Rappel sur les pointeurs

Programmation séquentielle en C, 2022-2023

Orestis Malaspinas (A401) et un tout petit peu Michaël El Kharroubi

2023-02-21

Informatique et Systèmes de Communication, HEPIA

Rappel sur les pointeurs (1/2)

Pour reprendre dans la joie après les ~~vacances~~ **semaines sans cours**.

Rappel sur les pointeurs (1/2)

Pour reprendre dans la joie après les ~~vacances~~ **semaines sans cours**.

Qu'est-ce qu'un pointeur?

Rappel sur les pointeurs (1/2)

Pour reprendre dans la joie après les ~~vacances~~ **semaines sans cours**.

Qu'est-ce qu'un pointeur?

- Un objet qui stocke une adresse mémoire et le type des données pointées.

Rappel sur les pointeurs (1/2)

Pour reprendre dans la joie après les ~~vacances~~ **semaines sans cours**.

Qu'est-ce qu'un pointeur?

- Un objet qui stocke une adresse mémoire et le type des données pointées.

Comment déclare-t-on les pointeurs en C?

Rappel sur les pointeurs (1/2)

Pour reprendre dans la joie après les ~~vacances~~ **semaines sans cours**.

Qu'est-ce qu'un pointeur?

- Un objet qui stocke une adresse mémoire et le type des données pointées.

Comment déclare-t-on les pointeurs en C?

```c
// type * variable;
double * d;
struct element * e;
```

- `double *` est le **type** de `d` et `struct element *` est le **type** de `e`.

Comment accède-t-on à la valeur se trouvant à une adresse mémoire?

Rappel sur les pointeurs (1/2)

Pour reprendre dans la joie après les ~~vacances~~ **semaines sans cours**.

Qu'est-ce qu'un pointeur?

- Un objet qui stocke une adresse mémoire et le type des données pointées.

Comment déclare-t-on les pointeurs en C?

```c
// type * variable;
double * d;
struct element * e;
```

- `double *` est le **type** de `d` et `struct element *` est le **type** de `e`.

Comment accède-t-on à la valeur se trouvant à une adresse mémoire?

```c
int c = 2;
// On assigne l'adresse de c au pointeur p_c
int * p_c = &c;
// On déréférence un pointeur
*p_c = 4;
```

Rappel sur les pointeurs (2/2)

Quelle est la seule adresse “interdite”?

Rappel sur les pointeurs (2/2)

Quelle est la seule adresse “interdite”?

```
// l'adresse 0 ou NULL
double * e = NULL;
*e = 10; // Runtime error...
double * e;
*e = 10; // Maybe error (e has a random value)...
```

Qu’est-ce qu’un pointeur n’est pas?

- Un pointeur **n’est pas** entier (mais bien plus)...
- même si en C on peut convertir un pointeur en entier.
- D’ailleurs, on ne **peut pas** déréférencer un entier

```
uint64_t c = 2;
*c = 3; // ERREUR!
```

Comment utilise-t-on les pointeurs? (1/2)

Dans les arguments des fonctions

```
void modif_argument(int * val) {
    *val = 5;
}
int main() {
    int var = 4;
    modif_argument(&var); // on passe l'adresse de var
}
```

Comment utilise-t-on les pointeurs? (1/2)

Dans les arguments des fonctions

```
void modif_argument(int * val) {
    *val = 5;
}
int main() {
    int var = 4;
    modif_argument(&var); // on passe l'adresse de var
}
```

Pour allouer de la mémoire sur le tas

```
int main() {
    int * var = malloc(sizeof(*var));
    struct element * revar = malloc(sizeof(struct element));
    double * tab = malloc(10 * sizeof(*tab));
    free(var);
    free(revar);
    free(tab); // oui il faut pas oublier de désallouer
}
```

Comment utilise-t-on les pointeurs? (2/2)

Allouer de la mémoire sur le tas dans fonction

```
void modif_argument(int ** val) {
    *val = malloc(sizeof(int));
}
int main() {
    int * var = NULL;
    modif_argument(&var); // on passe l'adresse de var et donc on alloue
}
```

Comment utilise-t-on les pointeurs? (2/2)

Allouer de la mémoire sur le tas dans fonction

```
void modif_argument(int ** val) {
    *val = malloc(sizeof(int));
}
int main() {
    int * var = NULL;
    modif_argument(&var); // on passe l'adresse de var et donc on alloue
}
```

Que se passerait-il si....

```
void modif_argument(int * val) {
    val = malloc(sizeof(int));
}
int main() {
    int * var = NULL;
    modif_argument(var);
}
```

Comment utilise-t-on les pointeurs? (2/2)

Allouer de la mémoire sur le tas dans fonction

```
void modif_argument(int ** val) {
    *val = malloc(sizeof(int));
}
int main() {
    int * var = NULL;
    modif_argument(&var); // on passe l'adresse de var et donc on alloue
}
```

Que se passerait-il si....

```
void modif_argument(int * val) {
    val = malloc(sizeof(int));
}
int main() {
    int * var = NULL;
    modif_argument(var);
}
```

- Un code buggé (99.9% du temps) **et** une fuite mémoire...
- Mais avec un peu de chance le code va marcher (comportement indéfini).

Last but not least

Les pointeurs et les tableaux

- On peut allouer un tableau et le manipuler avec les pointeurs:

```
float * tab = malloc(12 * sizeof(*tab));
*tab = 1.2; // première case de tab = 1.2
tab[0] = 2.3; // première case de tab = 2.3
tab[2] = 3.4; // 3e case de tab = 3.4
*(tab + 4) = 4.5; // 4e case de tab = 4.5
// ceci était de l'arithmétique de pointeur
// on déréférence l'adresse (tab + 4)
// en unités de float
```