Bases de la programmation

Séance 3

Les tableaux à une dimension

Sébastien Combéfis

mercredi 24 septembre 2014

Rappels du cours précédent

- Les différents types de données
- Représentation binaire des entiers

  Binaire, bit de signe, complément à deux
- Les types **int**, **float** et **char**
- Occupation mémoire d'un type de données avec **sizeof**
- limits.h, approfondissement de printf, opérateur modulo
- Génération de nombres pseudo-aléatoires
- Opérateurs logiques
- Opérateurs binaires

Préprocesseur

- Le préprocesseur fait un premier traitement sur le code source
- Transformation d'un code source en un autre code source

Instructions préprocesseur

- Instruction **#include**

  Inclus complètement un autre fichier

```
#include <chemin_fichier>
```

- Instruction **#define**

  Remplace une chaine de caractères par une autre

```
#define TOKEN valeur
```

Définir une constante I

- Une constante est une valeur qui ne change pas
- **#define** NOM_CONSTANTE Valeur

```c
#define TVA 1.21

int main()
{
    float price = 12.5;                 // Prix de l'article
    float total = price * TVA;

    return 0;
}
```

Définir une constante II

- Le préprocesseur traite le fichier source
  - TVA a été remplacé par 1.21
  - Le commentaire a été supprimé

```
1  int main()
2  {
3      float price = 12.5;
4      float total = price * 1.21;
5
6      return 0;
7  }
```

Définir un nouveau type de données

- Créer un nouveau type sur base d'un existant

```
typedef type_existant nouveau_type
```

- printf s'utilise comme pour le type original

```c
#include <stdio.h>

typedef int entier;          // Crée un type "entier"

int main()
{
    entier a = 10;
    printf ("a vaut %d\n", a);

    return 0;
}
```

Création d'un type booléen

```c
#include <stdio.h>

#define TRUE 1
#define FALSE 0

typedef int bool;

int main()
{
    bool alive = TRUE;
    printf ("alive vaut %d\n", alive);     // Affiche 1

    return 0;
}
```

- Crée un type booléen avec **typedef**

Création d'un type booléen

```c
#include <stdio.h>

#define TRUE 1
#define FALSE 0

typedef int bool;

int main()
{
    bool alive = TRUE;
    printf ("alive vaut %d\n", alive);     // Affiche 1

    return 0;
}
```

- Associe 1 et TRUE avec **#define**

Création d'un type booléen

```c
#include <stdio.h>

#define TRUE 1
#define FALSE 0

typedef int bool;

int main()
{
    bool alive = TRUE;
    printf ("alive vaut %d\n", alive);     // Affiche 1

    return 0;
}
```

- Associe 0 et FALSE avec **#define**

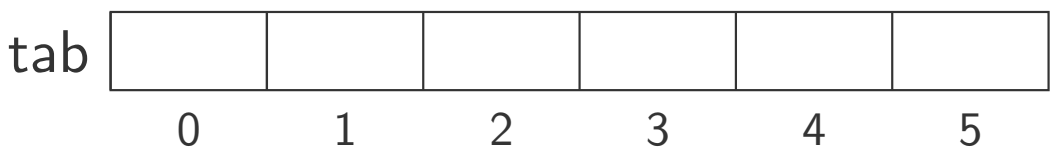
Création d'un type booléen

```c
#include <stdio.h>

#define TRUE 1
#define FALSE 0

typedef int bool;

int main()
{
    bool alive = TRUE;
    printf ("alive vaut %d\n", alive);     // Affiche 1

    return 0;
}
```

- Déclaration d'une variable de type bool

Tableaux

- Un tableau se définit par
  - Un type (**int**, **float**, **char**...)
  - Une taille N
- `tab[i]` est la case du tableau à l'indice `i` (de 0 à $N-1$)
- Exemple d'un tableau de taille 6



Opérations sur un tableau

- `int tab[N];`

  Déclare un tableau d'entiers de taille N

- `tab[i] = 3;`

  Stocke la valeur 3 à la case d'indice i

```
1  int N = 3;                  // Taille du tableau
2  int tab[N];                 // Déclaration du tableau d'entiers
3
4  int i;
5  for (i = 0; i < N; i++)
6  {
7      tab[i] = i + 1;         // Stocke des valeurs dans le tableau
8  }
```

Initialisation d'un tableau

- Méthode d'initialisation directe d'un tableau

  Uniquement lors de la déclaration

```
1  int N = 3;                // Taille du tableau
2  int tab[N] = {1, 2}; // Déclaration du tableau d'entiers
3
4  int tab2[5] = {5, 4, 3, 2, 1};
5  int tab3[] = {1, 2, 3, 4};
6
7  // tab3 = {2, 3, 4, 5}     => INTERDIT
```

Algorithme

- Un algorithme permet de résoudre un problème
- On spécifie un algorithme avec trois éléments
  - Les entrées
  - Les sorties
  - Les éventuels effets de de bords

Afficher les éléments d'un tableau I

- Étant donné un tableau d'entiers tab, on désire afficher tous ses éléments à l'écran, de manière esthétique

```
Entrée : • Un tableau d'entiers tab de taille N (≥ 0)
Sortie : • Une représentation des éléments du tableau de
           la forme [valeur_1, valeur_2, ..., valeur_N]
Effet de bord : –
```

Afficher les éléments d'un tableau II

```c
if (N == 0) // Cas de base, si le tableau est vide
{
    printf ("[]");
}
else
{
    printf ("[%d", tab[0]); // Affichage du 1er élément

    int i;
    for (i = 1; i < N; i++)
    {
        printf (", %d", tab[i]); // Si plusieurs éléments
    }
    printf ("]\n");
}
```

Réinitialiser un tableau I

- Étant donné un tableau d'entiers tab, on désire écrire 0 dans toutes ses cases

```
Entrée : • Un tableau d'entiers tab de taille N (≥ 0)
Sortie : –
Effet de bord : • Toutes les cases du tableau valent zéro
```

Réinitialiser un tableau II

```
1  int i;
2  for (i = 0; i < N; i++)
3  {
4      tab[i] = 0;
5  }
```

Recherche un élément dans un tableau I

- Étant donné un tableau d'entiers tab, on veut vérifier la présence d'un élément dedans et le signaler à l'aide d'une phrase

```
Entrée : • Un tableau d'entiers tab de taille N (≥ 0)
         • Une valeur entière seek
Sortie : • Affiche "trouvé" si seek se trouve au moins
           une fois dans tab "pas trouvé" sinon
Effet de bord : –
```

Rechercher un élément dans un tableau II

```
1  int occurrence = 0;
2
3  int i;
4  for (i = 0; i < N; i++)
5  {
6      if (tab[i] == seek)
7      {
8          occurrence++;
9      }
10 }
11
12 if (occurrence == 0)
13 {
14     printf("pas trouvé");
15 }
16 else
17 {
18     printf("trouvé");
19 }
```

Efficacité de l'algorithme `contains`

- L'algorithme précédent est-il vraiment efficace ?
- Combien d'opérations fera-t-il si :
  - tab $= [2,\ 3,\ 4,\ 5]$ et seek $= 3$?
  - tab $= [1,\ 0,\ 0,\ 0]$ et seek $= 1$?
  - tab $= [1,\ 1,\ 1,\ 4]$ et seek $= 4$?
  - tab $= [1,\ 2,\ 3,\ 4]$ et seek $= 0$?

Rechercher un élément dans un tableau III

```
1  int occurrence = 0;
2
3  int i;
4  for (i = 0; (i < N) && (occurrence == 0); i++)
5  {
6      if (tab[i] == seek)
7      {
8          occurrence++;
9      }
10 }
11 if (occurrence == 0)
12 {
13     printf ("trouvé");
14 }
15 else
16 {
17     printf ("pas trouvé");
18 }
```

Tableau trié

- Un tableau peut être trié de deux façons
  - En ordre croissant

    $[1,\ 2,\ 3,\ 4,\ 5]$
  - En ordre décroissant

    $[5,\ 4,\ 3,\ 2,\ 1]$
- Un tableau trié permet des recherches beaucoup plus rapides

Recherche dans un tableau trié I

- Étant donné un tableau d'entiers tab, on veut vérifier la présence d'un élément dedans et le signaler à l'aide d'une phrase

```
Entrée : • Un tableau d'entiers trié en ordre croissant
           tab de taille N (≥ 0)
         • Une valeur entière seek
Sortie : • Affiche "trouvé" si seek se trouve au moins
           une fois dans tab "pas trouvé" sinon
Effet de bord : –
```

Recherche dans un tableau trié II

```
1  int occurrence = 0;
2  int min = 0;
3  int max = N;
4  while (min != max && occurrence == 0)
5  {
6      int mid = (max + min) / 2;
7      if (tab[mid] == seek)   // Vérifie si seek est trouvé
8      {
9          occurrence++;
10     }
11     else if (tab[mid] < seek) // Si seek est plus grand
12     {
13         min = mid + 1;
14     }
15     else                      // Si seek est plus petit
16     {
17         max = mid;
18     }
19 }
```

Illustration de la recherche dichotomique

- Situation initiale

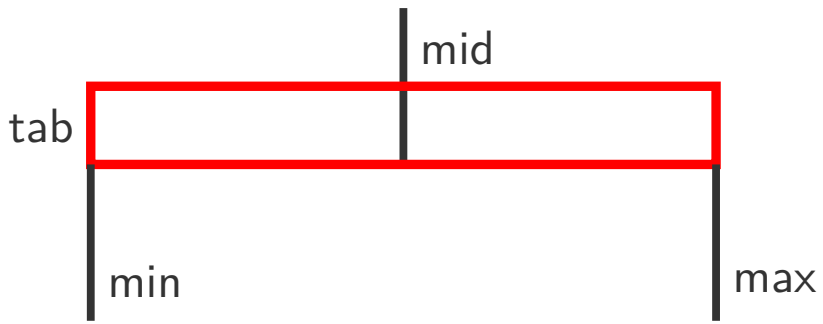


Illustration de la recherche dichotomique

- Si seek est plus grand que la valeur située à l'indice mid

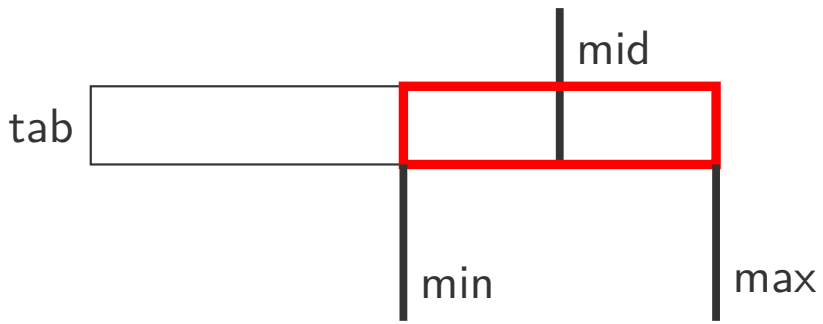


Illustration de la recherche dichotomique

- Si seek est plus petit que la valeur située à l'indice mid

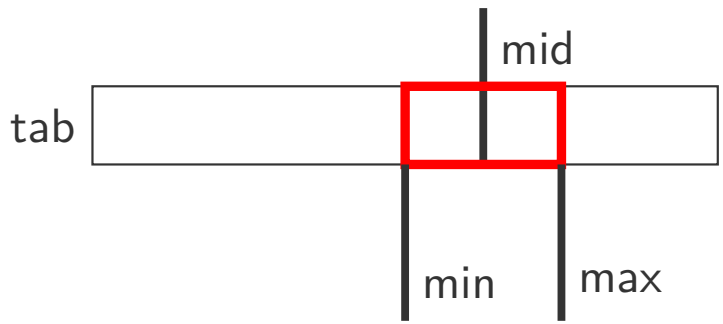


Illustration de la recherche dichotomique

- Si seek est égal à la valeur située à l'indice mid, alors seek a été trouvé et la recherche s'arrête
- Sinon la recherche continue à
  - droite si seek est plus grande que la valeur du milieu
  - gauche si seek est plus petite que la valeur du milieu