

### 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;}
```

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

### Correction

- ❶ 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 occurrence du maximum des éléments du tablea

### Définition

### Correction

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

### Définition

### Correction

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

### Définition

### Correction

- ❶ La fonction renvoie 1 (indice de l'élément maximal du tableau)
- ❷ la fonction renvoie 3, c'est l'indice de la dernière occurrence du maximum des éléments du tableau
- ❸ C'est un comportement indéfini, la variable `imax` n'est pas initialisée.
- ❹ 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

- ❶ La fonction renvoie 1 (indice de l'élément maximal du tableau)
- ❷ la fonction renvoie 3, c'est l'indice de la dernière occurrence du maximum des éléments du tableau
- ❸ C'est un comportement indéfini, la variable `imax` n'est pas initialisée.
- ❹ 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 donner une description formelle et détaillée de ses caractéristiques. En particulier :

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



### Correction

- ❶ La fonction renvoie 1 (indice de l'élément maximal du tableau)
- ❷ la fonction renvoie 3, c'est l'indice de la dernière occurrence du maximum des éléments du tableau
- ❸ C'est un comportement indéfini, la variable `imax` n'est pas initialisée.
- ❹ 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 donner 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 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 tableau *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 tableau *non vide* d'entiers.
- Renvoie l'indice de la première occurrence du maximum des éléments de ce tableau.

### Correction

```
1  #include <assert.h>
2
3  int get_imax(int tab[],int size) {
4      /* Prend en argument un tableau non vide d'entiers
5       Renvoie l'indice de la 1ere occurrence du maximum du tableau*/
6      assert (size > 0);
7      int max = tab[0];
8      int imax = 0;
9      for (int i=0; i<size; i++) {
10         if (tab[i] > max) {
11             max = tab[i];
12             imax = i;}}
13     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 où 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 où 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 où 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){  
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



### 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
  - Élément 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
  - Élément présent en plus de deux exemplaires consécutifs
  - Présence du double en tout début ou toute fin de tableau

## Définition

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

## Exemple

Tracer le graphe de flot de contrôle de la fonction suivante :

```
1 double puissance(double x, int n){  
2     /* prends en argument un flottant x et un entier n positif ou  
   ↪ nul et renvoie x puissance n*/  
3     assert (n>=0);  
4     double xn = 1.0;  
5     for (int i=1;i<=n;i++) {  
6         xn = xn * x;}  
}
```

## Définition

- Le graphe de flot de contrôle d'un programme représente les exécutions possibles de ce programme.
- Les noeuds de ce graphe sont les blocs d'instructions ou les tests.

## Exemple

Tracer le graphe de flot de contrôle de la fonction suivante :

```
1 double puissance(double x, int n){  
2     /* prends en argument un flottant x et un entier n positif ou  
   ↪ nul et renvoie x puissance n*/  
3     assert (n>=0);  
4     double xn = 1.0;  
5     for (int i=1;i<=n;i++) {  
6         xn = xn * x;}  
}
```

## Définition

- Le graphe de flot de contrôle d'un programme représente les exécutions possibles de ce programme.
- Les noeuds de ce graphe sont les blocs d'instructions ou les tests.
- Un arc entre deux noeuds  $n1$  et  $n2$  indique que l'exécution de  $n2$  peut suivre celle de  $n1$

## Exemple

Tracer le graphe de flot de contrôle de la fonction suivante :

```
1 double puissance(double x, int n){  
2     /* prends en argument un flottant x et un entier n positif ou  
   ↪ nul et renvoie x puissance n*/  
3     assert (n>=0);  
4     double xn = 1.0;  
5     for (int i=1;i<=n;i++) {  
6         xn = xn * x;}  
}
```

### Définition

- Le graphe de flot de contrôle d'un programme représente les exécutions possibles de ce programme.
- Les noeuds de ce graphe sont les blocs d'instructions ou les tests.
- Un arc entre deux noeuds  $n1$  et  $n2$  indique que l'exécution de  $n2$  peut suivre celle de  $n1$
- On marque les noeuds spéciaux représentant la sortie et l'entrée.

### Exemple

Tracer le graphe de flot de contrôle de la fonction suivante :

```
1 double puissance(double x, int n){  
2     /* prends en argument un flottant x et un entier n positif ou  
   ↪ nul et renvoie x puissance n*/  
3     assert (n>=0);  
4     double xn = 1.0;  
5     for (int i=1;i<=n;i++) {  
6         xn = xn * x;}  
}
```