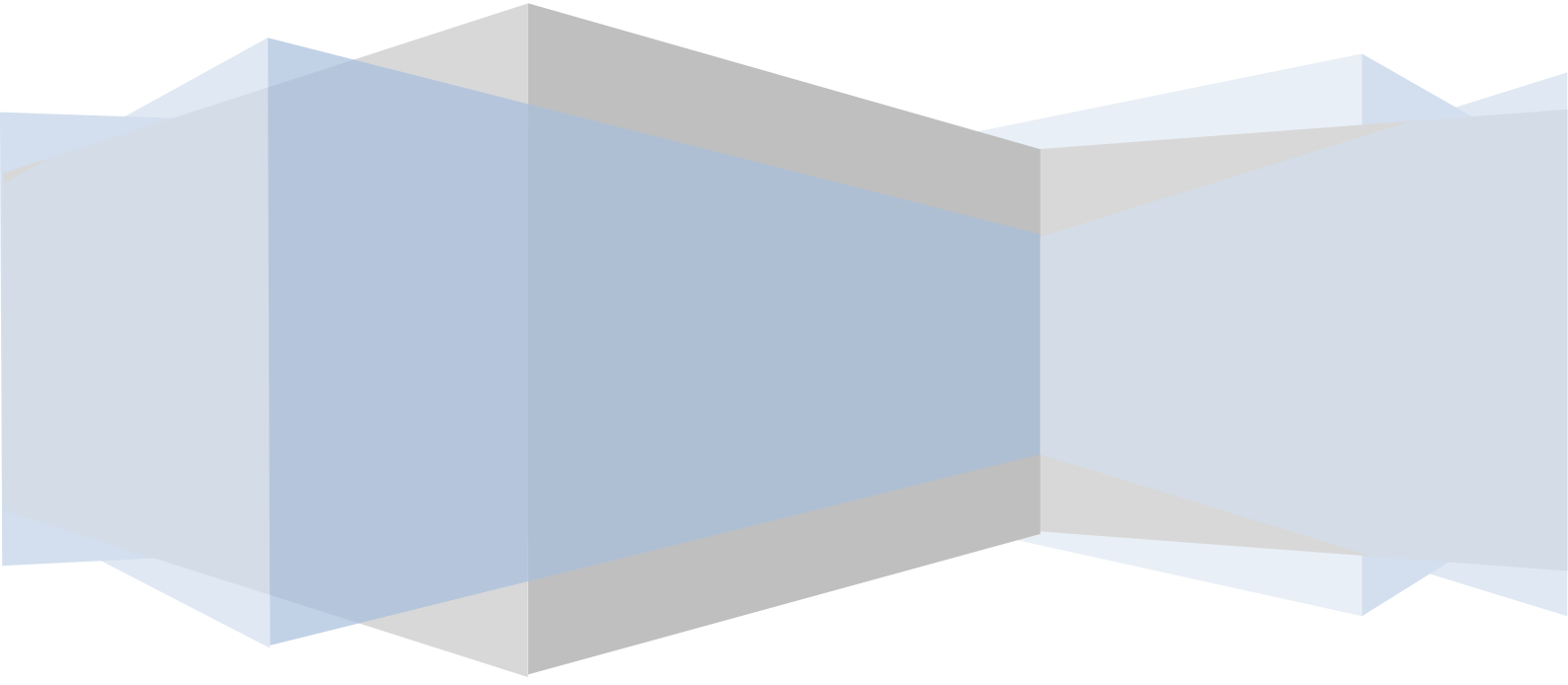
BTS SN-IR 1 Année

Lycée Diderot, Paris

TP Pointeurs

Programmation C++ sous Linux

Gilles Dalles



1. Introduction

Nous avons vu en cours ce que sont les pointeurs en théorie. Ici, il s'agit de passer à la pratique par une série d'exercices qui vont nous permettre d'appréhender les pointeurs et d'utiliser leurs propriétés.

1.1. Le type void

Le type **void**, que nous avons pour l'instant utilisé dans le cas des procédures est un type à part; il signifie en quelque sorte « rien » .

Du coup, ce type est peu particulier: il ne possède pas de taille, 'rien' n'occupe pas d'octet dans la mémoire, donc on ne peut pas déclarer de variable. Par contre, on peut l'utiliser pour créer des pointeurs sur **void**, c'est à dire 'sur rien'. Ceci dit, un pointeur, qu'il soit de **int**, de **char** ou de **float**, est une variable permettant de stocker des adresses mémoire. De fait, un pointeur sur **void** permet également de stocker une adresse. Ce pointeur est un pointeur générique, il permet de pointer sur n'importe quel type de variable.

Par exemple, dans le code ci-dessous:

```
int main()
{
    int i = 0;
    void *p = &i;
    *p = 35;
    cout<<i;    //affiche 35
    return 0;
}
```

Le pointeur **p** se retrouve à pointer la variable **i** de type **int**. Créer des pointeurs sur void devient alors utile si on ne sait pas à l'avance sur quel type de variable on doit pointer.

Ce code est cependant obsolète: la nouvelle version de votre compilateur génère une erreur lors de l'affection de la valeur 35 à *p ; "void*" is not a pointer-to-object type . Pas de bol! Ceci étant dit, void * reste utile dans certains cas, notamment lors de l'utilisation de fonctions standards de C.

1.2. Transtypage

Le **transtypage** ou **cast** en anglais, est une opération permettant faire passer un type pour un autre. Par exemple: il se peut que vous disposiez d'une fonction demandant en paramètre un **int**. Vous voulez de votre côté, utiliser cette fonction avec la valeur d'un **char** (pour rappel, dans l'absolu, un char est un entier). Le compilateur peut vous créer une erreur. Pour éviter cela, vous devez transtyper votre **char** en indiquant simplement le type souhaité devant la variable.

Dans l'exemple ci-dessous, on souhaite afficher la valeur entière d'un char (son code ASCII):

```
int main()
{
    char caract = 'a';
    cout<< "le caractère a a une valeur ASCII de: " <<(int)caract;
    return 0;
}
```

Le fait d'écrire **(int)** devant **caract** permet de demander à cout d'afficher la valeur entière de la variable et non pas le caractère qu'elle contient. Le résultat de l'instruction est donc: `"le caractère a a une valeur ASCII de : 97".`

Si vous voulez la valeur hexadécimale, il suffit de remplacer **(int)** par **(int *)**. Le résultat de l'instruction devient alors: `"le caractère a a une valeur ASCII de : 0x61".`

Attention, le transtypage n'est en aucun cas un moyen de conversion d'un type vers un autre.

2. Travail a realiser

2.1. Identification des zones memoire

Dans un premier temps, nous allons créer des variables de natures différentes afin d'identifier dans quelle zone mémoire elles sont stockées.

Pour cela, vous allez tester un programme permettant l'affichage des adresses des variables. Il vous faut 3 variables:

- Une globale
- Une statique
- Une locale

Affichez et notez les adresses de vos variables et identifiez les zones mémoire auxquelles elles appartiennent. C'est important pour la suite.

2.2. Passage de parametres

Comme on l'a vu en cours, il existe 3 passages de paramètres: par valeur, par adresse et par référence. J'espère que vous me voyez venir, je vais vous demander d'effectuer les trois.

2.2.1. Passage par valeur

Vous allez créer une fonction **procPassVal** sans valeur de retour. Cette fonction prendra un seul paramètre (un entier). Le but est donc d'effectuer un passage par valeur. La fonction doit changer la valeur du paramètre. Une fois n'est pas coutume,

pour les besoins du TP, vous afficherez dans cette fonction la valeur et l'adresse du paramètre.

Dans le même temps, dans votre fonction main, vous appellerez la fonction **procPassVal**. Vous lui passerez en paramètre une variable locale au main, initialisée et là aussi, vous afficherez sa valeur et son adresse.

2.2.2. PASSAGE PAR ADRESSE

Modifiez le programme établi au paragraphe **2.2.1** afin de passer cette fois une adresse en paramètre de la procédure **procPassVal**.

2.2.3. PASSAGE PAR REFERENCE

Modifiez votre programme afin de passer une référence en paramètre de **procPassVal**.

2.3. ETUDE D'UN PROBLEME

Dans le source qui suit, nous n'avons pas un comportement normal. Le but du jeu est de déterminer quelle(s) raison(s) pousse(nt) le programme a se comporter ainsi.

Pour cela, l'affichage des adresses des variables peut être instructif.

```
void remplir(int *tab)
{
    srand(time(NULL));
    int i=8;
    do{
        i--;
        tab[i]=rand()%50;
    }while(i>=0);
}

int main()
{
    int tableau[8];
    int borne=8;
    int i=0;
    remplir(tableau);
    for(i=0; i<borne; i++)
    {
        cout<< tableau[i] << " | ";
    }
    return 0;
}
```

Quelle (s) solution (s) proposeriez-vous pour remédier à ce problème? Que se passe-t-il réellement?

2.4. ALLOCATION DYNAMIQUE DE MEMOIRE

2.4.1. PROGRAMME

Il s'agit ici de créer un programme permettant de remplir et d'afficher un tableau 1 dimension de taille **n** avec des valeurs aléatoires comprises entre 1 et **n**.

n sera demandé à l'utilisateur lors de l'exécution du programme. Comme indiqué dans le titre, il vous faut passer par une allocation dynamique pour répondre au cahier des charges.

2.4.2. IDENTIFICATION DE ZONE MEMOIRE

Une fois votre programme réalisé, identifiez la zone mémoire dans laquelle sont stockées les valeurs du tableau. A partir d'ici, vous devez pouvoir reconnaitre les 3 zones mémoires allouées aux données de votre ordinateur.

2.4.3. CHAINES DE CARACTERES

Les chaines de caractères peuvent prendre plusieurs formes en C++:

- Tout d'abord, les **string** telles que nous les utilisons depuis le début
- Les tableaux de **caractères**
- Les pointeurs sur **char**

Le programme que vous devez écrire doit afficher, pour chacune des formes de la chaine de caractères, l'adresse mémoire de la variable ainsi que l'adresse de chaque caractère. Vous devez être en mesure d'identifier dans quelles zones mémoire tout ce beau monde est stocké.

Pour faire simple, vos chaines seront initialisées par la phrase :"Les optimistes pensent que nous vivons dans le meilleur des mondes possibles, les pessimistes en sont intimement persuadés".