Définitions

- Un algorithme est une suite d'instructions et d'opérations permettant de résoudre un problème.

Définitions

- Un algorithme est une suite d'instructions et d'opérations permettant de résoudre un problème.
  Par exemple pour résoudre le problème du tri d'une liste de valeurs, on peut utiliser l'algorithme du tri par insertion. Cela consiste à prendre une à une chaque valeur et à l'insérer au bon emplacement dans une liste initialement vide.

Définitions

- Un algorithme est une suite d'instructions et d'opérations permettant de résoudre un problème.
  Par exemple pour résoudre le problème du tri d'une liste de valeurs, on peut utiliser l'algorithme du tri par insertion. Cela consiste à prendre une à une chaque valeur et à l'insérer au bon emplacement dans une liste initialement vide.
- Un programme est la traduction d'un algorithme dans un langage informatique. On dit qu'un programme est la mise en oeuvre d'un algorithme.

Définitions

- Un algorithme est une suite d'instructions et d'opérations permettant de résoudre un problème.
  Par exemple pour résoudre le problème du tri d'une liste de valeurs, on peut utiliser l'algorithme du tri par insertion. Cela consiste à prendre une à une chaque valeur et à l'insérer au bon emplacement dans une liste initialement vide.
- Un programme est la traduction d'un algorithme dans un langage informatique. On dit qu'un programme est la mise en oeuvre d'un algorithme.
  L'algorithme du tri par insertion peut être écrit en Python, en C, ...

Définitions

- Un algorithme est une suite d'instructions et d'opérations permettant de résoudre un problème.
  Par exemple pour résoudre le problème du tri d'une liste de valeurs, on peut utiliser l'algorithme du tri par insertion. Cela consiste à prendre une à une chaque valeur et à l'insérer au bon emplacement dans une liste initialement vide.
- Un programme est la traduction d'un algorithme dans un langage informatique. On dit qu'un programme est la mise en oeuvre d'un algorithme.
  L'algorithme du tri par insertion peut être écrit en Python, en C, ...
- Un paradigme de programmation est une façon de d'approcher un problème et d'en concevoir et modéliser une solution.

Définitions

- Un algorithme est une suite d'instructions et d'opérations permettant de résoudre un problème.
  Par exemple pour résoudre le problème du tri d'une liste de valeurs, on peut utiliser l'algorithme du tri par insertion. Cela consiste à prendre une à une chaque valeur et à l'insérer au bon emplacement dans une liste initialement vide.
- Un programme est la traduction d'un algorithme dans un langage informatique. On dit qu'un programme est la mise en oeuvre d'un algorithme.
  L'algorithme du tri par insertion peut être écrit en Python, en C, ...
- Un paradigme de programmation est une façon de d'approcher un problème et d'en concevoir et modéliser une solution.
  Le langage C est une illustration du paradigme de programmation impérative.
  Le langage OCaml nous permettra d'illustrer le paradigme fonctionnel

Exemple introductif

On considère le programme C suivant :

```
1  int get_imax(int tab[],int size) {
2      int max = 0;
3      int imax;
4      for (int i=0; i<size; i++) {
5          if (tab[i] >= max) {
6              max = tab[i];
7              imax = i;}}
8      return imax;}
```

1. Quel est le résultat renvoyé si le tableau fourni en argument contient dans cet ordre les valeurs : `12, 18, 11, 9, 10` ?

Exemple introductif

On considère le programme C suivant :

```
1  int get_imax(int tab[],int size) {
2      int max = 0;
3      int imax;
4      for (int i=0; i<size; i++) {
5          if (tab[i] >= max) {
6              max = tab[i];
7              imax = i;}}
8      return imax;}
```

1. Quel est le résultat renvoyé si le tableau fourni en argument contient dans cet ordre les valeurs : `12, 18, 11, 9, 10` ?
2. Même question avec le tableau `12, 18, 11, 18, 10`

Exemple introductif

On considère le programme C suivant :

```
1  int get_imax(int tab[],int size) {
2      int max = 0;
3      int imax;
4      for (int i=0; i<size; i++) {
5          if (tab[i] >= max) {
6              max = tab[i];
7              imax = i;}}
8      return imax;}
```

1. Quel est le résultat renvoyé si le tableau fourni en argument contient dans cet ordre les valeurs : `12, 18, 11, 9, 10` ?
2. Même question avec le tableau `12, 18, 11, 18, 10`
3. Même question avec le tableau `-12, -15, -7`

Exemple introductif

On considère le programme C suivant :

```
1  int get_imax(int tab[],int size) {
2      int max = 0;
3      int imax;
4      for (int i=0; i<size; i++) {
5          if (tab[i] >= max) {
6              max = tab[i];
7              imax = i;}}
8      return imax;}
```

1. Quel est le résultat renvoyé si le tableau fourni en argument contient dans cet ordre les valeurs : `12, 18, 11, 9, 10` ?
2. Même question avec le tableau `12, 18, 11, 18, 10`
3. Même question avec le tableau `-12, -15, -7`
4. Même question si le tableau est vide (c'est à dire que `size =0`)

Correction

1. La fonction renvoie 1 (indice de l'élément maximal du tableau)

Définition

Correction

1. La fonction renvoie 1 (indice de l'élément maximal du tableau)
2. la fonction renvoie 3, c'est l'indice de la dernière occurence du maximum des éléments du tablea

Définition

Correction

1. La fonction renvoie 1 (indice de l'élément maximal du tableau)
2. la fonction renvoie 3, c'est l'indice de la dernière occurence du maximum des éléments du tablea
3. C'est un comportement indéfini, la variable `imax` n'est pas initialisée.

Définition

Correction

1. La fonction renvoie 1 (indice de l'élément maximal du tableau)
2. la fonction renvoie 3, c'est l'indice de la dernière occurence du maximum des éléments du tablea
3. C'est un comportement indéfini, la variable `imax` n'est pas initialisée.
4. Une nouvelle fois, le comportement est indéfini car on renvoie la variable `imax` qui n'a jamais été initialisée.

Définition

Correction

1. La fonction renvoie 1 (indice de l'élément maximal du tableau)
2. la fonction renvoie 3, c'est l'indice de la dernière occurence du maximum des éléments du tablea
3. C'est un comportement indéfini, la variable `imax` n'est pas initialisée.
4. Une nouvelle fois, le comportement est indéfini car on renvoie la variable `imax` qui n'a jamais été initialisée.

Définition

Ecrire la spécification d'une fonction c'est donné une description formelle et détaillée de ses caractéristiques. En particulier :

Correction

1. La fonction renvoie 1 (indice de l'élément maximal du tableau)
2. la fonction renvoie 3, c'est l'indice de la dernière occurence du maximum des éléments du tablea
3. C'est un comportement indéfini, la variable `imax` n'est pas initialisée.
4. Une nouvelle fois, le comportement est indéfini car on renvoie la variable `imax` qui n'a jamais été initialisée.

Définition

Ecrire la spécification d'une fonction c'est donné une description formelle et détaillée de ses caractéristiques. En particulier :

- les entrées admissibles : types et valeurs possibles des arguments (préconditions),

Correction

1. La fonction renvoie 1 (indice de l'élément maximal du tableau)
2. la fonction renvoie 3, c'est l'indice de la dernière occurence du maximum des éléments du tablea
3. C'est un comportement indéfini, la variable `imax` n'est pas initialisée.
4. Une nouvelle fois, le comportement est indéfini car on renvoie la variable `imax` qui n'a jamais été initialisée.

Définition

Ecrire la spécification d'une fonction c'est donné une description formelle et détaillée de ses caractéristiques. En particulier :

- les entrées admissibles : types et valeurs possibles des arguments (préconditions),
- ce que renvoie la fonction et les Les effets de bords (*side effects*) éventuels : modification des arguments, affichage, ... Ce sont les (postconditions).

Remarques

- La spécification est souvent donnée en commentaire dans le code source.

Remarques

- La spécification est souvent donnée en commentaire dans le code source.
  Les commentaires en C s'écrivent entre `/*` et `*/` et en OCaml entre `(*` et `*)`

Exemples

Remarques

- La spécification est souvent donnée en commentaire dans le code source.
  Les commentaires en C s'écrivent entre `/*` et `*/` et en OCaml entre `(*` et `*)`
- La vérification des préconditions peut s'effectuer à l'aide d'instructions `assert`. Elles sont de la forme `assert (condition)` en C, et nécessitent d'importer `assert.h`. Si la condition échoue le programme s'arrête et affiche une erreur, c'est ce qu'on appelle de la programmation défensive (anticipation des erreurs).

Exemples

Remarques

- La spécification est souvent donnée en commentaire dans le code source.
  Les commentaires en C s'écrivent entre `/*` et `*/` et en OCaml entre `(*` et `*)`
- La vérification des préconditions peut s'effectuer à l'aide d'instructions `assert`. Elles sont de la forme `assert (condition)` en C, et nécessitent d'importer `assert.h`. Si la condition échoue le programme s'arrête et affiche une erreur, c'est ce qu'on appelle de la programmation défensive (anticipation des erreurs).

Exemples

Ecrire en C, une fonction qui :

Remarques

- La spécification est souvent donnée en commentaire dans le code source.
  Les commentaires en C s'écrivent entre `/*` et `*/` et en OCaml entre `(*` et `*)`
- La vérification des préconditions peut s'effectuer à l'aide d'instructions `assert`. Elles sont de la forme `assert (condition)` en C, et nécessitent d'importer `assert.h`. Si la condition échoue le programme s'arrête et affiche une erreur, c'est ce qu'on appelle de la programmation défensive (anticipation des erreurs).

Exemples

Ecrire en C, une fonction qui :

- Accepte en argument un tablau *non vide* d'entiers.

Remarques

- La spécification est souvent donnée en commentaire dans le code source. Les commentaires en C s'écrivent entre `/*` et `*/` et en OCaml entre `(*` et `*)`
- La vérification des préconditions peut s'effectuer à l'aide d'instructions `assert`. Elles sont de la forme `assert (condition)` en C, et nécessitent d'importer `assert.h`. Si la condition échoue le programme s'arrête et affiche une erreur, c'est ce qu'on appelle de la programmation défensive (anticipation des erreurs).

Exemples

Ecrire en C, une fonction qui :

- Accepte en argument un tablau *non vide* d'entiers.
- Renvoie l'indice de la première occurence du maximum des éléments de ce tableau.

Correction

```c
#include <assert.h>

int get_imax(int tab[],int size) {
    /*  Prend en argument un tableau non vide d'entiers
    Renvoie l'indice de la 1ere occurrence du maximum du tableau*/
    assert (size > 0);
    int max = tab[0];
    int imax = 0;
    for (int i=0; i<size; i++) {
        if (tab[i] > max) {
            max = tab[i];
            imax = i;}}
    return imax;}
```

Jeu de tests

Le comportement correct d'une fonction peut être "validé" (mais pas prouvé), par l'utilisation d'un jeu de test. C'est à dire un ensemble de couple d'entrées du programme et de sorties attendues.

Jeu de tests

Le comportement correct d'une fonction peut être "validé" (mais pas prouvé), par l'utilisation d'un jeu de test. C'est à dire un ensemble de couple d'entrées du programme et de sorties attendues.

Dans la fonction précédente, on pourrait tester (entre autres) des cas limites (*edge cases*), comme par exemple un tableau à un seul élément, ou vide ou des situations ou le maximum se trouve en première ou dernière position du tableau.

Exemple

Jeu de tests

Le comportement correct d'une fonction peut être "validé" (mais pas prouvé), par l'utilisation d'un jeu de test. C'est à dire un ensemble de couple d'entrées du programme et de sorties attendues.
Dans la fonction précédente, on pourrait tester (entre autres) des cas limites (*edge cases*), comme par exemple un tableau à un seul élément, ou vide ou des situations ou le maximum se trouve en première ou dernière position du tableau.

Exemple

1. Ecrire une fonction `contient_double` qui prend en argument un tableau d'entiers et renvoie `true` si ce tableau contient deux éléments consécutifs égaux.

Jeu de tests

Le comportement correct d'une fonction peut être "validé" (mais pas prouvé), par l'utilisation d'un jeu de test. C'est à dire un ensemble de couple d'entrées du programme et de sorties attendues.
Dans la fonction précédente, on pourrait tester (entre autres) des cas limites (*edge cases*), comme par exemple un tableau à un seul élément, ou vide ou des situations ou le maximum se trouve en première ou dernière position du tableau.

Exemple

1. Ecrire une fonction `contient_double` qui prend en argument un tableau d'entiers et renvoie `true` si ce tableau contient deux éléments consécutifs égaux.
2. Proposer un jeu de tests pour cette fonction.

Correction

1.

```
1  bool contient_double(int tab[],int size){
2      for (int i=0;i<size-1;i++){
3          if (tab[i]==tab[i+1]){
4              return true;}}
5      return false;}
```

Correction

1.
```
1  bool contient_double(int tab[],int size){
2      for (int i=0;i<size-1;i++){
3          if (tab[i]==tab[i+1]){
4              return true;}}
5      return false;}
```

2. On pourrait tester les cas suivants :

Correction

1.
```
1  bool contient_double(int tab[],int size){
2      for (int i=0;i<size-1;i++){
3          if (tab[i]==tab[i+1]){
4              return true;}}
5      return false;}
```

2. On pourrait tester les cas suivants :
   - Tableau vide ou à un seul élément

Correction

1.
```
bool contient_double(int tab[],int size){
    for (int i=0;i<size-1;i++){
        if (tab[i]==tab[i+1]){
            return true;}}
    return false;}
```

2. On pourrait tester les cas suivants :
   - Tableau vide ou à un seul élément
   - Même élément mais non consécutifs

Correction

1.
```
1  bool contient_double(int tab[],int size){
2      for (int i=0;i<size-1;i++){
3          if (tab[i]==tab[i+1]){
4              return true;}}
5      return false;}
```

2. On pourrait tester les cas suivants :
   - Tableau vide ou à un seul élément
   - Même élément mais non consécutifs
   - Elément présent en plus de deux exemplaires consécutifs

Correction

1.
```
1  bool contient_double(int tab[],int size){
2      for (int i=0;i<size-1;i++){
3          if (tab[i]==tab[i+1]){
4              return true;}}
5      return false;}
```

2. On pourrait tester les cas suivants :
   - Tableau vide ou à un seul élément
   - Même élément mais non consécutifs
   - Elément présent en plus de deux exemplaires consécutifs
   - Présence du double en tout début ou toute fin de tableau

Exemple introductif

Ecrire la fonction `duree_vol` qui prend en argument un entier positif `n` et renvoie le nombre d'itérations nécessaires avant d'atteindre 1 en prenant cet entier comme valeur initiale dans la suite de syracuse. On rappelle que :

$$s_{n+1} = \begin{cases} \dfrac{s_n}{2} & \text{si } n \text{ est paire} \\ 3s_n + 1 & \text{sinon} \end{cases}$$

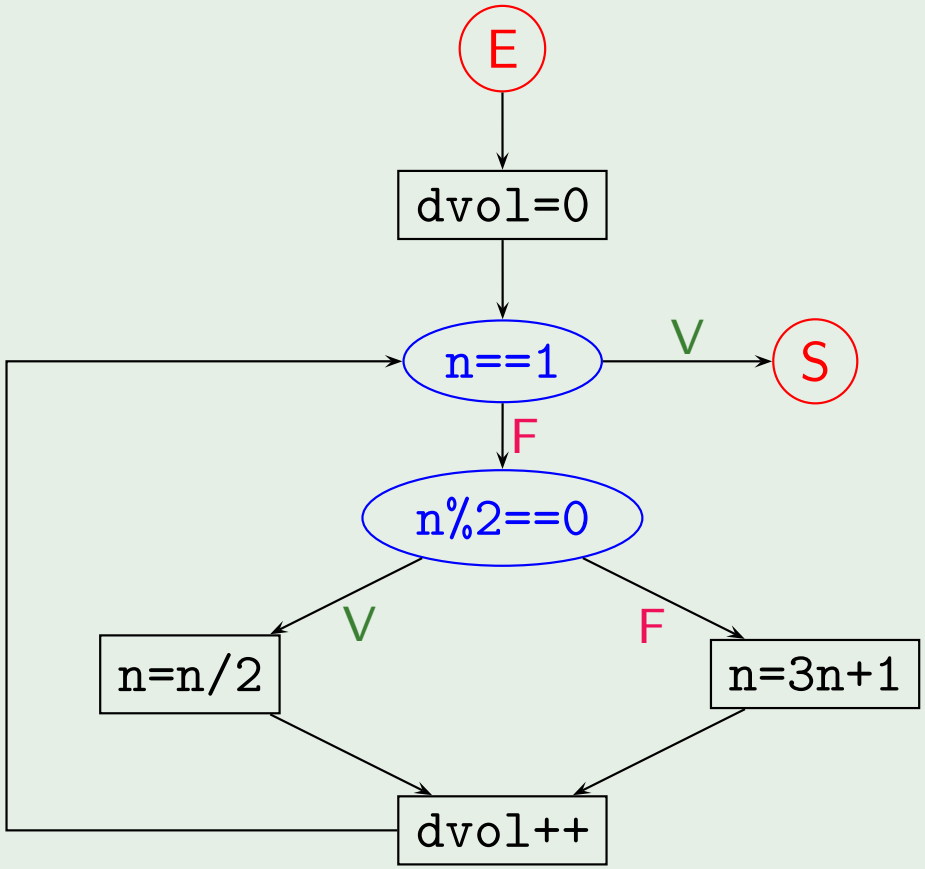
Exemple introductif

Ecrire la fonction `duree_vol` qui prend en argument un entier positif `n` et renvoie le nombre d'itérations nécessaires avant d'atteindre 1 en prenant cet entier comme valeur initiale dans la suite de syracuse. On rappelle que :

$$s_{n+1} = \begin{cases} \dfrac{s_n}{2} & \text{si } n \text{ est paire} \\ 3s_n + 1 & \text{sinon} \end{cases}$$

```
1  int duree_vol(int n) {
2      int dvol = 0;
3      while (n!=1) {
4          if (n%2==0)
5              {n = n/2;}
6          else
7              {n = 3*n+1;}
8          dvol ++;}
9      return dvol;}
```

Représentation des exécutions possibles



Définitions

Le graphe de flot de contrôle représente les exécutions possibles d'un programme.

- Les sommets E et S représentent l'entrée et la sorties

Définition

Définitions

Le graphe de flot de contrôle représente les exécutions possibles d'un programme.

- Les sommets `E` et `S` représentent l'entrée et la sorties
- Les autres noeuds sont les blocs d'instructions

Définition

Définitions

Le graphe de flot de contrôle représente les exécutions possibles d'un programme.

- Les sommets `E` et `S` représentent l'entrée et la sorties
- Les autres noeuds sont les blocs d'instructions
- On dessine un arc entre deux noeuds `A` et `B` lorsque l'exécution de `B` peut suivre celle de `A`

Définition

Définitions

Le graphe de flot de contrôle représente les exécutions possibles d'un programme.

- Les sommets `E` et `S` représentent l'entrée et la sorties
- Les autres noeuds sont les blocs d'instructions
- On dessine un arc entre deux noeuds `A` et `B` lorsque l'exécution de `B` peut suivre celle de `A`
- Les noeuds quittant les instructions conditionnelle sont étiquetés par `V` ou `F`.

Définition

Définitions

Le graphe de flot de contrôle représente les exécutions possibles d'un programme.

- Les sommets `E` et `S` représentent l'entrée et la sorties
- Les autres noeuds sont les blocs d'instructions
- On dessine un arc entre deux noeuds `A` et `B` lorsque l'exécution de `B` peut suivre celle de `A`
- Les noeuds quittant les instructions conditionnelle sont étiquetés par `V` ou `F`.
- On dira qu'un chemin dans le graphe est faisable lorsqu'il existe des valeurs d'entrées pour lesquels l'exécution passe par tous les noeuds de ce chemin.

Définition

Définitions

Le graphe de flot de contrôle représente les exécutions possibles d'un programme.

- Les sommets `E` et `S` représentent l'entrée et la sorties
- Les autres noeuds sont les blocs d'instructions
- On dessine un arc entre deux noeuds `A` et `B` lorsque l'exécution de `B` peut suivre celle de `A`
- Les noeuds quittant les instructions conditionnelle sont étiquetés par `V` ou `F`.
- On dira qu'un chemin dans le graphe est faisable lorsqu'il existe des valeurs d'entrées pour lesquels l'exécution passe par tous les noeuds de ce chemin.

Définition

On dit qu'un jeu de test couvre tous les sommets (resp. tous les arcs) lorsque son exécution permet de passer par tous les noeuds (resp. tous les arcs)

Exemple

1. Montrer que Le jeu de test `{n=1, n=8}` ne couvre ni tous les arcs, ni tous les sommets.

Exemple

1. Montrer que Le jeu de test `{n=1, n=8}` ne couvre ni tous les arcs, ni tous les sommets.
   On ne passe jamais par le noeud représentant l'instruction `n=3*n+1`, en effet pour `n=1` on atteint directement la sortie S et pour `n=8`, tous les valeurs de la suite sont paires.

Exemple

1. Montrer que Le jeu de test `{n=1, n=8}` ne couvre ni tous les arcs, ni tous les sommets.
   On ne passe jamais par le noeud représentant l'instruction `n=3*n+1`, en effet pour `n=1` on atteint directement la sortie `S` et pour `n=8`, tous les valeurs de la suite sont paires.
2. Proposer un jeu de test permettant de couvrir tous les arcs.

Exemple

1. Montrer que Le jeu de test `{n=1, n=8}` ne couvre ni tous les arcs, ni tous les sommets.
   On ne passe jamais par le noeud représentant l'instruction `n=3*n+1`, en effet pour `n=1` on atteint directement la sortie S et pour `n=8`, tous les valeurs de la suite sont paires.
2. Proposer un jeu de test permettant de couvrir tous les arcs.
   On peut ajouter par exemple le test `n=5`

Exercice

Exemple

1. Montrer que Le jeu de test `{n=1, n=8}` ne couvre ni tous les arcs, ni tous les sommets.
   On ne passe jamais par le noeud représentant l'instruction `n=3*n+1`, en effet pour `n=1` on atteint directement la sortie S et pour `n=8`, tous les valeurs de la suite sont paires.
2. Proposer un jeu de test permettant de couvrir tous les arcs.
   On peut ajouter par exemple le test `n=5`

Exercice

1. Ecrire une fonction prenant en argument un flottant `x` et un entier positif ou nuln et qui renvoie `x` puissance `n`.

Exemple

1. Montrer que Le jeu de test `{n=1, n=8}` ne couvre ni tous les arcs, ni tous les sommets.
   On ne passe jamais par le noeud représentant l'instruction `n=3*n+1`, en effet pour `n=1` on atteint directement la sortie S et pour `n=8`, tous les valeurs de la suite sont paires.
2. Proposer un jeu de test permettant de couvrir tous les arcs.
   On peut ajouter par exemple le test `n=5`

Exercice

1. Ecrire une fonction prenant en argument un flottant `x` et un entier positif ou nuln et qui renvoie `x` puissance `n`.
2. Tracer son graphe de flot de contrôle.

Exemple

1. Montrer que Le jeu de test `{n=1, n=8}` ne couvre ni tous les arcs, ni tous les sommets.
   On ne passe jamais par le noeud représentant l'instruction `n=3*n+1`, en effet pour `n=1` on atteint directement la sortie S et pour `n=8`, tous les valeurs de la suite sont paires.
2. Proposer un jeu de test permettant de couvrir tous les arcs.
   On peut ajouter par exemple le test `n=5`

Exercice

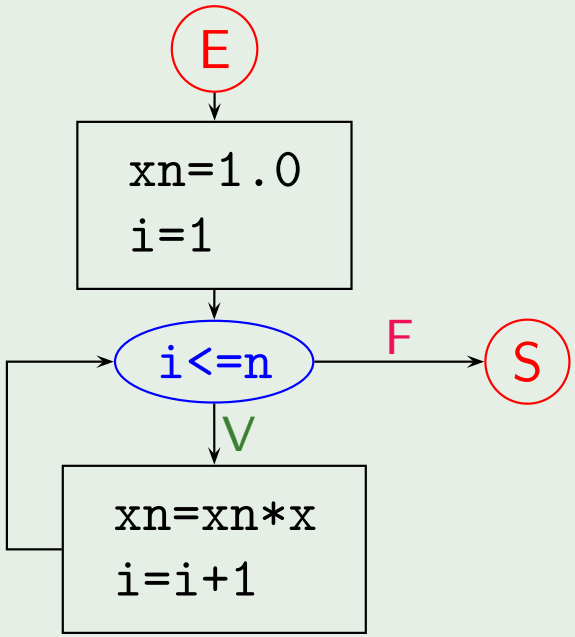
1. Ecrire une fonction prenant en argument un flottant `x` et un entier positif ou nuln et qui renvoie `x` puissance `n`.
2. Tracer son graphe de flot de contrôle.
3. Proposer un jeu de test.

Correction

```
1  double puissance(double x, int n){
2      assert (n>=0);
3      double xn = 1.0;
4      for (int i=1;i<=n;i++) {
5          xn = xn * x;}
6      return xn;}
```

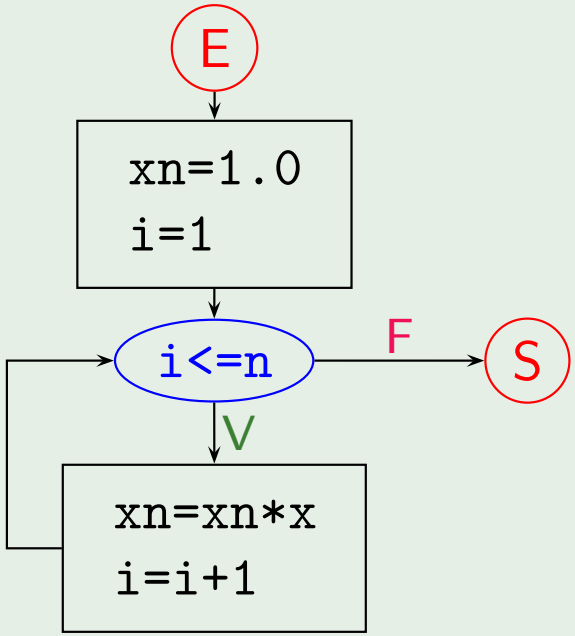
Correction

```
1  double puissance(double x, int n){
2      assert (n>=0);
3      double xn = 1.0;
4      for (int i=1;i<=n;i++) {
5          xn = xn * x;}
6      return xn;}
```



Correction

```
1  double puissance(double x, int n){
2      assert (n>=0);
3      double xn = 1.0;
4      for (int i=1;i<=n;i++) {
5          xn = xn * x;}
6      return xn;}
```

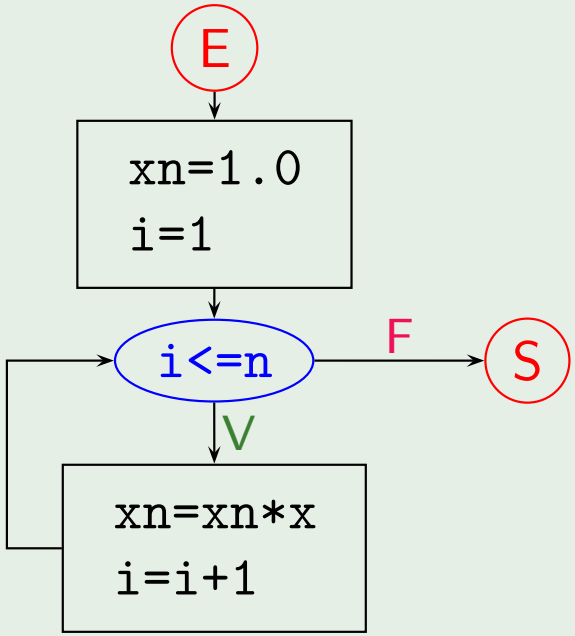


On peut proposer le jeu de test suivant :

- x=2.0, n=0

Correction

```
1  double puissance(double x, int n){
2      assert (n>=0);
3      double xn = 1.0;
4      for (int i=1;i<=n;i++) {
5          xn = xn * x;}
6      return xn;}
```



On peut proposer le jeu de test suivant :

- x=2.0, n=0
- x=2.0, n=3

Remarque

Lorsqu'une instruction conditionnelle comporte des conjonctions ou des disjonctions, on peut formuler des tests qui prévoient toutes les possibilités de la satisfaire.

Exemple

Une année est bissextile si elle est divisible par 4 mais pas par 100 ou s'il est divisible par 400.

1. Ecrire une fonction bissextile qui prend en argument un entier positif `annee` et renvoie `true` si l'année est bissextile et `false` sinon.

Remarque

Lorsqu'une instruction conditionnelle comporte des conjonctions ou des disjonctions, on peut formuler des tests qui prévoient toutes les possibilités de la satisfaire.

Exemple

Une année est bissextile si elle est divisible par 4 mais pas par 100 ou s'il est divisible par 400.

1. Ecrire une fonction bissextile qui prend en argument un entier positif `annee` et renvoie `true` si l'année est bissextile et `false` sinon.
2. Proposer un jeu de test pour cette fonction qui prévoit toutes les possibilités de satisfaire la condition d'être bissextile.

Correction

1.

```
1  bool bissextile(int annee) {
2    return ((annee%4==0 && annee%100!=0) || (annee%400=0))
3  }
```

Correction

1.

```
1  bool bissextile(int annee) {
2    return ((annee%4==0 && annee%100!=0) || (annee%400=0))
3  }
```

2. On peut proposer les tests suivants :

Correction

1.
```
1  bool bissextile(int annee) {
2    return ((annee%4==0 && annee%100!=0) || (annee%400=0))
3  }
```

2. On peut proposer les tests suivants :
   - `n=2004` qui permet de valider la première condition

Correction

1.
```
1  bool bissextile(int annee) {
2    return ((annee%4==0 && annee%100!=0) || (annee%400=0))
3  }
```

2. On peut proposer les tests suivants :
   - `n=2004` qui permet de valider la première condition
   - `n=2000` qui permet de valider la deuxième