AULA 1 - CORREÇÃO DE UM ALGORITMO

Especificação

- **Pré-condição:** propriedade que se assume como verdadeira no estado inicial da execução do programa, isto é, só interessa considerar as execuções do programa que satisfaçam esta condição
- **Pós-condição:** propriedade que se deseja provar verdadeira no estado final da execução do programa

Triplos de Hoare

O triplo $\{P\}\ C\ \{Q\}$ é válido quando todas as execuções de C, partindo de estados que satisfaçam P, caso terminem, resultem num estado final do programa que satisfaz Q.

Exemplos

1. Swap: programa que troca os valores das variáveis x e y

   pré: $x = x_0 \wedge y = y_0$

   pós: $x = y_0 \wedge y = x_0$

2. produto: programa que calcula o produto de dois inteiros

   pré: $x = x_0 \wedge y = y_0$

   pós: $m = x_0 * y_0$

3. mod: programa que coloca em m o resto da divisão inteira entre os valores iniciais das variáveis x e y

   pré: $x > 0 \wedge y > 0$ (1 tem de ser apenas >0 pois não existe divisão por 0)

   pós: $0 \leq m < y \wedge \exists_{d \geq 0}\ d * y + m = x$

4. div: programa que coloca em d o resultado da divisão inteira entre os valores iniciais das variáveis x e y

   pré: $x > 0 \wedge y > 0$

   pós: $d \geq 0 \wedge \exists_{0 \leq m < y}\ d * y + m = x$

5. divmod: programa que coloca em d o resultado da divisão inteira e em m o resto dessa divisão

   pré: $x > 0 \wedge y > 0$

   pós: $0 \leq m < y \wedge d \geq 0 \wedge d * y + m = x$

6. procura: programa que procura um dado valor (x) num vetor ordenado

   pré: $(\forall_{a \leq i \leq b} .\ v[i] = v_i) \wedge (\forall_{a \leq i < b} .\ v_i \leq v_{i+1})$

   pós: $(\forall_{a \leq i \leq b} .\ v[i] = v_i) \wedge ((\exists_{a \leq i \leq b} .\ v_i = x) \Rightarrow v[p] = x)$

EXERCÍCIOS AULA 1

Especificação

Exercício 1 Descreva por palavras as seguintes especificações:

1. **Pré-condição:** $x = x_0 \geq 0 \wedge e = e_0 > 0$

   Pós-condição: $\mid r * r - x_0 \mid < e_0$

2. **Pré-condição:** $\forall_{0 \leq i < N}\ A[i] \ = a_i$

   Pós-condição: $\forall_{0 \leq i < N}\ (A[i] = a_i \wedge A[p] \leq a_i)$

Exercício 2 Escreva especificações (pré e pós condições) para os seguintes problemas:

1. Um programa que coloca na variável `r` o mínimo múltiplo comum das variáveis `X` e `Y`.
2. Um programa que recebe dois arrays `A` e `B` como parâmetros, e verifica se eles têm um elemento em comum.
3. Um programa que recebe dois arrays `A` e `B` como parâmetros, e calcula o comprimento do prefixo mais longo que os dois têm em comum.

1. pre: $x > 0$ && $y > 0$
   pós:

2. pre: $N_A > 0$ && $N_B > 0$
   pós:

3. pre: $N_A > 0$ && $N_B > 0$

Ficha 1

1. a)

```
int fa (int x, int y){
   // pre: True
   ...
   // pos: (m == x || m == y) && (m >= x && m >= y)
   return m;
}
```

Calculo o máximo entre x e y.

1.b)

```
int fb (int x, int y){
   // pre: x >= 0 && y >= 0
   ...
   // pos: x % r == 0 && y % r == 0
   return r;
}
```

calcula um divisor comum de x e y

1.c)

```
int fc (int x, int y){
   // pre: x > 0 && y > 0
   ...
   // pos: r % x == 0 && r % y == 0
   return r;
}
```

calcula um múltiplo comum de x e y

1.d)

```
int fd (int a[], int N){
   // pre: N>0
   ...
   // pos: 0 <= p< N && forall_{0 <= i< N} a[p] <= a[i]
   return p;
}
```

calcula um índice do elemento mínimo de um array

1.e)

```
int fe (int a[], int N){
   // pre: N>0
   ...
   // pos: forall_{0 <= i< N} x <= a[i]
   return x;
}
```

calcula um valor que pode ser igual ou menor que os elementos do array

1.f)

```
int ff (int a[], int N){
   // pre: N>0
   ...
   // pos: (forall_{0 <= i< N} x <= a[i]) &&
   //      (exists_{0 <= i <N} x == a[i])
   return x;
}
```

→ permite que x seja igual ao mínimo

calcula o valor mínimo de um array

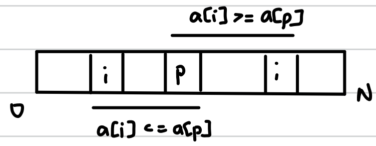
1.g)

```
int fg (int x, int a[], int N){
   // pre: N>=0
   ...
   // pos: (p == -1 && forall_{0 <= i < N} a[i] != x) ||
   //       ( (0 <= p < N ) && x == a[p])
   return p;
}
```

calcula um índice do array com o valor igual a x. Se não existir retorna -1.

1.h)

```
int fh (int a[], int N){
   // pre: N>0
   ...
   // pos: (p == -1) ||
            ((0 <= p < N) &&
   //        (forall_{0 <= i < p} a[i] <= a[p]) &&
   //        (forall_{p <  i < N} a[i] >= a[p]))
   return p;
}
```



calcula o índice no qual existe uma partição de valores sequenciais no array

2.a) A função `int prod (int x, int y)` que calcula o produto de dois inteiros.

```
int prod (int x, int y)
{
    // pre: True
    ...
    // pos: p = x * y
    return p;
}
```

2.b) A função `int mdc (int x, int y)` que calcula o maior divisor comum de dois números inteiros positivos.

```
int mdc (int x, int y)
{
    // Pre: x >= 0 && y >= 0 && (x != 0 || y != 0)
    ...
    // Pos: x % r = 0 && y % r == 0 && forall{r'. x % r == 0 && y % r == 0
                                               → r >= r'}
    return r;
}
```

2.c) A função `int sum (int v[], int N)` que calcula a soma dos elementos de um array.

```
int sum (int v[], int N)
{
    // pre : N >= 0
    ...
    // pos : r = Σ_{i=1}^{N-1} v[i]
    return r;
}
```

2.d) A função `int maximo (int v[], int n)` que calcula o maior elemento de um array.

```
int maximo (int v[], int n)
{
    // pre: n>0
    ..
    // pos: forall {0 ≤ i < n . r >= v[i]} && exists {0 ≤ i < n . r == v[i]}
    return r;
}
```

2.e) A função `int maxPOrd (int v[], int N)` que calcula o comprimento do maior prefixo ordenado de um array.

```
int maxPOrd (int v[], int N)
{
    // pre : N >= 0
    ...
    // pos: 0 ≤ r ≤ N && N == 0 → r == 0 && forall {0<i<r} v[i-1] <=
            v[i] && r == N || v[r] < v[r-1]
```

2.f) A função `int isSorted (int v[], int N)` que testa se um array está ordenado por ordem crescente.

```
int isSorted (int v[], int N)
{
    // pre : N >= 0
    ...
    // pos: ∀ 0 ≤ i, j < N   i<j → v[i] ≤ v[j]
```

Questão 2 [3 valores]
Considere o cálculo de máximos de um array de comprimento N por uma *sliding window* (janela deslizante) de comprimento k. A ideia é calcular os máximos de todas as subsequências (contíguas) de comprimento k, guardando-os num array de resultados.
Por exemplo com N=5, k=3, o resultado para o array [50, 10, 30, 20, 0] seria [50, 30, 30], correspondente aos máximos de [50, 10, 30], [10, 30, 20], e [30, 20, 0].
A função seguinte resolve recursivamente este problema, utilizando como auxiliar a função `maxarray` da Questão 1.

```c
void MaxWindowRec(int v[], int r[], int N, int k) {
  if (N >= k) {
    r[0] = v[maxarray(v, 0, k-1)];
    MaxWindowRec(v+1, r+1, N-1, k);
  }
}
```

Escreva uma especificação (**pré- e pós-condição**) para esta função.

pre: $0 < K \leq N$
Pos $\forall i.\ 0 \leq i \leq N-K \rightarrow (r[i] \in v[i..i+k-1] \wedge \forall j.\ i \leq j < i+K \rightarrow v[j] \leq r[i])$
ou $\forall i\ 0 \leq i \leq N-K \rightarrow r[i] = v[maxarray(v,i,i+K-1)$

Considere o seguinte programa parcialmente anotado:

```c
// 0 <= a <= b && forall (a <= i <= b) v[i] = v_i
j=a; r=0;
while (j<=b) {
  r = r+v[j]; j=j+1;
}
// (forall (a <= i <= b) v[i] = v_i) && r = sum (a <= i <= b) v_i
```

(a) Descreva informalmente (i.e., sucintamente e por palavras) a especificação deste programa.

Programa que calcula a soma dos valores do array no intervalo de a a b

AULA 2 - INVARIANTES E VARIANTES

Invariante

Propriedade verdadeira em todas as iterações
- Inicialização: I é verdade à entrada do ciclo
- Preservação: I é verdade no início de uma iteração, então também é verdade no final dessa iteração
- Utilidade: I juntamente com a negação da condição do ciclo, implica a verdade da pós-condição

Exemplos

1. Divisão Inteira: programa que calcula divisão de x por y, colocando o quociente em q e o resto em r

```
int divide (int x, int y) {
  // Pre: x >= 0 && y > 0

  // Pos: 0 <= r < y && q*y+r == x
  return q
}
```

triplo de Hoare:
$\{x \geq 0 \wedge y > 0\}\ C\ \{0 \leq r < y \wedge q*y+r = x\}$

Considerando, p.e., $x = 14$ e $y = 3$:

r	q	q*y+r
14	0	14
11	1	14
8	2	14
5	3	14
2	4	14

$I \equiv 0 \leq r \wedge q*y+r = x$

A pós-condição equivale à conjunção do invariante e da negação da condição do ciclo

2. Fatorial: calcula de forma iterativa o fatorial de um número natural

```
int fact (int n) {
  // Pre: n >= 0
  f = 1;
  i = 1;
  while (i<=n) {
    f = f*i;
    i = i+1;
  }
  // Pos: f = n!
  return f;
}
```

Para mostrar que é correta corresponde a provar a validade do triplo de Hoare:
$$\{n \geq 0\}\ F\ \{f = n!\}$$

- inicialização: $\{n \geq 0\}\ f = 1; i = 1\ \{I\}$
- preservação: $\{I \wedge i \leq n\}\ f = f*i; i = i+1;\ \{I\}$
- utilidade: $\{I \wedge i > n\} \rightarrow f = n!$

$I \equiv f = (i-1)! \wedge i \leq n+1$

1º substituir c/ valores da inicialização: $n \geq 0 \rightarrow f = 0! = 1$

2º $f*i = (i+1-1)! \wedge i+1 \leq n+1$
$\equiv f*i = i! \wedge i \leq n$

3º Se $f = (i-1)!$ então $f*i = i!$
Se $i \leq n+1$ e $\neg(i \leq n)$, então $i \leq n$

4º $i \leq n$ é falsa, o valor final é $i = n+1$ e o invariante implica $f = n!$

- correção parcial: implica provar que o programa termina sempre - basta mostrar que se terminar então a pós-condição é satisfeita

Variante

correção total: implica provar adicionalmente que o programa termina sempre.
– basta, para cada ciclo do programa, utilizar uma medida de distância entre o estado atual e o estado de terminação

- quando a execução numa iteração (a condição é verdadeira), o variante é positivo
- as iterações fazem decrescer (estritamente <) o valor do variante

Exemplos:

1. divide: $r-y$ é candidato a variante pois diminui em todas as iterações porque y não altera e r decresce sempre

No entanto, $r-y>0$ não é verdade sempre que é executada uma iteração uma vez que a condição $y \leq r$ só garante que $r-y \geq 0$.

Assim, o variante é $r-y+1$, sempre positivo nas iterações.

2. fatorial: $n-i+1$ é um variante do ciclo.
   - n mantém-se constante e i é incrementado em todas as iterações
   - é garantido por $i \leq n$ que $n-i+1>0$ à entrada das iterações

2. sum: o variante será $n-i$

```
while (i < n)  {
    result = vector[i] + result;
    i = i+1;
}
```

EXERCÍCIOS AULA 2

Ficha 1

CORREÇÃO

1. Para cada um dos seguintes triplos de Hoare, apresente um contra-exemplo que mostre a sua **não** validade.

(a)
$\{True\}$
```
r = x+y;
```
$\{r \geq x\}$

Não válido. $y = -2$, $x = 17$; $r = 15$ e $r < x$

(b)
$\{True\}$
```
x = x+y; y = x-y; x = x-y;
```
$\{x == y\}$

$x = x+y$;
$y = x+y-y \Leftrightarrow y = x$;
$x = x-x = 0$

Não válido $x = 2$ $y = 5$
$x = 7$; $y = -3$, $x = 4$

(c)
$\{True\}$
```
x = x+y; y = x-y; x = x-y;
```
$\{x \neq y\}$

Não válido $x = 0 \wedge y = 0$

(d)
$\{True\}$
```
if (x>y) r = x-y; else r = y-x;
```
$\{r > 0\}$

Válido $x - y > 0$ logo $r > 0$

(e)
$\{True\}$
```
while (x>0) { y=y+1; x = x-1;}
```
$\{y > x\}$

Não Válido $y = -17$, $x = 16$ $\rightarrow$ $y = -1$, $x = 0$

2. Modifique a pré-condição de cada um dos triplos de Hoare da alínea anterior de forma a obter um triplo válido.

1.a) $\{x \geq 0 \wedge y \geq 0\}$
$r = x+y$;
$\{r \geq x\}$

1.b) $\{x == 0 \wedge y == 0\}$
$x = x+y$; $y = x-y$; $x = x-y$;
$\{x == y\}$

1.c) $\{x != 0 \wedge y != 0\}$

$x = x+y;\ y = x-y;\ x = x-y;$

$\{x \neq y\}$

1.e) $\{x > 0 \wedge y > 0\}$

while $(x > 0)$ $\{y = y+1;\ x = x-1;\}$

$\{y > x\}$

INVARIANTES

1. Considere as seguintes implementações de uma função que calcula o produto de dois números.

```
int mult1 (int x, int y){
   // pre: x>=0
   int a, b, r;
   a=x; b=y; r=0;
   while (a>0){
      r = r+b;
      a = a-1;
   }
   // pos: r == x * y
   return r;
}
```

```
int mult2 (int x, int y){
   // pre: x>=0
   int a, b, r;
   a=x; b=y; r=0;
   while (a>0) {
      if (a%2 == 1) r = r+b;
      a=a/2; b=b*2;
   }
   // pos: r == x * y
   return r;
}
```

✓falta

(a) Para cada um dos predicados, indique se são verdadeiros no início (Init) e preservados pelos ciclos destas duas funções.

Predicado	mult1		mult2	
	Init	Pres	Init	Pres
$r == a * b$				
$a \geq 0$				
$b \geq 0$				
$r \geq 0$				
$a == x$				
$b == y$				
$a * b == x * y$				
$a * b + r == x * y$				

(b) Apresente invariantes dos ciclos destas duas funções que lhe permitam provar a sua correcção (parcial).

$I_{mult1} \equiv (x-a)y$ $\qquad\qquad$ $I_{mult2} \equiv ab + r = xy$

2. Para cada uma das funções seguintes, indique um invariante de ciclo que lhe permita provar a correcção parcial. Em cada um dos casos, mesmo informalmente, apresente argumentos que lhe permitam demonstrar as propriedades (inicialização, preservação e utilidade) dos invariantes definidos.

(a) Índice do menor elemento de um array.

```
int minInd (int v[], int N) {
    // pre: N>0
    int i = 1, r = 0;
    // inv: ???
    while (i<N) {
        if (v[i] < v[r]) r = i;
        i = i+1;
    }
    // pos: 0 <= r < N && forall_{0 <= k < N} v[r] <= v[k]
    return r;
}
```

1) Inicialização: $\{N>0\}$ $i=1; r=0$ $\{I\}$
2) Preservação: $\{I \wedge i<N\}$ if (v[i] < v[r]) r=i; i=i+1 ; $\{I\}$
3) Utilidade: $\{I \wedge i \geq N\} \rightarrow 0 \leq r < N \wedge \forall_{0 \leq k < N} \; v[r] \leq v[k]$

(possível) $I \equiv 0 \leq r < i \wedge i \leq N \wedge \forall_{0 \leq k \leq i} \; v[r] \leq v[k]$

1) $N>0 \rightarrow 0 \leq 0 < 1 \wedge \underbrace{1 \leq N}_{N>0} \wedge \underbrace{\forall_{0 \leq k < 1}}_{k=0} v[0] \leq v[k]$
$v[0] \leq v[0]$

2) Condição True / False

True $\quad I \wedge i<N \wedge v[i] < v[r] \rightarrow I[i+1/i]/[r/i]$
False $\quad I \wedge i \geq N \wedge v[i] \geq v[r] \rightarrow I[i+1/i]$

(b) Menor elemento de um array.

```
int minimo (int v[], int N) {
    // pre: N>0
    int i = 1, r = v[0];
    // inv: ???
    while (i<N) {
        if (v[i] < r) r = v[i];
        i=i+1;
    }
    // pos: (forall_{0 <= k < N} r <= v[k]) &&
    //      (exists_{0 <= p < N} r == v[p])
    return r;
}
```

1) Inicialização: $\{N>0\}$ $i=1; r=v[0]$ $\{I\}$
2) Preservação: $\{I \wedge i<N\}$ if (v[i] < r) r = v[i] ; i=i+1; $\{I\}$
3) Utilidade: $\{I \wedge i \geq N\} \rightarrow (\forall_{0 \leq k < N} . r \leq v[k] \wedge \exists_{0 \leq p < N} . r == v[p])$

$I \equiv i \leq N \wedge \forall_{0 \leq k < i} . r \leq v[k] \wedge \exists_{0 \leq p < i} . r == v[p]$

(c) Soma dos elementos de um array.

```
int soma (int v[], int N) {
    // pre: N>0
    int i = 0, r = 0;
    // inv: ???
    while (i<N) {
        r = r + v[i]; i=i+1;
    }
    // pos: r == sum_{0 <= k < N} v[k]
    return r;
```

1) Inicialização: $\{N>0\}$ $i=0; r=0$ $\{I\}$
2) Preservação: $\{I \wedge i<N\}$ r=r+v[i] ; i=i+1; $\{I\}$
3) Utilidade: $\{I \wedge i \geq N\} \rightarrow r == \sum_{k=0}^{N} v[k]$

$I \equiv i \leq N \wedge r = \sum_{k=0}^{i} v[k]$

a	b	r
6	6	0
3	12	0
1	24	12
1	48	36

$a*b+r = x^2$

(d) Quadrado de um número inteiro positivo.

```
int quadrado1 (int x) {
    // pre: x>=0
    int a = x, b = x, r = 0;
    // inv: ??
    while (a!=0) {
          if (a%2 != 0) r = r + b;
          a=a/2; b=b*2;
    }
    // pos: r == x^2
    return r;
}
```

1) Inicialização: $\{x \geq 0\}$ $a=x$; $b=x$; $r=0$ $\{I\}$
2) Preservação: $\{I \wedge a != 0\}$ if(a%2 != 0) r=r+b; a=a/2; b=b*2; $\{I\}$
3) Utilidade: $\{I \wedge a == 0\} \rightarrow r == x^2$

$I \equiv a*b + r = x^2$

(f) Tamanho do maior prefixo ordenado de um array.

```
int maxPOrd (int v[], int N){
    // pre: N>0
    int r = 1;
    // inv: ??
    while (r < N && v[r-1] <= v[r])
        r = r+1;
    // pos: 0≤r<N && forall {0≤k<r-1} v[k] ≤ v[k+1]
    return r;   && (r<N → v[r] < v[r-1])
}
```

1) Inicialização: $\{N>0\}$ $r=1$ $\{I\}$
2) Preservação: $\{I \wedge r<N \wedge v[r-1] \leq v[r]\}$ $r=r+1$; $\{I\}$
3) Utilidade: $\{I \wedge r \geq N \wedge v[r-1] > v[r]\} \rightarrow (0 \leq r < N \wedge \forall_{0 \leq k < r-1} . v[k] \leq v[k+1] \wedge (r<N \rightarrow v[r] < v[r-1]))$

$I \equiv 0 \leq r \leq N \wedge \forall_{0 \leq k < r-1} . v[k] \leq v[k+1]$