Bases de la programmation

Séance 5

Découpe en sous-problèmes et librairie standard

Sébastien Combéfis

mercredi 8 octobre 2014

Rappels du cours précédent

- Représentation des flottants en binaire (IEEE 754)
- Procédures (avec et sans paramètres)

  *Représente une action à exécuter (type **void**)*
- Fonctions (avec et sans paramètres)

  Représente une opération à calculer
- Prototype pour les procédures et les fonctions
- L'instruction **return**

Analyse et résolution d'un problème

1. Identification précise du problème
2. Spécification du problème
3. Implémentation de tests

   Les tests doivent couvrir les cas décrits par les spécifications

4. Décomposition en sous-problèmes

   Répéter l'analyse pour chaque sous-problème

5. Implémentation dans un langage de programmation

Spécifications

- Les spécifications décrivent précisément ce que fait un algorithme

  Conditions qui doivent être remplies pour avoir le résultat attendu

- Trois éléments à fournir

  Entrée Paramètres, conditions qui doivent être remplies, et état du programme

  Sortie Ce que la fonction produit comme résultat

  Effet de bord Ce que la procédure ou fonction a modifié sur les paramètres et sur l'état du programme

Rechercher un élément dans un tableau I

1. Identification du problème

- Retrouver un élément donné dans un tableau d'entiers
- On s'intéresse à l'indice de l'élément dans le tableau
- Si l'élément apparait plusieurs fois, on cherche le dernier indice
- Par exemple, soit le tableau [1 8 9 2 1 4]
  - Renvoie -1 pour 7 : élément pas dans le tableau
  - Renvoie 2 pour 9 : [1 8 9 2 1 4]
  - Renvoie 4 pour 1 : [1 8 9 2 1 4]

Rechercher un élément dans un tableau II

2. Les spécifications

Entrée
- Un tableau d'entiers tab de taille N > 0
- Un nombre entier e

Sortie
- Si l'élément e se trouve dans le tableau, renvoie l'indice de la dernière occurrence de l'élément dans le tableau
- Sinon, renvoie -1

Effet de bord —

Tests

- Un jeu de tests permet de tester plusieurs cas

  On plus on a de tests, au mieux c'est

- Établi à partir des spécifications, sans avoir l'implémentation

  Les tests doivent être indépendants de l'implémentation

- Doit couvrir le plus de cas possibles, notamment les cas limites

  On ne pourra jamais tester qu'un programme est correct

Rechercher un élément dans un tableau III

3. Les tests

- e n'est pas dans le tableau
- e apparait une fois, au début ou à la fin
- e apparait une fois, au milieu
- e apparait plusieurs fois

```
tab = [1, 2, 3, 2, 4, 5]   et   e = 7          R = −1

tab = [1, 2, 3, 2, 4, 5]   et   e = 1          R = 0
tab = [1, 2, 3, 2, 4, 5]   et   e = 5          R = 5

tab = [1, 2, 3, 2, 4, 5]   et   e = 4          R = 4

tab = [1, 2, 3, 2, 4, 5]   et   e = 2          R = 3
```

Décomposition en sous-problèmes

- Décomposer un problème en sous-problèmes

  Les sous-problèmes doivent être plus petits et faciles à résoudre

- Solution des sous-problèmes à combiner

  Pour construire la solution au problème général

- Chaque sous-problème est aussi défini avec des spécifications

Rechercher un élément dans un tableau IV

4. Décomposition en sous-problèmes

- Pas besoin de décomposer en sous-problèmes
- Problème principal suffisamment simple

  Trouver le plus grand indice d'un élément `e` *dans un tableau* `tab`

Analyse de la boucle

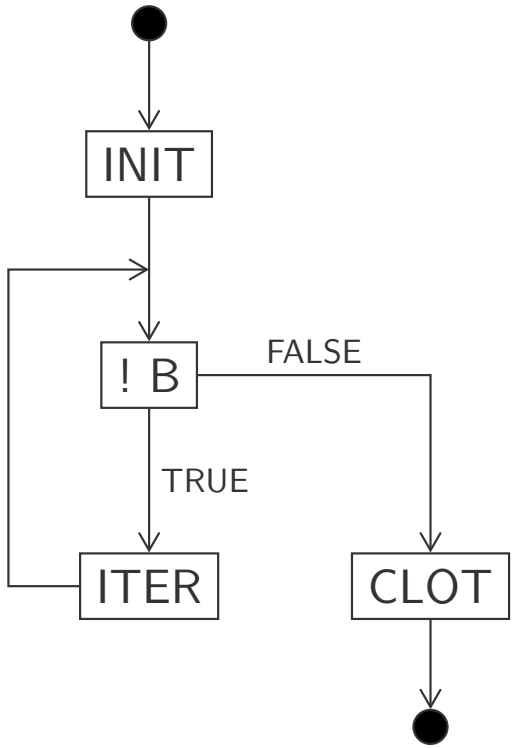
- Quatre parties à une boucle

  INIT **Initialisation**, une fois avant le début de la boucle

  B **Condition d'arrêt** de la boucle

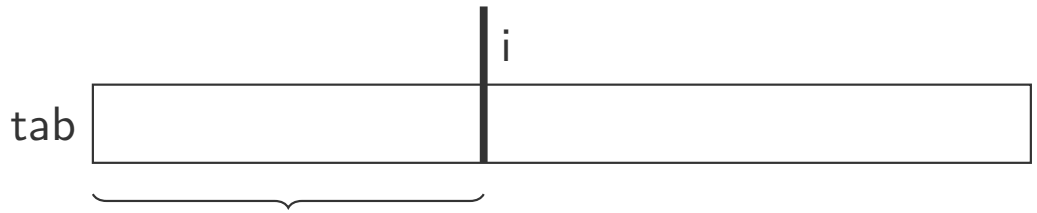
  ITER **Itération**, répétée par la boucle

  CLOT **Clôture**, une fois après la boucle



Rechercher un élément dans un tableau V

5. Analyse de la boucle



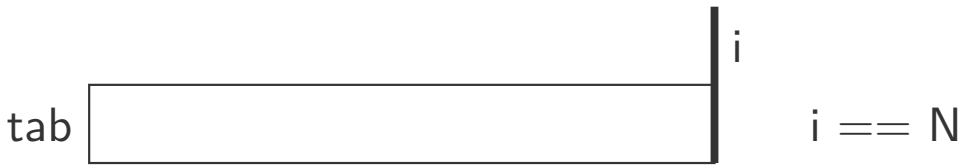
pos vaut -1 si e ne se trouve pas dans la partie déjà explorée du tableau. Sinon, pos contient le plus grand indice $<$ i tel que tab[pos] $=$ e

- Situation initiale

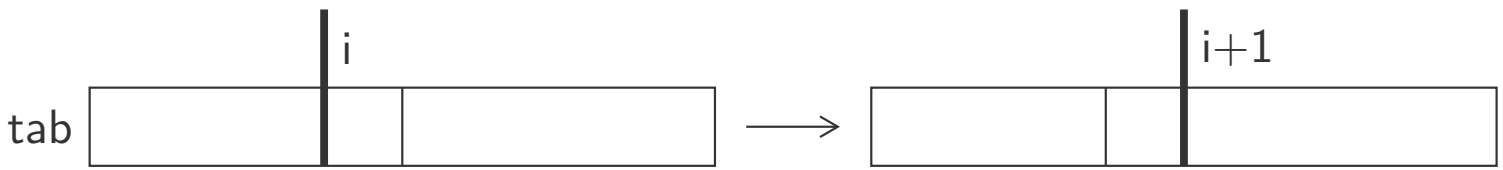
$i = 0$
$pos = -1$

Rechercher un élément dans un tableau VI

- Situation finale



- Itération



Soit tab[i] == e : on doit faire pos = i

Soit tab[i] != e : on ne doit rien faire

Rechercher un élément dans un tableau VII

INIT

```
int i = 0;
int pos = -1;
```

B

```
i == N
```

ITER

```
if (tab[i] == e)
{
    pos = i;
}
i++;
```

CLOT

```
return pos;
```

Rechercher un élément dans un tableau VIII

6. Code final

```
1  int findElem (int tab[], int N, int e)
2  {
3      int i = 0;
4      int pos = -1;
5
6      while (! (i == N)) // ou alors : while (i != N)
7      {
8          if (tab[i] == e)
9          {
10             pos = i;
11         }
12         i++;
13     }
14
15     return pos;
16 }
```

Trouver un sous-tableau I

- Étant donné deux tableaux A et B, A est un sous-tableau B si chacun des éléments de A se retrouve dans B

Entrée	• Un tableau d'entiers A de taille N ≥ 0
	• Un tableau d'entiers B de taille M ≥ 0
Sortie	• TRUE si A est un sous-tableau de B et FALSE sinon
Effet de bord	—

```
bool isSubtab (int A[], int N, int B[], int M);
```

Trouver un sous-tableau II

- A ou B est vide
- A est un sous-tableau
- A n'est pas un sous-tableau
- A ou B contient plusieurs fois un même élément

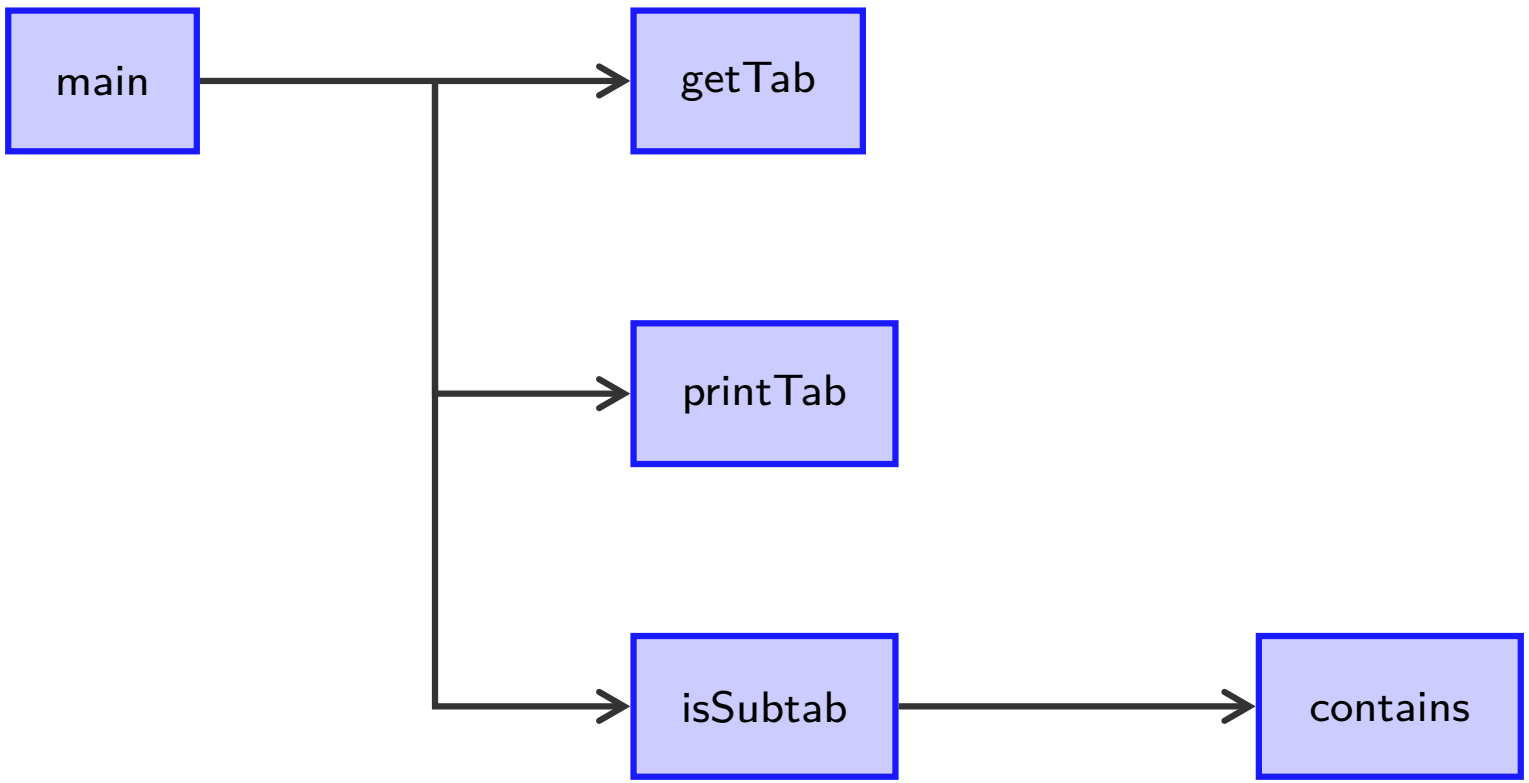
```
A = []              et  B = []                    R = TRUE
A = [1, 2, 3]       et  B = []                    R = FALSE
A = []              et  B = [1, 2, 3]             R = TRUE

A = [1, 3, 4]       et  B = [5, 1, 2, 4, 3]       R = TRUE
A = [1, 3, 4]       et  B = [5, 2, 4, 3]          R = FALSE

A = [1, 4, 1, 4]    et  B = [1, 4]                R = TRUE
A = [7, 2]          et  B = [2, 8, 2, 2]          R = FALSE
```

Trouver un sous-tableau III

- Décomposition en cinq sous-problèmes



La fonction contains I

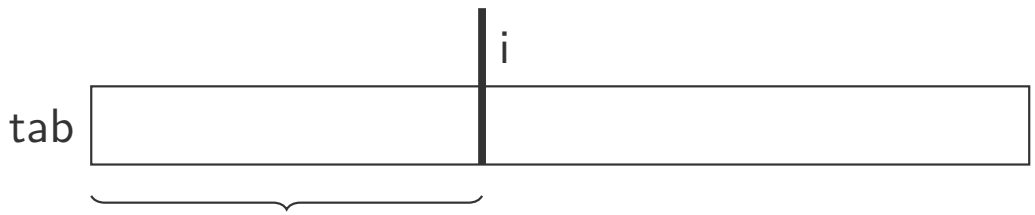
- Cherche si un élément donné se trouve ou non dans un tableau

Entrée
- Un tableau d'entiers tab de taille N ≥ 0
- Un entier e

Sortie
- TRUE si e se trouve dans tab et FALSE sinon

Effet de bord —

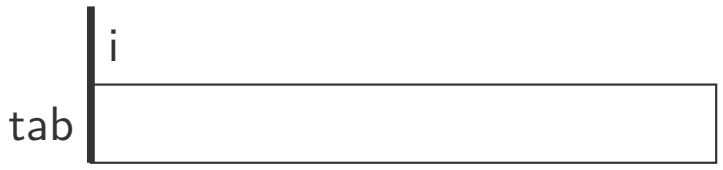
```
bool contains (int tab [], int N, int e);
```



found vaut TRUE si e se trouve dans la partie déjà explorée du tableau et FALSE sinon

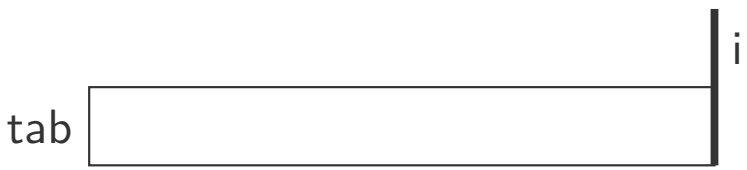
La fonction `contains` II

- Situation initiale



i = 0
found = FALSE

- Situation finale (premier cas)

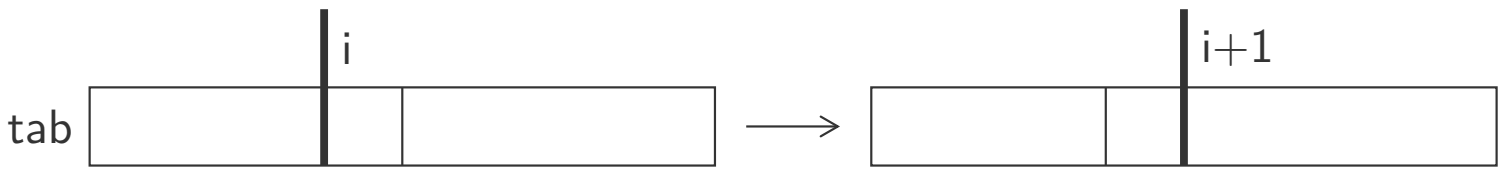


i == N

La fonction `contains` III

- Situation finale (second cas) :

- Itération :



Soit `tab[i] == e` : on doit faire `found = TRUE`

Soit `tab[i] != e` : on ne doit rien faire

La fonction contains IV

INIT

```
int i = 0;
bool found = FALSE;
```

B

```
i == N || found == TRUE
```

ITER

```
if (tab[i] == e)
{
    found = TRUE;
}
i++;
```

CLOT

```
return found;
```

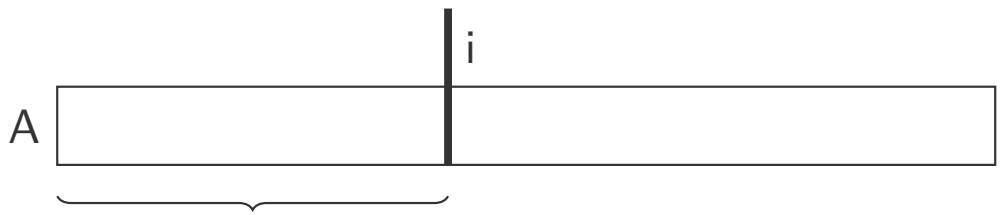
La fonction contains V

```
1  bool contains (int tab[], int N, int e)
2  {
3      int i = 0;
4      bool found = FALSE;
5      while (! (i == N || found == TRUE))
6      {
7          if (tab[i] == elem)
8          {
9              found = TRUE;
10         }
11         i++;
12     }
13     return found;
14 }
```

- $[\text{found} == \text{TRUE}] \equiv [\text{found}]$
- $[\text{found} == \text{FALSE}] \equiv [!\ \text{found}]$
- $[!\ (\text{i} == \text{N} \mid\mid \text{found})] \equiv [\text{i} != \text{N} \&\& !\ \text{found}]$

La fonction `isSubTab` I

- Teste si un tableau est un sous-tableau d'un autre



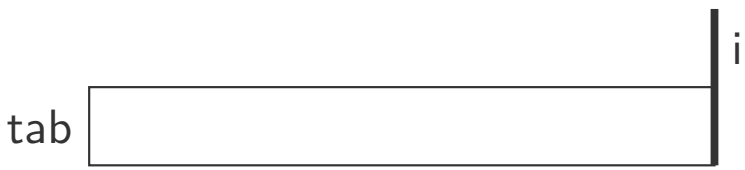
inB vaut TRUE si tous les éléments déjà parcourus se retrouvent dans B et FALSE sinon

La fonction `isSubTab` II

- Situation initiale

i = 0
inB = TRUE

- Situation finale (premier cas)

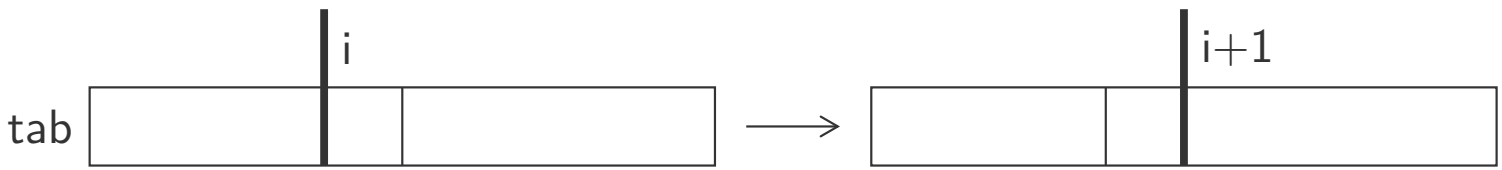


i == N

La fonction `isSubTab` III

- Situation finale (second cas)

- Itération



Soit tab[i] ne se trouve pas dans B : on doit faire inB = FALSE

Soit tab[i] se trouve dans B : on ne doit rien faire

La fonction isSubTab IV

INIT

```
int i = 0;
bool inB = TRUE;
```

B

```
i == N || inB == FALSE
```

ITER

```
if (! contains (B, M, A[i]))
{
    inB = FALSE;
}
i++;
```

CLOT

```
return inB;
```

La fonction isSubTab V

```
1  bool isSubtab (int A[], int N, int B[], int M)
2  {
3      int i = 0;
4      bool inB = TRUE;
5      while (! (i == N || inB == FALSE))
6      {
7          if (! contains (B, M, A[i]))
8          {
9              inB = FALSE;
10         }
11         i++;
12     }
13     return inB;
14 }
```

- $[!\ (i == N \mid\mid !\ inB)] \equiv [i\ != N\ \&\&\ inB]$

La librairie standard

- Collection de prototypes pour des opérations courantes

  Fonctions mathématiques, manipulation de chaines de caractères entrées/sorties, manipulation de types...

- Fichiers `.h` à inclure dans son programme

ctype.h

- **int** isalpha (**int** c);

  Teste si c est un caractère alphanumérique

- **int** isdigit (**int** c);

  Teste si c est un caractère numérique

- **int** isupper (**int** c);

  Teste si c est un caractère alphabétique majuscule

`float.h` et `limits.h`

- Série de constantes liées aux nombres flottants et entiers
- Valeurs maximales et minimales pour les **float**

  FLT_MIN, *FLT_MAX*
- Valeurs maximales et minimales pour les **int**

  INT_MIN, *INT_MAX*

math.h

- Fonctions mathématiques
- **double** cos (**double** x);

  sin, *tan*, *log*, *log10*, *sqrt*, *pow*, *abs*...
- Arrondi

  ceil, *floor*, *round*...