Cours Modélisation et vérification des systèmes informatiques
Exercices
Utilisation d'un environnement de vérification Frama-c (II)
par Dominique Méry
11 décembre 2023

Exercice 1 *Définir une fonction max avec son contrat.*

```
int max ( int x, int y ) {
  if ( x >=y ) return x ;
  return y ; }
```

Exercice 2 *Définir une fonction maxpointer calculant la valeur du maximum du ciontenu de deux adresses avec son contrat.*

```
int max_ptr ( int *p, int *q ) {
if ( *p >= *q ) return *p ;
return *q ; }
```

Exercice 3 *Définir une fonction abs calculant la valeur absolue d'un nombre entier avec son contrat.*

```
#include <limits.h>
int abs (int x) {
  if (x >= 0) return x ;
  return -x;}
```

Exercice 4 *Etudier les fonctions pour la vérification de l'appel de abs et max.*

```
int abs ( int x );
int max ( int x, int y );
// returns maximum of absolute values of x and y
int max_abs( int x, int y ) {
x=abs(x); y=abs(y);
return max(x,y);
}
```

Exercice 5 Question 5.1

```
/*@ requires a >= 0 && b >= 0;
  ensures 0 <= \result;
  ensures \result < b;
  ensures \exists integer k; a == k * b + \result;
*/
int rem(int a, int b) {
  int r = a;
  int q=0;
  /*@
    loop invariant
    a == q * b + r &&
    r >= 0
    ;
```

```
    loop assigns r;
   */
  while (r >= b) {
    r = r - b;
    q = q+1;
  };
  return r;
}
```

Question 5.2

```
/*@ axiomatic Fact {
  @ logic integer Fact(integer n);
  @ axiom Fact_1: Fact(1) == 1;
  @ axiom Fact_rec: \forall integer n; n > 1 ==> Fact(n) == n * Fact(n-1);
  @ } */

/*@ requires n > 0;
  ensures \result == Fact(n);
*/
int fact(int n) {
  int y = 1;
  int x = n;
  /*@ loop invariant x >= 1 &&
                     Fact(n) == y * Fact(x);
    loop assigns x, y;
   */
  while (x != 1) {
    y = y * x;
    x = x - 1;
  };
  return y;
```

Question 5.3

```
/*@  ensures \result >= a;
  ensures \result >= b;
  ensures \result == a || \result == b;
*/
int max (int a, int b) {
  if (a >= b) return a;
  else return b;
}

/*@
  requires n > 0;
  requires \valid(t+(0..n-1));
  ensures 0 <= \result < n;
  ensures \forall int k; 0 <= k < n ==>
    t[k] <= t[\result];
*/
int indice_max (int t[], int n) {
  int r = 0;
  /*@ loop invariant 0 <= r < i <= n
    && (\forall int k; 0 <= k < i ==> t[k] <= t[r])
    ;
```

```
    loop assigns i, r;
  */
  for (int i = 1; i < n; i++)
    if (t[i] > t[r]) r = i;
  return r;
}


/*@
  requires n > 0;
  requires \valid(t+(0..n-1));
  ensures \forall int k; 0 <= k < n ==>
    t[k] <= \result;
  ensures \exists int k; 0 <= k < n && t[k] == \result;
*/
int valeur_max (int t[], int n) {
  int r = t[0];
  /*@ loop invariant 0 <= i <= n
    && (\forall int k; 0 <= k < i ==> t[k] <= r)
    && (\exists int k; 0 <= k < i && t[k] == r)
    ;
    loop assigns i, r;
  */
  for (int i = 1; i < n; i++)
    if (t[i] > r) r = t[i];
  return r;
}
```