

Mise à niveau en C

Pointeurs et allocation dynamique

Enseignant: P. Bertin-Johannet

Mémoire et adressage

- Dans la mémoire, les octets sont tous identifiés par une adresse
- Par exemple, le code ci-dessous pourrait être représenté ainsi :

```
int i = 5;  
char a = 'a';  
int j = 24;
```

Valeurs Adresses

5	0xff17065
	0xff17066
97	0xff17067
24	0xff17068
??	0xff17069

Pointeurs

- On peut créer des variables contenant une adresse en mémoire, on appelle ça un pointeur
- Pour déclarer une variable de type de pointeur on ajoute une étoile (*) après le type :

```
int i ; // i est un int
int* iptr ; // iptr est un pointeur vers un int
char* cptr ; // cptr est un pointeur vers un char
```

Pour récupérer l'adresse d'une variable on utilise l'opérateur &

```
int i ; // i est un int
int* iptr = &i; // iptr contient l'adresse de i
```

Exemple

```
int i = 5;  
char a = 'a';  
int* iptr = &i;  
char* aptr = &a;
```

Mémoire	Adresse
5	0xff150
	0xff151
97	0xff152
0xff150	0xff153
	0xff154
	0xff155
	0xff156
0xff152	0xff157
	0xff156
	0xff156
	0xff156

Utilisation des pointeurs

- Quand on a un pointeur, on peut accéder à la valeur vers laquelle il pointe avec l'opérateur *

```
int* iptr = &i; // iptr pointe vers un entier
int j = *iptr;  // j est la valeur vers laquelle pointe iptr
```

- On peut afficher un pointeur avec printf en utilisant le format "%p" :

```
int i ; // i est un int
int* iptr = &i; // iptr contient l'adresse de i
printf("l'adresse de i est %p et sa valeur est%d", iptr,
*iptr) ;
```

Pourquoi utiliser un pointeur

Un pointeur peut permettre de:

- Créer une fonction qui modifie une variable extérieure à la fonction
- Créer une fonction qui renvoie plusieurs valeurs
- Accéder à de la mémoire hors de la pile de fonctions
- Utiliser des tableaux de taille dynamique
- Maîtriser précisément l'usage de la mémoire

Exemples d'usage

- Une fonction qui modifie les arguments passés en paramètres

```
void set_to_n(int n, int* a, int* b, int* c){  
    *a = n; // on écrit la valeur de n à l'adresse vers laquelle pointe a  
    *b = n; // idem à l'adresse vers laquelle pointe b  
    *c = n; // idem pour c  
}  
  
int main(){  
    int a, b, c;  
    set_to_n(5, &a, &b, &c); // on passe les addresses de a, b et c  
    printf("values of a, b, c : %d, %d, %d\n", a, b, c); // affiche 5, 5, 5  
}
```

Exemples d'usage

- Une fonction qui modifie les arguments passés en paramètres afin de renvoyer plusieurs valeurs

```
void division_avec_reste(int a, int b, int* resultat, int* reste){  
    *resultat = a / b;  
    *reste = a % b;  
}
```

```
int main(){  
    int resultat, reste;  
    division_avec_reste(35, 8, &resultat, &reste);  
    printf("35/8 = %d reste %d\n", resultat, reste);  
    // affiche "35/8 = 4 reste 3"  
}
```


Arithmétique de pointeurs

- Additionner un nombre à un pointeur revient à faire avancer le pointeur dans la mémoire.
- Le pointeur avance du nombre d'octet occupé par le type vers lequel il pointe
- Si un int prend 4 octets, ajouter 1 à un pointeur vers un int le déplacera de quatres cases mémoire.

Exemple

```
int i = 5;  
int j;  
int* iptr = &i;  
int* jptr = iptr + 1;  
*jptr = 4;
```

Mémoire	Adresse
5	0xff150
	0xff151
4	0xff152
	0xff151
0xff150	0xff153
	0xff154
	0xff155
	0xff156
0xff152	0xff157
	0xff156
	0xff156
	0xff156

Tableaux et pointeurs

- Un tableau fonctionne comme un pointeur vers son premier élément.
- accéder à l'élément n d'un tableau revient à y additionner n puis récupérer la valeur vers laquelle elle pointe

```
int tab[4];  
tab[2] = 0; // cette ligne  
*(tab + 2) = 0; // est transformée en cette ligne par le  
compilateur
```

Allocation statique de la mémoire

- On parle d'allocation quand on reserve une partie de la mémoire
- Jusqu'à présent nous avons pu réserver des octets dans la mémoire en déclarant des variables

```
int i ; // des octets sont réservés pour i  
char c ; // un octet est réservé pour c  
int* ptr; // des octet sont réservé pour ptr
```

- Chaque fonction, lorsqu'elle est appelée, réserve juste assez de mémoire pour contenir toutes ses variables ainsi que les arguments.
- Le calcul de la mémoire utilisée par une fonction est habituellement fait **au moment de la compilation**
- Il est donc difficile de changer dynamiquement la mémoire occupée par les variables d'une fonctions

La pile d'appels

- Chaque appel de fonction alloue de la mémoire pour ses variables à la suite de la mémoire utilisée par la fonction précédente
- Elle alloue aussi de la mémoire pour stocker des informations permettant de reprendre l'exécution de la fonction précédente (adresse de la prochaine instruction, adresse de début de la fonction, etc...)
- La partie de la mémoire ainsi utilisée est appelée la pile (ou stack en anglais)

```
void f(){printf();}  
void h(){f();}  
int main(){h();}
```

Mémoire

putc
putchar
printf
f
h
main

Allocation dynamique sur la pile

L'allocation dynamique de mémoire dans la pile est possible en utilisant principalement deux méthodes.

- Les tableaux de tailles variables (Variable Length Array ou VLA en anglais). La mémoire réservée l'est uniquement tant que la variable existe.
- La fonction `alloca()`. La mémoire réservée l'est jusqu'à la fin de l'exécution de la fonction.

```
int n = rand();  
if (n > 0){  
    char tableau[n]; // ici la mémoire réservée par tableau est calculée pendant  
l'exécution  
    char* tableau2 = alloca(n); // idem  
} // la mémoire réservée par tableau n'est plus valide à partir d'ici  
// la mémoire réservée par tableau2 sera valide jusqu'à la fin de la fonction
```

Désavantages de l'allocation dynamique sur la pile

- Le nombre d'instructions utilisées augmente
- La pile est limitée en mémoire (consultable avec : `ulimit -a`) et la nature imprévisible de l'allocation dynamique entraine un fort risque d'erreur stack overflow.
- La mémoire allouée n'est réservée qu'un temps et le risque de l'utiliser après sa validité est accru.
- Pour ces raisons, l'utilisation de tableaux de tailles variables est interdite ou fortement déconseillée dans de nombreux projets (par exemple le noyau linux)

Allocation dynamique sur le tas

- Il est possible de réserver de la mémoire dans une autre zone du programme appelée le **tas** (en anglais **heap**)
- La fonction `malloc()` prend accepte en argument le nombre d'octets à réserver
- Elle renvoie un pointeur vers le début de la mémoire réservée
- La mémoire réservée est valide **jusqu'à la fin de l'exécution du programme**

```
int* a = malloc(8); // on reserve 8 octets pour un tableau d'entiers
a[0] = 5;
a[1] = 4;
printf("%d\n", *(a+1));
// affiche 4
```


L'opérateur sizeof

- Pour calculer la taille mémoire occupée par un type, on peut utiliser `sizeof`
- `sizeof` prend en argument un **type** et renvoie le nombre d'octets qu'il occupe.
- Cela permet de facilement allouer un tableau sur le tas.

```
float* tab = malloc(9 * sizeof(float));  
// ici on reserve assez de place pour 9 floats quelque soit leur  
taille mémoire.
```

Libération de la mémoire

- Si l'on ne fait rien, la mémoire réservé par malloc l'est jusqu'à la fin de l'exécution
- Lorsqu'on ne s'en sert plus, on doit manuellement la “libérer” en utilisant `free()`
- Tant qu'on ne la libère pas, la mémoire réservée ne sera pas utilisable ailleurs

```
float* tab = malloc(9 * sizeof(float)); // on reserve  
...  
free(tab); // on libère
```

Fuite mémoire

- Il est possible de ne plus avoir accès au pointeur renvoyé par malloc
- La mémoire reste réservée mais on ne peut plus la libérer
- On appelle ça une fuite mémoire

```
if (1){  
    char* tab = malloc(9); // on reserve 9 chars  
    ...  
}  
free(??); // on ne peut plus libérer la mémoire
```

Mise en pratique

```
printf(")
```

```
< TP 4 >
```

```
-----
```

```
 \      ^  ^  
  \      ^  ^  
   \      ^  ^  
    \      ^  ^  
     \      ^  ^  
      \      ^  ^  
       \      ^  ^  
        \      ^  ^  
         \      ^  ^  
          \      ^  ^  
           \      ^  ^  
            \      ^  ^  
             \      ^  ^  
              \      ^  ^  
               \      ^  ^  
                \      ^  ^  
                 \      ^  ^  
                  \      ^  ^  
                   \      ^  ^  
                    \      ^  ^  
                     \      ^  ^  
                      \      ^  ^  
                       \      ^  ^  
                        \      ^  ^  
                         \      ^  ^  
                          \      ^  ^  
                           \      ^  ^  
                            \      ^  ^  
                             \      ^  ^  
                              \      ^  ^  
                               \      ^  ^  
                                \      ^  ^  
                                 \      ^  ^  
                                  \      ^  ^  
                                   \      ^  ^  
                                    \      ^  ^  
                                     \      ^  ^  
                                      \      ^  ^  
                                       \      ^  ^  
                                        \      ^  ^  
                                         \      ^  ^  
                                          \      ^  ^  
                                           \      ^  ^  
                                            \      ^  ^  
                                             \      ^  ^  
                                              \      ^  ^  
                                               \      ^  ^  
                                                \      ^  ^  
                                                 \      ^  ^  
                                                  \      ^  ^  
                                                   \      ^  ^  
                                                    \      ^  ^  
                                                     \      ^  ^  
                                                      \      ^  ^  
                                                       \      ^  ^  
                                                        \      ^  ^  
                                                         \      ^  ^  
                                                          \      ^  ^  
                                                           \      ^  ^  
                                                            \      ^  ^  
                                                             \      ^  ^  
                                                              \      ^  ^  
                                                               \      ^  ^  
                                                                \      ^  ^  
                                                                 \      ^  ^  
                                                                  \      ^  ^  
                                                                   \      ^  ^  
                                                                    \      ^  ^  
                                                                     \      ^  ^  
                                                                      \      ^  ^  
                                                                       \      ^  ^  
                                                                        \      ^  ^  
                                                                         \      ^  ^  
                                                                          \      ^  ^  
                                                                           \      ^  ^  
                                                                            \      ^  ^  
                                                                             \      ^  ^  
                                                                              \      ^  ^  
                                                                               \      ^  ^  
                                                                                \      ^  ^  
                                                                                 \      ^  ^  
                                                                                  \      ^  ^  
                                                                                   \      ^  ^  
                                                                                    \      ^  ^  
                                                                                     \      ^  ^  
                                                                                      \      ^  ^  
                                                                                       \      ^  ^  
                                                                                        \      ^  ^  
                                                                                         \      ^  ^  
                                                                                          \      ^  ^  
                                                                                           \      ^  ^  
                                                                                            \      ^  ^  
                                                                                             \      ^  ^  
                                                                                              \      ^  ^  
                                                                                               \      ^  ^  
                                                                                                \      ^  ^  
                                                                                                 \      ^  ^  
                                                                                                  \      ^  ^  
                                                                                                   \      ^  ^  
                                                                                                    \      ^  ^  
                                                                                                     \      ^  ^  
                                                                                                      \      ^  ^  
                                                                                                       \      ^  ^  
                                                                                                        \      ^  ^  
                                                                                                         \      ^  ^  
                                                                                                          \      ^  ^  
                                                                                                           \      ^  ^  
                                                                                                            \      ^  ^  
                                                                                                             \      ^  ^  
                                                                                                              \      ^  ^  
                                                                                                               \      ^  ^  
                                                                                                                \      ^  ^  
                                                                                                                 \      ^  ^  
                                                                                                                  \      ^  ^  
                                                                                                                   \      ^  ^  
                                                                                                                    \      ^  ^  
                                                                                                                     \      ^  ^  
                                                                                                                      \      ^  ^  
                                                                                                                       \      ^  ^  
                                                                                                                        \      ^  ^  
                                                                                                                         \      ^  ^  
                                                                                                                          \      ^  ^  
                                                                                                                           \      ^  ^  
                                                                                                                            \      ^  ^  
                                                                                                                             \      ^  ^  
                                                                                                                              \      ^  ^  
                                                                                                                               \      ^  ^  
                                                                                                                                \      ^  ^  
                                                                                                             || - - - - w ||  
                                                                                                             ||  
                                                                                                             ||
```

```
)
```