

Algorithmie en langage C

Semestre 3

Table des matières

1	Paradigmes de programmation	5
1.1	Programmation fonctionnelles	5
1.2	Programmation déclarative	5
1.3	Programmation Impérative	6
2	Programmation impérative en C	7
2.1	Description de l'organisation des données en mémoire	7
2.2	Code syntaxe	7
2.3	Structure d'une programme en C	10
2.4	La compilation	11
3	Méthodologie de la programmation impérative	12
3.1	Programmation "en petit"	12
3.2	Programmation en large	12
3.3	Développement d'un algorithme	12
3.4	Étape 1 : comprendre le problème	12
3.5	Étape 2 : Spécification du programme : spécification formelle	13
3.6	Étape 3 : Donner un modèle de solution	13
3.7	Étape 4 : Programmer et unifier le programme	13
4	Le tableau de situation	14
5	Spécification d'un programme	15
5.1	Mots clés à utiliser dans les prédicats	15

5.2	Écriture de la spécification	15
6	Vérification formelle de programmes	16
6.1	Système de réécriture Pfp	16
6.2	Calcul de pfp d'une affectation	16
6.3	Calcul du pfp d'une séquence	17
6.4	Calcul du pfp de la sélection	17
6.5	Calcul du pfp d'une répétition.	18
7	Ecriture d'un programme à partir de sa preuves	19
7.1	Modèle de boucle	19
A	Glossaire	21
B	Liste des codes sources	22
C	Exercices	23
C.1	Initiation	23
C.2	Tableau de situation	25
C.3	Effets de bords	26
C.4	Spécification	28
C.5	Preuves de programmes	30
C.6	Écriture d'un programme à partir de sa preuve	38

Ecriture d'un programme à partir de sa preuves

Il s'agit d'écrire un programme à partir de ses spécifications..

```

/* P */
action(N,r);
/* Q */

```

Il permet de définir un modèle de solution :

- Séquence, boucle, recherche linéaire, recherche dichotomique
- Dédit de l'analyse de Q et de P

7.1 Modèle de boucle

Il faut déterminer l'invariant et la condition de boucle afin de construire complètement le programme. Nous allons ainsi utiliser la propriété suivante :

$$INV \wedge \neg C \rightarrow Q$$

On pose $INV \wedge \neg C = Q$, si cette propriété est vérifiée, alors l'implication sera vérifiée.

7.1.1 1^{ère} approche : Preuve avec invariant trivial

$Q = \top \wedge Q$ Q étant \neg condition

1. $p \rightarrow \text{pfp}(\text{init}, \top) = p \rightarrow \top$ Toujours vrai
2. $\top \wedge Q \rightarrow \text{pfp}(\text{corps}, \top) = \top \wedge Q \rightarrow \top$
3. $\top \wedge Q \rightarrow Q$ Toujours vrai¹

Toute la difficulté est de prouver la terminaison.

7.1.1.1 Exemple

```

1 /* (x = A) ∧ (y=B) ∧ (z=C) */
2 init
3 /* ⊤ */

```


-
1. $Q \rightarrow Q$

```

4 while ((x >= y) || (y >= z)) {
5     /*  $\top \wedge (x \geq y) \wedge (y \geq z)$  */
6     if(x >= y)
7         echange(&x, &y);
8
9     if(y >= z)
10        echange(&y, &z);
11    /* \top */
12 }
13 /*  $\top \wedge (x < y) \wedge (y < z)$  */
14 /*  $(x < y) \wedge (y < z)$  */

```

Nous pouvons prendre $x - z + |A + B + C|$ comme variante

 Prendre un invariant trivial complique la preuve de terminaison et cela réduit l'écriture du programme à la recherche de la variante.

7.1.2 2nd approche : Élimination de conjoint

$Q : A \wedge B \wedge C \wedge \dots$ ²

Si on peut trouver une séquence d'initialisation qui permet de vérifier les conjoints de façon simple ; ces conjoints forment l'invariant.

$Q = A \wedge B \wedge E$ on peut écrire $INV \wedge E$.

7.1.2.1 Exemple

```

1  /* N > 0 */
2  a = N;
3  /*  $a^2 \leq N$  */
4  while ( a*a > N) {
5      /*  $a^2 \leq N \wedge cond$  */
6      a = a - 1;
7      /*  $a^2 \leq N$  */
8  }
9
10 /*
11  *  $a^2 \leq N \leq (a+1)^2$  peut aussi être écrit  $a^2 \leq N \wedge N \leq (a+1)^2$ 
12  *  $a^2 \leq N$  : INV
13  *  $N \leq (a+1)^2$  :  $\neg C$ 
14  */

```

2. A, B et C sont des conjoints

R Cette solution fonctionne, cependant le programme à une complexité linéaire (N), celui-ci peut être résolu avec une complexité logarithmique.

7.1.3 3^{ème} approche : introduction d'une variable dans Q

En général, Q s'écrit $Q(N)$. On va le réécrire en introduisant une variable : $Q(i) \wedge (i = N)$

Ainsi $Q(i)$ devient l'invariant et $i = N$ la condition de boucle (\neg condition).

7.1.3.1 Exemple

```
1  /* N > 0 */
2  f = 1;
3  i = 1;
4  /* f = i! */
5  while (i != N) {
6      ++i;
7      f *= i;
8  }
9
10 /*
11  * f = N !
12  * On le réécrit f = i! ^ i = N
13  */
```

Exercices

C.1 Initiation

C.1.1 Exercice 1

Écrire un programme qui lit une série de 10 valeurs et affiche la position du minimum et du maximum de la série.

C.1.1.1 Étape 1 : Analyser le problème

1. Lire les valeurs
2. calculer les min et max
3. afficher le résultat

C.1.1.2 Étape 2 : Spécifier les sous-problèmes

Identifier les entrée, les sorties et leurs propriétés.

LireLesValeurs

Entrée Nombre, les valeurs à lire

Sortie Tableau contenant les valeurs lues

CalculerMinEtMax

Entrée Le tableau des valeurs et le nombre de valeur

Sortie Position, min et max.

C.1.1.3 Étape 3 : Le code

```

1 #include <stdlib.h>
2 #define N 100
3
4 void read (int nb, int* tab) ;
5 void calculerMinMax(int nb, int* t, int *pmin, int *vmin, int *pmax
   , int *vmax);
6 void read (int nb, int* tab) ;
7
8 int main (int argc, char** argv) {
9     int pmin, vmin, pmax, vmax;
10    int tab[N];
11    read(10, tab);
12    calculerMinMax(10, tab, &pmin, &vmin, &pmax, &vmax);
13    printf(...);
14 }
15 // un tableau est un pointeur sur le premier élément
16 // peut aussi être écrit int tab[N]
17 void read (int nb, int* tab) {
18     int i;
19     for(i=0; i < nb ; ++i) {
20         scanf('%d', tab+i);
21     }
22 }
23
24 void calculerMinMax(int nb, int *tab, int *pmin, int *vmin, int *
   pmax, int *vmax) {
25     *pmin = 0;
26     *pmax = 0;
27     *vmin = t[pmin];
28     *vmax = t[pmax];
29
30     for(; --nb = b > 0;) {
31         if(tab[nb] < *vmin) {
32             *vmin = tab[nb];
33             *pmin = nb;
34         }
35         if(tab[nb] > *vmax) {
36             *vmax = tab[nb];
37             *pmax = nb;
38         }
39     }
40 }

```

Listing C.1 – Exercice 1 – Code du programme

C.2 Tableau de situation

C.2.1 Exercice 2

C.2.2 Exercice 3

```

1  typedef tab int[3];
2  tab T;
3  int j = 2;
4
5  void modifier1(int x) {
6      j++;
7      x = 1;
8  }
9
10 void modifier2 (int *x) {
11     j++;
12     *x = 1;
13 }
14
15 void modifier3 (tab b) {
16     T[0] = b[0] + b[j] + b[2] + b[3] + b[4];
17     t[1] = b[0] + b[1] + b[2] + b[3] + b[4];
18 }
19
20 int main(int argc, char** argv) {
21     int i;
22     for(i=0; i < j; ++i) {
23         T[i] = 0;
24     }
25     modifier1(T[j]); // ... 1
26     modifier2(&T[j]) ; // ... 2
27
28     for(i = 0; i < j; ++i) {
29         T[i] = 1;
30     }
31     modifier3(T);
32
33     return 0;
34 }

```

Listing C.2 – Exercice 3

Point d'arrêt	T	j	i	n
1	0,0,0,0,0	3	5	1
2	0,0,0,1,0	4	3	@
3	5,9,1,1,1	4	5	

C.3 Effets de bords

C.3.1 Exercice 4

```

1  int y;
2
3  int f(int x) ;
4
5  int main(int argc, char *argv[]) {
6      int i, z;
7      y = 10;
8      i = 1; //...2
9      z = f(i) + y; //...3
10
11     return z;
12 }
13
14 int f(int x) {
15     int t = x;
16
17     ++y;
18     ++t;
19
20     return (t+y); //...1
21 }
```

Listing C.3 – Exercice 4

Point d'arrêt	y	x	t	i	z	f
2	10	/	/	1		
1	11	1	2	1	/	13
3	11	/	/	1	24	/

C.3.2 Exercice 5

```
1 int i;
2 int j;
3
4 void pr (int *x, int *y, int *z) ;
5
6 int main(int argc, char *argv[]) {
7     i = 3;
8     j = 7; // ... 3
9     pr(&i, &i, &j); // ... 4
10
11     i = 3;
12     j = 7; // ... 5
13     pr(&j, &j, &i); /// ...6
14 }
15
16 void pr (int *x, int *y, int *z) {
17     *y = i + *x; // ... 1
18     *z = *x + *y; // ... 2
19 }
```

Listing C.4 – Exercice 5

Point d'arrêt	i	j	x	y	z
3	3	7	/	/	/
1			@i	@i	@j
2	6	12	@i	@i	@j
4	6	12	/	/	/
5	3	7	/	/	/
1	3	10	@j	@j	@i
2	20	10	@j	@j	@i

C.3.3 Exercice 6

```

1  int i;
2
3  int f(int a, int b){
4      i = i + a;
5
6      return (a + b); // 1
7  }
8
9  int main(int argc, char *argv[]) {
10     int j, x;
11
12     i = 10;
13     j = 10; // 2
14     x = f(i, j); // 3
15 }

```

Listing C.5 – Exercice 6

Point d'arrêt	i	j	x	a	a	f
2	10	40	/	/	/	/
1	20	40	/	10	40	50
3	20	40	50	/	/	/

C.4 Spécification

C.4.1 Exercice 7

Écrire la spécification d'un programme qui dans un tableau T de N entiers calcul le nombre n de nombre positifs dans le tableau.

- $N > 0$
- `calculeNbPos(T, N, n)`
- $(0 \leq n \leq N) \wedge (n = \nu I : 0 \leq I < NT[I] \geq 0)$

C.4.1.1 Exercice 8

Soit T un tableau croissant (non strict) de N entier et X un entier.

Spécifier un programme qui calcule la position de la dernière occurrence de T inférieure ou égale à X avec $T[0] \leq X < T[N - 1]$

- $(N > 1) \wedge (T[0] \leq X) \wedge (X < T[N - 1]) \wedge (\forall I : 0 \leq I < N - 1 \rightarrow T[I] \leq T[I + 1])$
- `searchPosition(T, N, X, p);`
- $(0 < p < N - 1) \wedge (T[p] \leq X) \wedge (T[p + 1] > X)$

R Dans la suite du cours, nous pourrions utiliser un raccourci afin de savoir si un tableau est trié par ordre croissant : (T, N, \leq)
 Celle-ci pourra être utilisée dans la copie à condition qu'elle soit définie au préalable.

C.4.2 Exercice 8

Soit un tableau T non vide de N entiers. Écrire la spécifications du programme qui calculent :

- La première position de la valeur max de T
- La dernière position de la valeur max de T

C.4.2.1 Calcule de la première position

- $N > 0$
- `searchFirstPosition(T, N, f);`
- $(\forall I : 0 \leq I < f \rightarrow T[I] < T[f]) \wedge (\forall I (f \leq I < N) \rightarrow (T[I] \leq T[f]))$

C.4.2.2 Calcule de la dernière position

- $N > 0$
- `searchLastPosition(T, N, l);`
- $(\forall I : 0 \leq I < l \rightarrow T[I] \leq T[l]) \wedge (\forall I (l < I < N) \rightarrow (T[I] < T[l]))$

C.4.3 Exercice 9

Écrire la spécification d'un programme qui, dans un tableau T de N entiers tous différents cherche la position d'une valeur X si elle existe ou retourne N si elle n'existe pas.

- $(N \geq 0) \wedge (\forall I : 0 \leq I < N \rightarrow (\forall (I, J) : 0 \leq I < N \wedge (0 \leq J < N) \rightarrow T[I] = T[J] \leftrightarrow (I = J)))^1$
- `search(T, N, x)`
- $(0 \leq p < N \wedge T[p] = X) \vee (p = N) \leftrightarrow \forall I (0 \leq I < N) \rightarrow T[I] \neq X)$

C.4.4 Exercice 10

Spécifier un programme qui, dans un tableau T de N éléments trié par ordre croissant non strict retourne la longueur du plus grand plateau².

- $(T, N, \leq) \wedge N > 0$
- `longueurPlusGrandPlateau(T, N, l);`
- $(1 \leq l \leq N) \wedge (\exists I : 0 \leq I < N - l) \wedge (T[I] = T[I + l - 1])$

1. Cela peut aussi s'écrire $N \geq 0) \wedge (\forall I (0 \leq I < N) \rightarrow \forall J : J \neq I \wedge 0 \leq J < N \rightarrow T[I] \neq T[J])$

2. Un plateau est quand il y a plusieurs fois le même caractère

C.4.5 Exercice 11

Spécifier un programme qui, dans un tableau de N entiers calcule le nombre de doublons : un doublon est une succession de 2 nombres identiques.

- $N > 0$
- `calculeDoublons(T, N, n);`
- $n = \nu I : 0 \leq I < N \wedge T[I] = T[I + 1]$

R Dans le cas ou deux doublons ne sont pas forcément côte à côte, le prédicat de sortie deviendrait :

$$n = \sum_{I=0}^{N-1} \nu J : I < J < N \wedge T[J] = T[I]$$

C.5 Preuves de programmes

C.5.1 Séquence

- `/* f = i! */`
- `f = f * (i + 1);`
- `i = i + 1;`
- `/* f = i! */`

$$\begin{aligned} f = i! &\rightarrow \text{pfp}("f = f \times (i + 1); i = i + 1;", f = i!) \\ f = i! &\rightarrow \text{pfp}("f = f \times (i + 1);", \text{pfp}("i = i + 1", f = i!)) \end{aligned}$$

- `/* (x = A) ∧ (y = B) ∧ (z = C)`
- `x = x + y + z;`
- `z = x - y - z;`
- `y = x - y - z;`
- `x = x - y - z;`
- `/* (x = B) ∧ (y = C) ∧ (z = A) */`

$$PE \rightarrow \text{pfp}("x = x + y + z; z = x - y - z; y = x - y - z; x = x - y - z;", (x = B) \wedge (y = C) \wedge (z = A))$$

$$PE \rightarrow \text{pfp}("x = x + y + z, z = x - y - z; y = x - y - z;", \text{pfp}("x = x - y - z; (x = B) \wedge (y = C) \wedge (z = A)"))$$

$$PE \rightarrow \text{pfp}("x = x + y + z, z = x - y - z; y = x - y - z;", (x - y - z = B) \wedge y = C \wedge z = A)$$

$$PE \rightarrow \text{pfp}("x = x + y + z, z = x - y - z", (y = B) \wedge (x - y - z = C) \wedge z = A)$$

$$PE \rightarrow \text{pfp}("x = x + y + z", (y = B) \wedge (z = C) \wedge (x - y - z = A))$$

$$PE \rightarrow (y = B) \wedge (z = C) \wedge (x = A) \text{ Vrai parceque } p \rightarrow p = \text{vrai}$$

C.5.2 Sélection

C.5.2.1 Exercice 1

```

1  /* x = A */
2  if (x < 0)
3  x = -x;
4
5  /* x = |A| */

```

$$\begin{aligned}
 /* x = A */ &\rightarrow \text{pfp}("if(x < 0)\{x = -x\}", x = |A|) \\
 /* x = A */ &\rightarrow (((x < 0) \rightarrow \text{pfp}("x = -x;", x = |A|) \wedge (x \geq 0 \rightarrow x = |A|))) \\
 /* x = A */ &\rightarrow (((x < 0) \rightarrow (-x = |A|)) \wedge ((x \geq 0) \rightarrow (x = |A|)))
 \end{aligned}$$

$(A(A < 0) \rightarrow (-A = |A|)) \wedge (A \geq 0 \rightarrow (A = |A|))$ Définition de la valeur absolue $|\cdot|$.

C.5.2.2 Exercice 2

```

1  /* x = A ∧ y = B */
2  if (A < B) {
3  x = A;
4  y = B;
5  } else {
6  x = B;
7  y = A;
8  }
9  /* x ≤ y */

```

$$\begin{aligned}
 PE &\rightarrow \text{pfp}("if(A < B)\{x = A; y = B\}else\{x = B; y = A\}", x \leq y) \\
 PE &\rightarrow ((A < B) \rightarrow \text{pfp}("x = A; y = B", x \leq y)) \wedge ((A \geq B) \rightarrow \text{pfp}("x = B; y = A", x \leq y)) \\
 PE &\rightarrow (((A < B) \rightarrow (A \leq B)) \wedge (A \geq B) \rightarrow (B \leq A))
 \end{aligned}$$

Vrai par définition. A est toujours inférieur à B. $(A \geq B) \rightarrow (B \leq A)$ est une Tautologie.

C.5.2.3 Exercice 3

$$\begin{aligned}
 &\text{pfp}("if(x \geq y)\{z = x;\}else\{z = y\}", z = \max(x, y)) \\
 x \geq y &\rightarrow \text{pfp}("if(x \geq y)\{z = x;\}else\{z = y\}", z = \max(x, y)) \\
 x \geq y &\rightarrow \text{pfp}("z = x", z = \max(x, y)) \\
 x < y &\rightarrow \text{pfp}("z = y", z = \max(x, y)) \\
 x \geq y &\rightarrow x = \max(x, y) \\
 x < y &\rightarrow y = \max(x, y)
 \end{aligned}$$

C'est une tautologie par définition de **max**.

C.5.2.4 Exercice 4

$\text{pfp}(\text{"if}(x > y)\{if(x \% 2 == 0)\{x = x - 2\}\}\text{else}\{y = y - 1;\}\text{"}, y - 2 < x);$

$(x > y) \rightarrow \text{pfp}(\text{"if}(x \% 2 == 0)x = x - 2;\text{"}, y - 2 < x)) \wedge ((x \leq y) \rightarrow \text{pfp}(\text{"y = y - 1;\text{"}, y - 2 < x))$

$(x > y) \rightarrow (((x \% 2 == 0) \rightarrow \text{pfp}(\text{"x = x - 2;\text{"}, y - 2 < x) \wedge$
 $(x \% 2 \neq 0) \rightarrow (y - 2 < x)) \wedge ((x \leq y) \rightarrow (y - z < x))$

$(x > y) \rightarrow ((x \% 2 == 0) \rightarrow ((y - 2) < (x - 2)) \wedge ((x \% 2 \neq 0) \rightarrow (y - 2 < x))) \wedge (x \leq y \rightarrow y - z < x)$

$(x > y) \rightarrow (x \leq y \rightarrow y - z < x)$

C.5.2.5 Exercice 5

```
1 /* N ≥ 0 */
2
3 /* P Tableau de polynôme
4 * N Degré du polynôme
5 * X Point ou je veux évoluer le polynôme
6 * r Résultat du polynôme
7 */
8 calculPolynome(P,N,X,r);
9 /* r = ∑k=0N A[k]Xk */
```

```
1 /* N ≥ 0 */
2
3 /* INV = (0 ≤ i ≤ N) ∧ (r = ∑k=0i A[k]xk) ∧ (y = xi) */
4 while(c) {
5 /* c \wedge INV */
6 corps;
7 /* INV */
8 }
9 /* ¬c ∧ INV */
10 /* p = ∑k=0N A[k]xk */
```

INVARIANT

$$(0 \leq i \leq N) \wedge (r = \sum_{k=0}^i A[k]x^k) \wedge (y = x^i)$$

Intitialisation

1. INIT 0

```
i = 0;
p = 0;
y = 1;
```

$$\begin{aligned}
PE &\rightarrow \text{pfp}(\text{init } 0, \text{INV}) \\
N \geq 0 &\rightarrow \text{pfp}("i = 0; p = 0", p = \sum_{k=0}^i A[k]x^k \wedge 1 = x^i); \\
N \geq 0 &\rightarrow \text{pfp}("i = 0", 0 = \sum_{k=0}^i A[k]x^k \wedge 1 = x^i) \\
N \geq 0 &\rightarrow 0 = \sum_{k=0}^0 A[k]x^k \wedge 1 = x^0 \\
(N \geq 0 \rightarrow 0 = A[0] \wedge 1 = 1) &\Rightarrow \text{Faux et Vrai, donc le programme est Faux}
\end{aligned}$$

2. INIT 1

```

i = 0;
p = A[0];
y = 1;

```

$$\begin{aligned}
PE &\rightarrow \text{pfp}(\text{init } 1, \text{INV}) \\
N \geq 0 &\rightarrow \text{pfp}("i = 0; p = A[0]", p = \sum_{k=0}^i A[k]x^k \wedge 1 = x^i); \\
N \geq 0 &\rightarrow \text{pfp}("i = 0", A[0] = \sum_{k=0}^i A[k]x^k \wedge 1 = x^i) \\
N \geq 0 &\rightarrow A[0] = \sum_{k=0}^0 A[k]x^k \wedge 1 = x^0 \\
(N \geq 0 \rightarrow A[0] = A[0] \wedge 1 = 1) &\Rightarrow \text{Vrai et Vrai, donc le programme est correct}
\end{aligned}$$

L'initialisation nécessaire est donc init 1.

Boucles

1.

```

while(i < N) {
  ++i;
  p = p + A[i-1] * y;
  y = y * X;
}

```

Étape numéro 3 (Cf section 6.5 page 18)

$$\begin{aligned}
\neg C \wedge INV &\rightarrow PS \\
(i > N) \wedge (p = \sum_{k=0}^i A[k]x^k) \wedge (y = x^i) &\rightarrow p = \sum_{k=0}^N A[k]x^k \\
(i > N) \wedge (p = \sum_{k=0}^i A[k]x^k) \wedge (y = x^i) &\rightarrow p = \sum_{k=0}^N A[k]x^k \wedge (i = N) \\
(i > N) &\rightarrow (i = N) \Rightarrow \text{C'est donc faux.}
\end{aligned}$$

R Nous sommes partis de la conclusion et avons essayé de faire apparaître notre hypothèse en partie droite.

```

while( i != N) {
  ++i;
  p = p + a[i-1] * y;
  y = y * X
}

```

Étape numéro 3 (Cf section 6.5 page 18)

$$\begin{aligned}
& \neg C \wedge INV \rightarrow PS \\
& (i \neq N) \wedge (p = \sum_{k=0}^i A[k]x^k) \wedge (y = x^i) \rightarrow p = \sum_{k=0}^N A[k]x^k \\
& (i \neq N) \wedge (p = \sum_{k=0}^i A[k]x^k) \wedge (y = x^i) \rightarrow p = \sum_{k=0}^N A[k]x^k \wedge (i = N) \\
& (i \neq N) \rightarrow (i = N) \Rightarrow \text{C'est donc vrai.}
\end{aligned}$$

Étape numéro 2 (Cf section 6.5 page 18)

$$\begin{aligned}
& C \wedge INV \rightarrow \text{pfp}(\text{corps}, INV) \\
& C \wedge INV \rightarrow \text{pfp}("i++; p = p + A[i-1] * y;", (p = \sum_{k=0}^i A[k]x^k) \\
& \quad \wedge (y * x = x^i) \wedge (0 \leq i \leq n) \\
& C \wedge INV \rightarrow \text{pfp}("i++;", (p + A[i-1] * y = \sum_{k=0}^i A[k]x^k) \\
& \quad \wedge (y * x = x^i) \wedge (0 \leq i \leq n) \\
& (i \neq N) \wedge (0 \leq i \leq N) \wedge \\
& (p = \sum_{k=0}^i A[k]x^k) \wedge (y = X^i) \rightarrow (p + A[i] * i = \sum_{k=0}^{i+1} A[k]x^k) \wedge (y * x = x^{i+1}) \wedge (0 \leq i+1 \leq N) \\
& (i \neq N) \wedge (0 \leq i \leq N) \wedge \\
& (p = \sum_{k=0}^i A[k]x^k) \wedge (y = X^i) \rightarrow A[i] * x^i = p + A[i+1]x^{i+1} \wedge x^i * y = x^{i+1} \wedge \text{Tautologie} \\
& (i \neq N) \wedge (0 \leq i \leq N) \wedge \\
& (p = \sum_{k=0}^i A[k]x^k) \wedge (y = X^i) \rightarrow \text{Faux} \wedge \text{Tautologie} \wedge \text{Tautologie} \Rightarrow \text{Le programme est donc faux}
\end{aligned}$$

3.

```

while(i == N) {
  ++i;
  p = p + (A[i] * y * X);
  y = y * X;
}

```

Ce programme effectue la correction de l'erreur détectée plus haut. Il est correct.

C.5.3 Exercice 6 : Suite de Fibonacci

R Le rang n de la suite de Fibonacci est défini comme suit :

$$F_n = 1 \text{ si } n = 0 \text{ ou } n = 1$$

$$F_n = F_{n-2} + F_{n-1} \text{ si } n > 1$$

```
/* N ≥ 0 */
i = 0;
a = 1;
b = 1;
/* INV = (F_i = a) ∧ (F_{i+1} = b) ∧ (0 ≤ i ≤ N) */
while(c) {
/* c ∧ INV */
i++;
b += a;
a = b - a;
/* INV */
}
/* ¬c ∧ INV */
/* F_N = {...} */
```

R Il est conseillé de commencer par l'étape la plus facile, en effet, une étape et nous n'avons pas à effectuer les autres

Etape 1

$$\begin{aligned} PE &\rightarrow \text{pfp}(\text{"init"}, INV) \\ N \geq 0 &\rightarrow (F_{i+1} = 1) \wedge (F_i = 1) \wedge (i = 0) \\ N \geq 0 &\rightarrow (F_1 = 1) \wedge (F_0 = 1) \Rightarrow \text{Tautologie par définition} \end{aligned}$$

Etape 3

$$\begin{aligned} (i = N) \wedge (F_i = a) \wedge (F_{i+1} = b) \wedge (0 \leq i \leq N) &\rightarrow F_N = a \\ (F_n = a) \wedge (F_{N+1}) \wedge (0 \leq N) &\rightarrow F_N = a \Rightarrow \text{Tautologie} \end{aligned}$$

Etape 2

$$\begin{aligned} (i \neq N) \wedge (F_i = a) \wedge (F_{i+1} = b) \wedge (0 \leq i \leq N) &\rightarrow \text{pfp}(\text{"..."}, F_i = b - a \wedge F_{i+1} = b \wedge \dots) \\ &\rightarrow \text{pfp}(\text{"..."}, F_i = b + a - a \wedge F_{i+1} = b + a) \\ &\rightarrow F_i + 1 = b \wedge F_{i+2} = b + a \wedge i + 1 \leq N \\ &\rightarrow T \wedge T \text{ par définition} \wedge T \end{aligned}$$

Étape 4 Tester une existence de $f > 0$ avant l'exécution du corps : $c \wedge INV \rightarrow f > 0$.

$$(i \neq N) \wedge (F_i = a) \wedge (F_n - 1 = b) \wedge (0 \leq i \leq N) \rightarrow f > 0$$

$$f = N - i \Rightarrow N \geq 0 \text{ donc Vrai.}$$

Étape 5

$$f = F \wedge c \wedge INV \rightarrow \text{pfp}(\text{"corps"}, f < F)$$

$$N - i = F \wedge c \wedge INV \rightarrow \text{pfp}(\text{"corps"}, N - i < F)$$

$$N - i = F \wedge c \wedge INV \rightarrow N - (i + 1) < F$$

$$\rightarrow N - i - 1 < F$$

$$\rightarrow F - 1 < F \Rightarrow \text{Tautologie}$$

C.5.4 Exercice 7 : plus grand plateau

```

1  /* (N ≥ 1) ∧ (B(N), ≥) */
2  j = 1;
3  p = 1;
4  /* INV = (1 ≤ p ≤ j ≤ N) ∧ (∃k : 0 ≤ k ≤ j - p) ∧ (B[k] = B[k + p - 1]))
5  ∧ (∀k(0 ≤ k ≤ j - p - 1) → (B[k] ≠ B[k + p])) */
6  while (j != N) {
7      /* INV ∧ c */
8      if (B[j-p] == B[j])
9          p++;
10
11     j++;
12     /* INV */
13 }
14 /* ¬c ∧ INV */
15 /* R = (1 ≤ p ≤ N) ∧ (∃k : 0 ≤ k ≤ N - p) ∧
16 (B[k] = B[k + p - 1])) ∧ (∀k(0 ≤ k ≤ N - p - 1) → (B[k] ≠ B[k + p])) */

```

R $(B(N), \geq)$ signifie le tableau est ordonné dans l'ordre croissant non strict

Soit PE le prédicat d'entrée.

$$x \geq 1 \rightarrow \text{pfp}(j = 1, p = 1, Inv)$$

$$x \geq 1 \rightarrow (0 \leq 1 \leq 1 \leq N) \wedge \exists k(0 \leq k \leq 0(B[k] = B[k])) \wedge \forall k(0 \leq k \leq -1 \rightarrow B[k] \neq B[k + 1])$$

$$\Rightarrow \text{Toujours vrai}$$

$$C \wedge INV \rightarrow \text{pfp}(\text{if}(B[j - p] == B[k])p ++; j ++, INV)$$

$$\begin{aligned}
C \wedge INV &\rightarrow \text{pfp}(\text{if}(B[j-p] == B[j])p++, \text{pfp}(j++, INV)) \\
C \wedge INV &\rightarrow \text{pfp}(\dots, 1 \leq p \leq j+1 \leq N \wedge \exists k(0 \leq k \leq j+1-p) \\
&\quad \wedge B[k] = B[k+p-1] \wedge \forall k(0 \leq k < j+1-p-1 \rightarrow B[k] \neq B[k+p])) = A \\
C \wedge INV &\rightarrow B[j-p] == B[j] \rightarrow \text{pfp}(p++, A) \wedge B[j-p] \neq B[j] \rightarrow A \\
C \wedge INV &\rightarrow (B[j-p] \neq B[j] \rightarrow (0 \leq p \leq j+1 \leq N) \wedge \exists k 0 \leq k \leq j+1-p \wedge B[k] = B[k+p-1] \\
&\quad \wedge \forall k 0 \neq k \leq j-p \rightarrow B[k] \neq B[k-p] \\
&\rightarrow T \wedge \exists k 0 \leq k < j-p \wedge B[k] = B[k+1-p] \wedge B[j+1-p] = B[j+1-p+1-p] \wedge \dots
\end{aligned}$$

C.5.5 Définition variante

$$\begin{aligned}
C \wedge INV &\rightarrow f > 0 \\
J \neq N \wedge INV &\rightarrow (f = N - j) > 0 \Rightarrow \text{Vrai}
\end{aligned}$$

$$\begin{aligned}
N - j = F \wedge C \wedge INV &\rightarrow \text{pfp}(\text{corps}, N - j < F) \\
N - j = F \wedge C \wedge INV &\rightarrow \text{pfp}(\text{if}(\dots)) \dots, \text{pfp}(j++, N - j < F) \\
N - j = F \wedge C \wedge INV &\rightarrow B[j-p] = B[j] \rightarrow N - j - 1 < F \wedge B[j-p] \neq B[j] \rightarrow F \\
N - j = F \wedge C \wedge INV &\rightarrow N - j - 1 < F \wedge F - 1 < F
\end{aligned}$$

C.5.6 Exercice 8

```

1  /* A ≤ 0 ∧ B ≥ 0 */ x = A; y = B; z = 1; /* (z = x^y = A^B) ∧ y ≥ 0 */
2
3
4
5
6
7  while(y != 0) {
8      /* z * x^y = A^B ∧ y ≥ 0 */
9      while(y \% 2 == 0) {
10         y = y / 2 ;
11         x = x*x;
12     }
13     /* z = x^y = A^B ∧ y > 0 ∧ y%2 ≠ 0 */
14     y--;
15     z = z * x;
16     /* z = x^y = A^B ∧ y ≥ 0 */
17 }
18 PS : /* z = A^b */

```

C.6 Écriture d'un programme à partir de sa preuve

C.6.1 Dichotomie

```

1  /* N > 0 */
2  a = 1;
3  b = N;
4  while(b != a + 1) {
5      m = (a+b)/2;
6      if(0 <= N-m*m) {
7          a = m;
8      } else {
9          b = m;
10     }
11 }
12 /*
13  * Réécriture :  $a^2 \leq N \wedge N \leq b^2 \wedge (b = a + 1)$ 
14  *  $a^2 \leq N \wedge N \leq (A + 1)^2$ 
15  */

```

Variante : $b - a$: taille de l'intervalle. Fonction décroissante.

C.6.2

```

1  /* N ≤ 2 ∧ B(0..N-1, ≤) ∧ B[0] ≤ X ∧ B[N-1] > X */
2  cherchep(B,N,X,p);
3  i = 2;
4  p = 0;
5  /* 0 ≤ p ≤ i-2 ∧ B(p) ≤ x ∧ B[p+1] > x */
6  while (B[p+1] <= X || i != N) {
7      ++p;
8      ++i;
9  }
10 /*
11  * 0 ≤ p ≤ N-2 ∧ B[p] ≤ x < B[p+1]
12  * Réécriture 0 ≤ p ≤ i-2 ∧ B[p] ≤ x < B[p+1] ∧ i=N
13  */

```

Variante : $N - i$