ESPACE ENSEIGNEMENT CARDINAL MERCIER

Bases de la programmation

Séance 8

Arithmétique des pointeurs et mémoire dynamique

Sébastien Combéfis

mercredi 19 novembre 2014

Rappels du cours précédent

- Structure de la mémoire
  - La pile (variables locales)
  - Le tas (variables globales)
- Pointeur
  - Accès à l'adresse d'une variable (&)
  - Accès à la valeur référencée par un pointeur ($*$)
- Mémoire dynamique
  - Demande d'espace mémoire (malloc)
  - Libération d'espace mémoire (free)

Lecture au clavier I

- On peut lire une valeur depuis la console avec scanf
- On passe l'adresse d'une variable que scanf va remplir

```c
#include <stdio.h>

int main()
{
    int value;

    printf ("Entrez un entier : ");
    scanf ("%d", &value);
    printf ("Vous avez entré %d !\n", value);

    return 0;
}
```

Lecture au clavier II

- On peut évidemment directement utiliser un pointeur

```c
#include <stdio.h>

int main()
{
    int *value = (int*) malloc (sizeof (int));

    printf ("Entrez un entier : ");
    scanf ("%d", value);
    printf ("Vous avez entré %d !\n", *value);

    free (value);
    return 0;
}
```

Allocation de mémoire dynamique

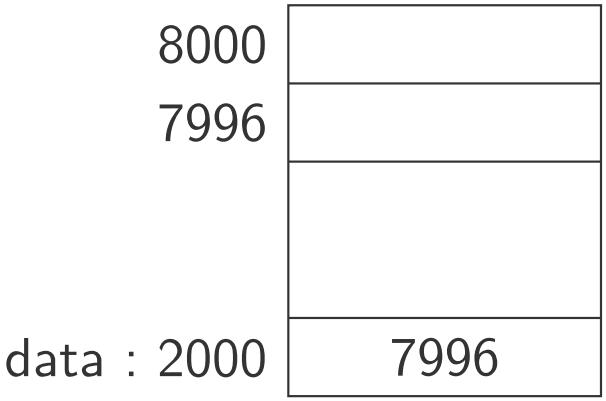
- La fonction malloc permet d'allouer un espace mémoire
- Renvoie l'adresse du début du bloc mémoire alloué
- En cas d'échec, renvoie un pointeur null

```
double *data = (double*) malloc (10 * sizeof (double));

if (data != NULL)
{
   // ... utilisation de la zone mémoire allouée ...
}
```

Déplacement dans la mémoire I

- L'opérateur + passe à la « case » suivante du pointeur

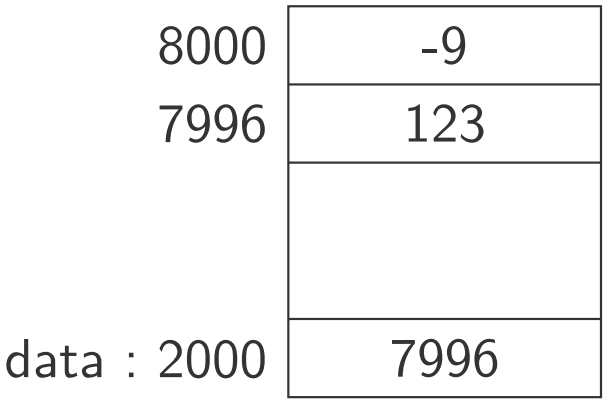
```c
int *data = (int*) malloc (2 * sizeof (int));

printf ("%p\n", data);          // Affiche 7996
printf ("%p\n", data + 1);      // Affiche 8000
```



Déplacement dans la mémoire II

- On peut modifier les valeurs en mémoire où on veut

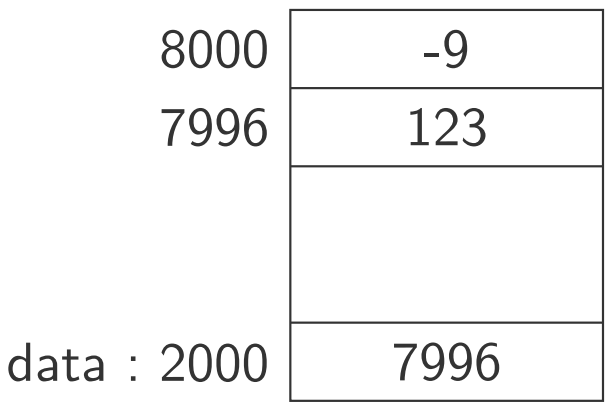
```
int *data = (int*) malloc (2 * sizeof (int));

*data = 123;
*(data + 1) = -9;
```



Dépassement de bornes

- Il faut faire attention aux accès illégaux à la mémoire

```
int *data = (int*) malloc (2 * sizeof (int));

printf ("%d\n", *(data + 5));
```



Nouveau regard sur les tableaux

- Un tableau est un bloc de mémoire
- Une variable tableau est un pointeur

```c
int tab[] = {1, 2, 3};

printf ("%p\n", tab);

printf ("%d\n", *tab);
printf ("%d\n", *(tab + 2));
```

2016	3
2012	2
2008	1
tab : 2000	2008

Équivalence de notations

- Soit la déclaration de variable **int** tab[] ,

  les notations suivantes sont équivalentes :

tab[i]	*(tab + i)
&tab[i]	tab + i

Initialisation d'un tableau

```c
void initTab (int tab[], int N)
{
    int i;
    for (i = 0; i < N; i++)
    {
        tab[i] = 0;
    }
}

int main()
{
    int size = 5;
    int data[size];

    initTab (data, size);

    return 0;
}
```

Initialisation d'un tableau

```c
void initTab (int tab[], int N)
{
    int i;
    for (i = 0; i < N; i++)
    {
        tab[i] = 0;
    }
}

int main()
{
    int size = 5;
    int data[size];

    initTab (data, size);

    return 0;
}
```

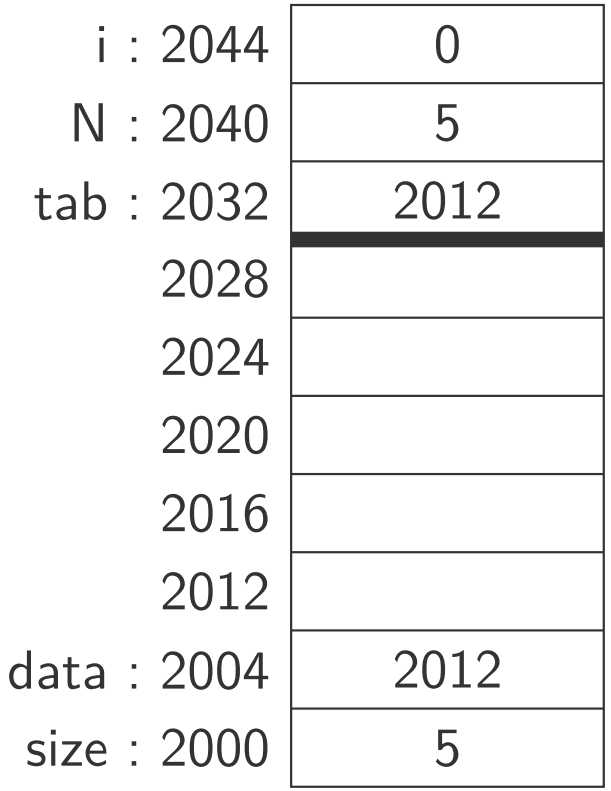


Tableau dynamique

- Un tableau dynamique est une zone mémoire constituée de N cases adjacentes et dont on possède un pointeur vers la première

```
int N = 5;
int *tab = (int*) malloc (N * sizeof (int));

int i;
for (i = 0; i < N; i++)
{
   tab[i] = i + 1;
}

free (tab);
```

Arithmétique des pointeurs I

- Navigation avec la notation [] et un indice
- Navigation avec le pointeur et l'arithmétique des pointeurs

```c
for (i = 0; i < N; i++)
{
    printf ("%d\n", tab[i]);
}
```

```c
for (i = 0; i < N; i++)
{
    printf ("%d\n", *(tab + i));
}
```

Arithmétique des pointeurs II

- Addition et soustraction sur un pointeur avec $+$ et $-$
- Incrémenter (décrémenter) un pointeur avec $++$ $(--)$
- Comparaison de pointeurs avec $==$, $<$, $>$, $<=$ et $>=$

```
int tab[] = {1, 2, 3, 4, 5};
int *start = &tab[0];
int *end = start + 4;

while (start <= end)
{
    printf ("%d\n", *start);
    start++;
}
```

Arithmétique des pointeurs III

- Un tableau est un pointeur constant
- Le code suivant provoque une *erreur de compilation*

```
int tab[] = {1, 2, 3, 4, 5};

tab = tab + 1;
```

Priorité des opérateurs

- Il faut bien faire attention à la priorité des opérateurs
- Soit la déclaration **int** $*$p;

(*p)++	*(p++)
*p + 2	*(p + 2)

Les fonctions `calloc` et `realloc`

- La fonction `calloc` crée une zone de mémoire initialisée à 0

```
double *tab = (double*) calloc (5, sizeof (double));
```

- La fonction `realloc` modifie la taille d'une zone de mémoire

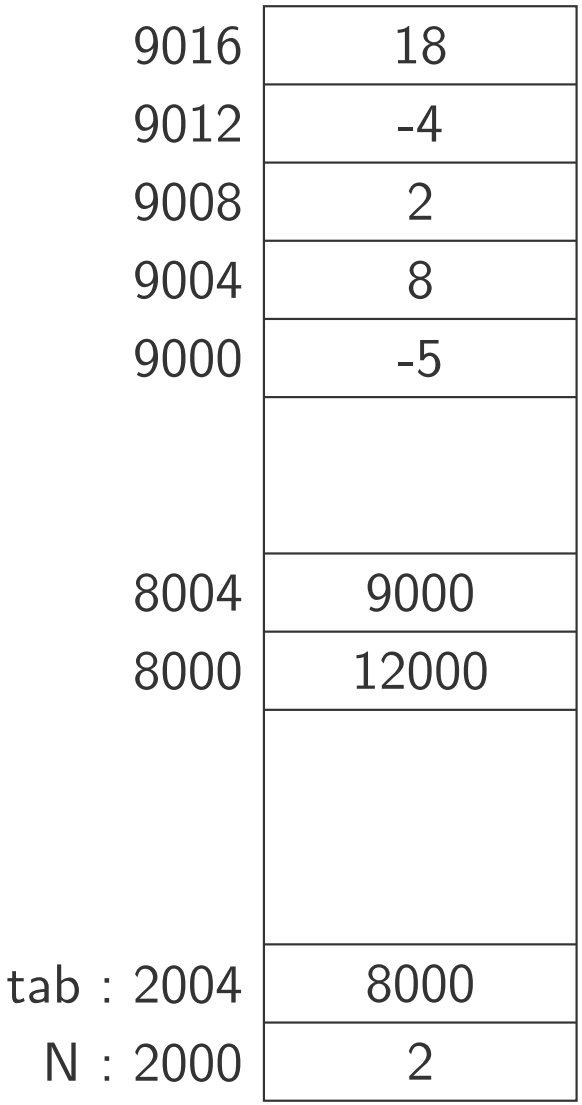
```
tab = (double*) realloc (tab, 10 * sizeof (double));
```

Fonction qui renvoie un tableau

- Une fonction qui renvoie un tableau doit le faire en mémoire dynamique

```
int* newTab (int size)
{
    return (int*) calloc (size, sizeof (int));
}
```

Exercice



Sachant qu'on a `int **tab`

1. `tab`
2. `*tab`
3. `**tab`
4. `&tab`
5. `*(tab + 1)`
6. `*(*(tab + 1) + 2)`
7. `*(*(tab + 2) + 1)`