Les pointeurs

DAKKAR Borhen-eddine

Lycée le Corbusier

BTS SN-IR

Table des matières

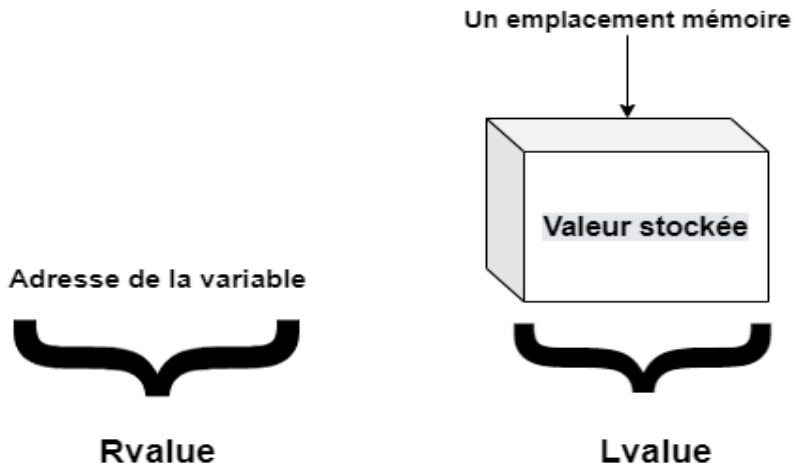
Table des matières

La puissance du langage C

- L'un des avantages de C est qu'il permet au programmeur d'accéder aux adresses de variables utilisées dans un programme.
- Nous ayons déjà utilisé l'opérateur d'adresse, &, en appelant la fonction scanf ().
- Le réel pouvoir d'utiliser des adresses est qu'il permet à un programmeur d'entrer directement dans le fonctionnement interne de l'ordinateur et accéder à la structure de stockage de ce dernier.
- Cette possibilité d'accès donne au programmeur des capacités et une puissance qui ne sont pas disponibles dans d'autres langages de haut niveau.
- Nous appelons **pointeurs** les variables stockant des adresses.

Adresses et pointeurs

- En C chaque variable a trois éléments principaux qui lui sont associés: son **type**, la **valeur stockée** dans la variable et **l'adresse** de la variable.
- Le type est déclaré à l'aide d'une instruction de déclaration.
- La valeur actuelle stockée dans une variable est formellement appelée **rvalue**.
- L'adresse mémoire de la variable est appelée **lvalue**.



- Une **lvalue** fait référence à un objet qui persiste. Une **rvalue** est une valeur temporaire qui ne persiste pas.
- En général, nous nous intéressons à la valeur attribuée à une variable (son contenu ou rvalue) et nous accordons peu d'attention à l'endroit où la valeur est stockée (son adresse, ou lvalue).

Adresses et pointeurs

- L'exemple suivant :

```c
#include <stdio.h>
main ( )
{
int N;
N = 22;
printf ("La valeur stockee dans N est %d.",N) ;
printf("\nL'ordinateur utilise %d octets pour stoker cette valeur.",sizeof(N));
//  sizeof donne la taille de l'espace mémoire occupée par la varible
}
```

Affiche :
La valeur stockee dans N est 22.
L'ordinateur utilise 4 octets pour stoker cette valeur.

- La valeurs 22 est la **rvalue** stockée dans **N** et 4 est la quantité de stockage utilisée pour ce nombre (**rvalue**).

Adresses et pointeurs

- Nous pouvons obtenir l'adresse, ou **lvalue**, correspondant à la variable **N**. L'adresse affichée correspond à l'adresse du premier octet de cette variable.
- Pour déterminer l'adresse de **N**, nous devons utiliser l'opérateur d'adresse, &, qui signifie l'adresse de, directement devant le nom de la variable (pas d'espace entre & et la variable).

```c
#include <stdio.h>
    main ( )
    {
        int N;
        N = 22;
        printf ("La valeur stockee dans N est %d.",N) ;
        printf ("\nN = %d a l'adresse %p", N, &N);
    }
```

Le programme affiche :
N = 22 a l'adresse 0061FF1C

- L'adresse affichée dépend de l'ordinateur utilisé.
- L'adresse est imprimée en utilisant la séquence de contrôle **%p**.

Utilisation des adresses

- Il est possible de stocker des adresses dans des variables correctement. Par exemple, l'instruction

```
N_adr = &N;
d = &m;
```

stocke les adresses des variables **N** et **m** dans les variables **N_adr** et **d**.

- Les vriables **N_adr** et **d** sont appelée **pointeurs**. Il sont simplement des variables utilisées pour stocker les adresses d'autres variables.
- Un pointeur est une variable qui contient un addresse. Le symbole * est utilisé pour déclarer un pointeur.
- ***N_adr** est un pointeur, ***N_adr** désigne la variable dont l'adresse est stockée dans **N_adr**.

Utilisation des adresses

- Lors de l'utilisation d'une variable pointeur, sa valeur finale est obtenue en allant sur l'adresse contenue, qui elle ensuite utilisée pour obtenir le contenu souhaité.
- Un pointeur doit être déclaré avant de pouvoir être utilisé.
- Par exemple, si l'adresse dans le pointeur N_adr est l'adresse d'un entier, la déclaration correcte pour le pointeur est :

```
int *N_adr;
```

- Cette déclaration précise deux choses:
  1. que la variable pointée par **N_adr** est un entier.
  2. que **N_adr** doit être un pointeur (car il est utilisé avec l'opérateur d'indirection *).
- Cette déclaration peut être lue la variable dont l'adresse est stockée dans **N_adr** est un entier ou la variable pointée par **N_adr** est un entier.

Exemple

```c
#include <stdio.h>
main( )
{
int *N_adr;/* declare un pointeur sur un entier */
int miles, dist;/* declare deux variables de types entiers */

dist = 158;/* stocke la valeur de 158 dans dist */
miles = 22;/* stocke la valeur de 22 dans miles */
N_adr = &miles; /* stocke l'adresse de miles dans num_addr */
printf("\nL'adresse stockee dans N_adr est %p\n",N_adr);
printf("\nLa valeur a pointee par N_adr est %d\n\n",*N_adr);
N_adr = &dist; /* Maintenant on stocke l'adresse de dist dans num_addr */
printf("\nMaintenant l'adresse stockee dans N_adr est %p\n",N_adr);
printf("\nMaintenant la valeur a pointee par  N_adr est %d\n",*N_adr);
}
```

L'adresse stockee dans N_adr est 0061FF18
La valeur a pointee par N_adr est 22
Maintenant l'adresse stockee dans N_adr est 0061FF14
Maintenant la valeur a pointee par N_adr est 158

- **NB :** seules les adresses doivent être stockées dans des pointeurs.

Table des matières

Passage des paramètres par référence

```
max = fct_max (N1, N2);
```

- Nous avons aeppelé ce mode de passage de paramètres le **passage par valeur**. Autrement dit la fonction **fct_max** reçoit des copies des valeurs contenues dans **N1** et **N2** lorsque l'appel est effectué.
- Il existe un autre type de passage de paramètre, qui est appelé le passage par référence.
- Ce mode de passage permet la fonction de se référencer ou d'accéder à la variable en utilisant l'adresse transmise.

```
tri = sort_val(&N1, &N2);
```

- Etant donné que les adresses sont stockés dans des pointeurs, cela signifie que les arguments de la fonction **sort_val()** doivent être déclarés comme des pointeurs.
- Le fonction **sort_val ()** sera définie pour comparer les valeurs contenues dans les adresses passées et permutées les valeurs, si nécessaire, de sorte que la valeur la plus petite soit stockée dans la première adresse.

Passage par référence des paramètres

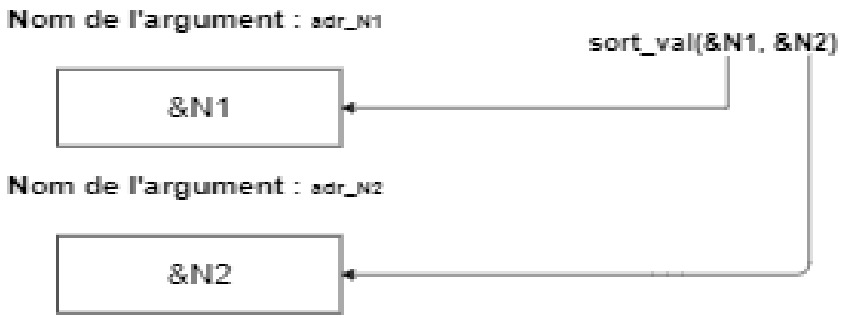
```c
#include <stdio.h>
main()
{
double N1 = 20.0, N2 = 5.0;
/* Le prototype de la fonction */
void sort_val(double *, double *);
sort_val(&N1, &N2);
}
void sort_val(double *adr_N1, double *adr_N2)
{
//  pour afficer une valeur double nous utilisons %lf
printf("l'adresse adr_N1 contient %lf",*adr_N1);
printf("\nLa l'adresse adr_N1 contient est %lf",*adr_N2);
}
```

La focntion reçoit deux pointeurs pour stocker les adresses de deux valeurs doubles.
Avant d'écrire le corps de **sort_val ()**, vérifions d'abord que les valeurs accédées à l'aide des adresses dans **adr_N1** et **adr_N2** sont corrects.

- La fonction **sort_val()** ne renvoie aucune valeur et que ses arguments sont deux pointeurs qui "pointent vers" des valeurs doubles.
- Lorsque la fonction est appelé, il faudra que deux adresses soient passées, et que chaque adresse soit l'adresse d'une valeur double précision.

Passage par référence des paramètres

- Notez que l'opérateur & est utilisé pour accéder aux valeurs stockées dans **N1** et **N2**.
- La fonction **sort_val** n'a aucune connaissance de ces noms de variables, mais elle a l'adresses de **N1** et **N2** qui sont stockées dans **adr_N1** et **adr_N2**.
- Nous avons utilisé des pointeurs pour permettre à notre fonction d'accéder aux variables de **main ()**.



- Après avoir vérifié que **sort_val()** peut accéder aux variables locales de **main ()**, nous pouvons maintenant développer cette fonction pour comparer les valeurs dans ces variables avec une instruction **if**.

Exemple final

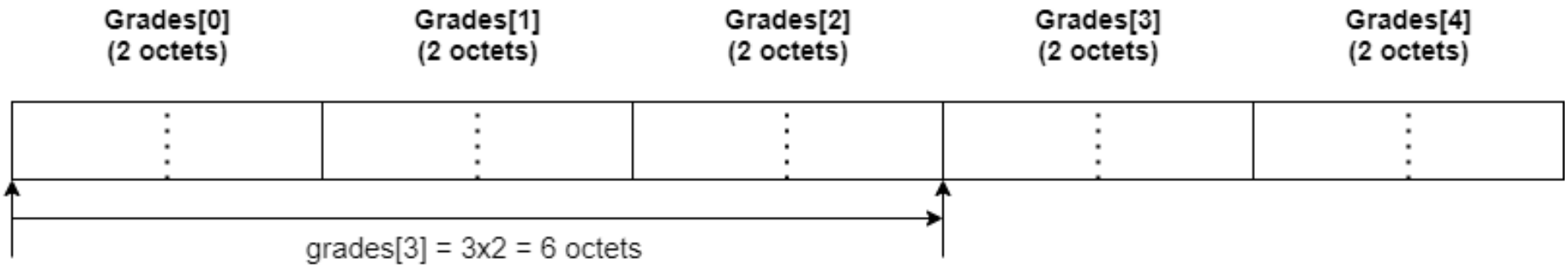
```c
#include <stdio.h>
main()
{
    double N1 , N2;
    /* Le prototype de la fonction */
    void sort_val(double *, double *);
    printf("Entrer deux nombres : ");
    scanf ("%lf %lf", &N1, &N2);
    /* Appel de sort_val( ) */
    sort_val(&N1, &N2);
    printf ("La petite valeur est %6.2lf", N1);
    printf("\nLa grande valeur est %6.2lf", N2);
}
void sort_val(double *adr_N1, double *adr_N2)
{
    double temp; // Variable temporaire
    if(*adr_N1 > *adr_N2)
    {
        temp = *adr_N1;//Sauvgarder la valeur N1
        *adr_N1 = *adr_N2;//Permuter la valeur de N2 dans N1
        *adr_N2 = temp;//Changer la valeur de N2
    }
}
```

- Notez bien que la raison pour laquelle les pointeurs sont utilisés; c'est qu'ils nous permettent d'échanger la valeur de **N1** avec **N2** si elles ne sont pas dans le bon ordre.

Table des matières

Noms des tableaux et pointeurs

- Il existe une relation directe entre les noms des tableaux et les pointeurs.
- Le tableau **grades** contient cinq entiers. Supposons que chaque entier nécessite deux octets de stockage.



- L'ordinateur utilise immédiatement l'indice pour calculer l'adresse de l'élément souhaité en fonction à la fois du début l'adresse du tableau et la quantité de stockage utilisée par chaque élément.
- Appelant le quatrième élément grades [3] force l'ordinateur, en interne, de faire le calcul suivant pour obtenir l'adresse :

```
&grades [3] = &grades [0] + (3 * 2)
```

Noms des tableaux et pointeurs

- L'opérateur d'adresse, &, signifie que l'adresse de **grades [3]** est égale à l'adresse de **grades [0]** plus 6.
- L'exemple suivant affiche les valeurs de du tableau **grades** :

```
#include <stdio.h>
main( )
{
int i, grades [ ] = {98, 87, 92, 79, 85};
for (i = 0; i <= 4; ++i)
printf("\nElement d'indice %d est %d",i,grades[i]);
}
```

- Nous allons maintenant créer un pointeur pour stocker l'adresse du premier élément dans le tableau **grades** pour pouvoir accéder aux éléments du tableau.
- Stocker l'adresse de l'élément **0** du tableau dans un pointeur, *, permet d'accéder indirectement à chaque élément du tableau.

Noms des tableaux et pointeurs

- L'exemple précédent devient :

```c
#include <stdio.h>
main()
{
int *g_ptr;/* déclare un  pointeur vers un int */
int i, grades[] = {98, 87, 92, 79, 85};
g_ptr = &grades[0];/* stocke l'adressse de l'élément 0 du tableau */
for (i = 0; i <= 4; ++i)
printf("\nElement d'indice %d est %d", i, *(g_ptr + i) );
}
```

- L'utilisation des pointeurs simule la façon dont l'ordinateur référencie en interne tous les éléments de tableau. Tout indice est automatiquement converti en un pointeur.
- Les parenthèses dans l'expression * **(g_ptr + 3)** sont nécessaires. L'omission des parenthèses entraîne l'ajout de 3 à la variable pointée par **g_ptr**.
- Notez également que l'expression * **(g_ptr + 3)** ne change pas l'adresse stockée dans **g_ptr**.

Noms des tableaux et pointeurs

- En C toute référence à **grades** utilisant un indice peut être remplacé par une référence équivalente utilisant **grades**.
- Ainsi, partout où l'expression **grades [i]** est utilisée, l'expression ***(grades + i)** peut également être utilisée.

```c
#include <stdio.h>
main()
{
int i, grades[] = {98, 87, 92, 79, 85};
for (i = 0; i <= 4; ++i)
printf("\nElement d'indice %d est %d", i, *(grades + i) );
}
```

- Le nom d'un tableau est en même temps un pointeur que l'on peut utiliser de manière interchangeable.
- Un vrai pointeur est une variable et l'adresse qui y est stockée peut être modifiée.
- Par contre un nom de tableau est une constante de pointeur et l'adresse stockée dans ce pointeur ne peut pas être modifiée.

Table des matières

Pointeur Arithmétique

- L'adresse stockée dans un pointeur que nous pouvant la manipuler comme toute vraible.
- Ainsi, en ajoutant et en soustrayant des nombres aux pointeurs nous pouvons obtenir différentes adresses.
- En exécutant des opérations arithmétique sur les pointeurs, nous devons faire attention à produire des adresses qui indiquent quelque chose de significatif.
- Les adresses peuvent également être incrémentées ou décrémentées.
- L'ajout d'un à un pointeur provoque force le pointeur à pointer vers l'élément suivant du type sur lequel il pointe.
- Par exemple, si le la variable de pointeur **p** est un pointeur sur un entier, l'expression **p++** provoque l'incrémentation de l'adresse pour pointer vers l'entier suivant.

Pointeur Arithmétique

- Toutes les combinaisons suivantes utilisant des pointeurs sont valide:

```
*pt_num++;/* utilise le pointeur et l'incrémente */
*++pt_num;/* incrémente le pointeur avant de l'utiliser */
*pt_num--;/* utilise le pointeur et le décrémente */
*--pt_num;/* décrément le pointeur avant de l'utiliser */
```

- Parmi les quatre formes possibles, la plus couramment utilisée est la forme ***pt_num ++**.

- Pour L'exemple suivant montre l'utilisation de l'opérateur d'incrémentation.

Pointeur Arithmétique

- Dans ce programme, chaque élément du tableau **nums** est récupéré en incrémentant successivement l'adresse en **n_pt**.

```c
#include <stdio.h>
main( )
{
int nums[5] = {16, 54,7,43, -5};
int i, total = 0, *n_pt;
n_pt = nums; /* stocke l'adresse de nums[0] dans n_pt */
for (i = 0; i <= 4; ++i)
{
    total = total + *n_pt++;
}
printf("Le total des elements du tableau est %d", total);
}
```

- L'instruction ***n_pt++** récupére d'abord l'entier pointé à l'aide de *n_pt.
- Alors que l'incrément ++ ajoute un à la adresse dans *n_pt.

Comparaison de pointeurs

- Des pointeurs peuvent également être comparés. Par exemple, au lieu d'utiliser un compteur dans une boucle for pour accéder correctement à chaque élément d'un tableau, l'adresse dans un pointeur peut être comparé à l'adresse de début et de fin du tableau lui-même.

```c
#include <stdio.h>
main( )
{
int nums[5] = {16, 54,7,43, -5};
int i, total = 0, *n_pt;
n_pt = nums; /* stocke l'adresse de nums[0] dans n_pt */
while(n_pt <= nums+ 4)
{
    total = total + *n_pt++;
}
printf("Le total des elements du tableau est %d", total);
}
```

- Puisque nums est une constante de pointeur qui contient l'adresse de **nums [0]**, le terme **& nums [4]** peut être remplacé par le terme équivalent **nums +4**.

Table des matières

Exemple 1

- Le programme suivant lit deux variables entières **a** et **b** avec la fonction **scanf**, en passant par deux pointeurs **ptr_a** et **ptr_b** contenant leurs adresses.

- La **somme** et le **produit** de ces deux enitiers sont calculés en utilisant des pointeurs pointant sur leurs adresses respective.

Exemple 1

```c
#include <stdio.h>
main()
{
int a,b,somme, produit;
int *ptr_a,*ptr_b,*ptr_somme,*ptr_produit;
ptr_a = &a;
ptr_b = &b;
ptr_somme = &somme;
ptr_produit = &produit;
printf("donnez deux entiers:");
printf("\nEntier 1 = ");
scanf("%d",ptr_a);
printf("\nEntier 2 = ");
scanf("%d",ptr_b);
*ptr_somme = *ptr_a + *ptr_b;
*ptr_produit = (*ptr_a)*(*ptr_b);
printf("Leur somme est  %d et leur produit est %d.\n",somme,produit);
}
```

Table des matières

Exemple 2

```c
#include<stdio.h>
#include<stdlib.h>
main()
{
int tab[50];
int i,n, somme=0;
printf("Calcul de la somme des elements les pointeurs\n");
printf("en utilisant un tableau\n");
printf("1- Donnez le nombre d'elements du tableau: ");
scanf("%d",&n);
for(i=0; i<n; i++)
{
    printf("Entrez l'element %d =",i);
    scanf("%d", &tab[i]);
    printf("\n");
}
for(i=0; i<n; i++)
{
     somme = somme + *(tab+i);
}
printf("La somme = %d\n",somme);
}
```

Table des matières

Exemple 3

```c
#include <stdio.h>
main()
{
 int A[100], B[50]; // tableaux //
 int N, M;  // dimensions des tableaux //
 int i;     // indice courant //
 // --Saisie des données --//
 printf("Dimension du tableau A (max.50) : ");
 scanf("%d", &N );
 for (i=0; i<N; i++)
     {
      printf("Element %d : ", i);
      scanf("%d", A+i);
     }
 printf("Dimension du tableau B (max.50) : ");
 scanf("%d", &M );
 for (i=0; i<M; i++)
     {
      printf("Element %d : ", i);
      scanf("%d", B+i);
     }
```

Exemple 3

```
//-- Affichage des tableaux --//
     printf("Tableau  A :\n");
     for (i=0; i<N; i++)
     {
         printf("%d ", *(A+i));
     }
     printf("\n");
     printf("Tableau B :\n");
     for (i=0; i<M; i++)
     {
         printf("%d ", *(B+i));
     }
     printf("\n");
     //-- Copie de B à la fin de A */
     for (i=0; i<M; i++)
     {
         *(A+N+i) = *(B+i);
     }
    //--- Nouvelle dimension de A ---//
     N += M;
    //--- Affichage du tableau r&sultants ---//
     printf("Tableau resultat A :\n");
     for (i=0; i<N; i++)
     {
         printf("%d ", *(A+i));
    }
    printf("\n");
    }
```

Table des matières

Passer et utiliser des adresses de tableau

- Lorsqu'un tableau est passé à une fonction, son adresse est le seul élément réellement passé (c.à.d. l'adresse du premier emplacement utilisé pour stocker le tableau).
- Considérons le programme suivant :

```c
#include <stdio.h>
main( )
{

    int nums[5] = {2, 18, 1, 27, 16};
    int find_max(int [], int); // prototype de la fonction //
    printf("La valeur max est %d", find_max(nums,5));
}
int find_max(int vals[], int num_els)
{
    int i, max = vals[0];
    for (i = 1; i < num_els; ++i)
    if (max < vals[i])
    {
        max = vals[i];
    }
    return (max) ;
}
```

Passer et utiliser des adresses de tableau

- L'argument nommé **vals** dans la déclaration de **find_max ()** reçoit l'adresse du tableau **nums**. Comme nous l'avons dit précédemment, **vals** est un pointeur puisqu'il stocke des adresses.

- L'adresse passée dans **find_max ()** est l'adresse d'un entier, on peut utiliser une autre déclaration pour cette fonction :

```
find_max(int *vals, int num_els)
```

- Ici **vals** est déclaré comme un pointeur vers un entier.

Passer et utiliser des adresses de tableau

```c
#include <stdio.h>
main( )
{
    int nums[5] = {2, 18, 1, 27, 16};
    int find_max(int *, int); // prototype de la fonction //
    printf("La valeur max est %d", find_max(nums,5));
}
int find_max(int *vals, int num_els)
{
    int i, max = *vals;
    /* Trouve l valeur max */
    for (int i=0; < num_els; ++i)
    {
        if (max < *(vals + i) )
        {
            max = *(vals + i);
        }
    }
    return (max);
}
```

Références