. Si le triplet est valide, en donner une démonstration en utilisant les règles de la logique de Hoare.

Question 1 : $\{x = -y \wedge x \neq n\}\ x := x + 1;\ y := y - 1\ \{x = -y\}$

Question 2 : $\{z = 0\}$ while $z = z$ do $z := z + 1$ od $\{z = 2\}$

Exercice 2 : division euclidienne par soustractions (*9 points*)

Dans cet exercice, on considère une logique de Hoare dont toutes les constantes et toutes les variables sont des entiers relatifs (nombres positifs ou négatifs), et qui admet une addition, une multiplication et des comparaisons ($\leq$, $<$, $\geq$, $>$ et $=$) entre deux entiers relatifs.

Question 1 : Exprimer par un triplet de Hoare que le programme

$$q := 0;\ r := a;\ \text{while } r \geq b \text{ do } q := q + 1;\ r := r - b \text{ od}$$

calcule dans q et r le quotient et le reste de la division euclidienne de a par b, pour tous les entiers naturels a et pour tout entier naturel b non nul. La postcondition de ce triplet doit relier a, b, q et r par une égalité et fixer les bornes les plus précises possibles pour r.

Question 2 : Dans un tableau, appliquer à ce triplet de Hoare les règles (*sequence*), (*assignment*) et (*precondition*) de la logique de Hoare, jusqu'à ce qu'il ne reste plus qu'à démontrer un triplet de la forme $\{\dots\}$ while ... do ... od $\{\dots\}$.

Question 3 : Proposer un invariant pour la boucle while, assez précis pour permettre de démontrer la validité de ce triplet de Hoare.

Question 4 : Donner les trois prémisses qu'on obtient en appliquant les règles (*consequence*) et (*while*), avec cet invariant, au triplet de la forme $\{\dots\}$ while ... do ... od $\{\dots\}$ obtenu précédemment.

Question 5 : En appliquant une par une les règles de la logique de Hoare, démontrer la deuxième de ces prémisses, qui est le triplet de Hoare qui exprime que l'invariant est préservé par le corps de la boucle.

Exercice 3 : inversion d'un tableau dans en autre tableau, en micro-C

(6 points)

Question 1 : Reproduire et compléter le code suivant pour que son exécution stocke les éléments du tableau `tab`, supposé de taille `n`, dans le tableau `out`, mais dans l'ordre inverse, sans modifier le contenu du tableau `tab`.

Le tableau `out` est supposé disjoint du tableau `tab`, alloué avant cette exécution, et de taille suffisante. Par exemple, si le tableau `tab` est | 4 | 1 | 3 | 3 |, alors, après l'exécution de l'instruction

```
inversion(tab,out);
```

le début du tableau `out` doit être | 3 | 3 | 1 | 4 |.

```
void inversion (int tab[], int n, int out[]) {
  /*@ requires length(tab) == n;

  */
  int i = 0;
  while (i < ..) {
    /*@

    */
    out[..] = tab[n-1-i];
    i = ..;
  }
}
```

Question 2 : Donner en syntaxe micro-C une précondition pour que le tableau `out` ait une taille suffisante pour contenir les éléments du tableau `tab` dans l'ordre inverse.

Question 3 : Donner en syntaxe micro-C une postcondition qui exprime que le tableau `out` contient les éléments du tableau `tab` dans l'ordre inverse.

Question 4 : Proposer en syntaxe micro-C un invariant de boucle suffisant pour démontrer cette postcondition.

Question 5 : Proposer en syntaxe micro-C un variant de boucle permettant de démontrer la terminaison de la boucle et de la fonction.

Annexe : rappels de cours

Logique de Hoare

La logique de Hoare est le système formel <L, R> où

— L est l'ensemble des formules du premier ordre et des formules de la forme $\{P\}\ c\ \{Q\}$, appelées triplets de Hoare, où P et Q sont des formules de la logique du premier ordre et c est un programme du mini-langage défini par la grammaire

$c, c' ::=$	$\mid x := e$	affectation
	$\mid c;\ c'$	séquence
	$\mid$ if b then c fi	conditionnelle simple
	$\mid$ if b then c else c' fi	conditionnelle double
	$\mid$ while b do c od	répétition

— R est l'ensemble de règles de déduction de la figure 1.

$$\frac{}{\{P\ (e/x)\}\ x := e\ \{P\}}\ (\textit{assignment})$$

$$\frac{\{P\}\ c\ \{Q\} \qquad \{Q\}\ c'\ \{R\}}{\{P\}\ c; c'\ \{R\}}\ (\textit{sequence})$$

$$\frac{\{b \wedge P\}\ c\ \{Q\} \qquad (\neg b \wedge P) \Rightarrow Q}{\{P\}\ \text{if } b \text{ then } c \text{ fi}\ \{Q\}}\ (\textit{if-then})$$

$$\frac{\{b \wedge P\}\ c_1\ \{Q\} \qquad \{\neg b \wedge P\}\ c_2\ \{Q\}}{\{P\}\ \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ fi}\ \{Q\}}\ (\textit{if-then-else})$$

$$\frac{\{b \wedge P\}\ c\ \{P\}}{\{P\}\ \text{while } b \text{ do } c \text{ od}\ \{\neg b \wedge P\}}\ (\textit{while})$$

$$\frac{P \Rightarrow P' \qquad \{P'\}\ c\ \{Q'\} \qquad Q' \Rightarrow Q}{\{P\}\ c\ \{Q\}}\ (\textit{consequence})$$

$$\frac{P \Rightarrow P' \qquad \{P'\}\ c\ \{Q\}}{\{P\}\ c\ \{Q\}}\ (\textit{precondition})$$

$$\frac{\{P\}\ c\ \{Q'\} \qquad Q' \Rightarrow Q}{\{P\}\ c\ \{Q\}}\ (\textit{postcondition})$$

FIGURE 1 – Règles de la logique de Hoare.

Syntaxe de la logique du langage micro-C

La logique du langage micro-C doit être écrite dans un fichier C, soit sur une ligne qui commence par `//@`, soit dans un bloc de commentaires qui commence par `/*@` et se termine par `*/`.

Les types de micro-C sont les types C suivants :

— Le type `void`.

— Le type `int` des entiers relatifs des mathématiques, avec le symbole fonctionnel unaire `-` (préfixé) et les symboles fonctionnels binaires infixés `+`, `-`, `*`, `/` et `%`, respectivement pour l'opposé, l'addition, la soustraction, la multiplication, le quotient et le reste de la division euclidienne.

— Le type `int ..[]` des tableaux d'entiers.

La logique propositionnelle de micro-C contient les formules `true` et `false` pour "vrai" et "faux". Les connecteurs propositionnels de la négation, du "et", du "ou", de l'implication et de l'équivalence sont respectivement `!`, `&&`, `||` (comme en C), `->` et `<->`. Dans nos listings, ces symboles apparaissent sous la forme plus lisible !, && , || , -> et <-> .

La quantification universellle $\forall x$... s'écrit `forall x. ..` si `x` est un entier et `forall x[]. ..` si `x` est un tableau d'entiers. De même pour la quantification existentielle. Le quantificateur existentiel $\exists$ s'écrit `exists`. Dans nos listings, ces symboles apparaissent sous la forme plus lisible $\forall$ et $\exists$.

Le prédicat d'égalité infixé (`==`) est prédéfini sur les types des entiers et des tableaux d'entiers.

Sur le type `int`, les symboles relationnels binaires (infixés) `!=`, `<`, `>`, `<=` et `>=` sont aussi définis. Ils correspondent respectivement à la négation de l'égalité et aux relations d'ordre strictes et larges sur les entiers. Ces symboles sont notés $\neq$, $<$, $>$, $<=$ et $>=$ dans nos listings.

Dans la partie logique du langage micro-C le terme `length(a)` désigne la longueur du tableau d'entiers `a`. Comme en C, le terme `a[i]` désigne l'élément du tableau `a` à l'indice `i`, si cet indice est entre `0` et `length(a)-1` inclus.

En micro-C, la ligne

```
predicate p();
```

déclare une proposition atomique p, aussi appelée "variable propositionnelle".

La ligne

```
predicate p(int i, int t[]);
```

déclare un prédicat binaire p dont le premier argument est un entier et le second argument est un tableau d'entiers.

La ligne

```
function T f(int i, int t[]);
```

déclare une fonction binaire f dont le premier argument est un entier, le second argument est un tableau d'entiers, et qui calcule une donnée de type `T`, qui peut être `void` ou `int`.

Les lignes

```
predicate p(int i, int t[]) = ..;
```

et

```
function T f(int i, int t[]) = ..;
```

déclarent un prédicat p et une fonction similaires, tout en donnant une définition à ce prédicat ou à cette fonction. Pour le prédicat, il faut remplacer les pointillés par une formule micro-C. Pour la fonction, il faut remplacer les pointillés par une expression de type `T`.

La ligne

```
lemma l: ..
```

déclare un lemme nommé `l`. Les pointillés doivent être remplacés par une formule micro-C.

Pour décrire en micro-C la valeur d'une expression **avant** exécution d'une fonction, il faut entourer cette expression par `old(` et `)`, dans toute postcondition ou invariant d'une boucle de cette fonction.