Section 3

Les pointeurs

Adresses

Les pointeurs

Mémoire d'un ordinateur $\approx$ ensemble de **cases-mémoire**.
Chaque case est repérée par une **adresse** (entier long).

	Mémoire	Adresse	
		0	
		1	
		2	
		⋮	
`var`	12	66844	(= `&var`)
		⋮	

L'adresse où est stockée la valeur d'une variable `var` s'obtient grâce à l'opérateur d'adresse `&` : cette adresse se note `&var`.

Pointeurs

Les pointeurs

Pointeur : variable contenant l'adresse d'une autre variable.
La déclaration d'un pointeur est de la forme suivante :

```
<type> *<nom>;
```

Par exemple :

```
int *p; // pointe sur une
        // case quelconque
int x = 0;


cout << x << endl; // affiche 0
```

	Mém.	Adr.
		⋮
p	58456	18424
		⋮
x	0	36758
		⋮
*p		58456

La variable `p` est alors un pointeur sur un `int`. La variable entière dont `p` contient l'adresse est dite pointée par `p` et se note `*p`.

Pointeurs

Les pointeurs

Pointeur : variable contenant l'adresse d'une autre variable.
La déclaration d'un pointeur est de la forme suivante :

```
<type> *<nom>;
```

Par exemple :

```
int *p; // pointe sur une
        // case quelconque
int x = 0;
p = &x; // pointe maintenant sur x

cout << x << endl; // affiche 0
```

	Mém.	Adr.
		⋮
p	36758	18424
		⋮
*p,x	0	36758
		⋮
		58456

La variable p est alors un pointeur sur un int. La variable entière dont p contient l'adresse est dite pointée par p et se note *p.

Pointeurs

Les pointeurs

Pointeur : variable contenant l'adresse d'une autre variable.
La déclaration d'un pointeur est de la forme suivante :

```
<type> *<nom>;
```

Par exemple :

```
int *p; // pointe sur une
        // case quelconque
int x = 0;
p = &x; // pointe maintenant sur x
*p = 5; // modifie la valeur de x
cout << x << endl; // affiche 5
```

	Mém.	Adr.
		⋮
p	36758	18424
		⋮
*p,x	5	36758
		⋮
		58456

La variable p est alors un pointeur sur un int. La variable entière dont p contient l'adresse est dite pointée par p et se note *p.

Pointeurs
Les pointeurs

Initialisation

Un pointeur s'initialise à `NULL`.

```
double *ptr = NULL; // aucune case mémoire n'est pointée
```

Types

Un pointeur est toujours lié au type sur lequel il pointe.

```
int *ip;
double *dp;
dp = ip;   // illégal car ip et dp ne sont pas de même type
*dp = *ip; // correct (conversion implicite de type)
```

Références

Les pointeurs

Une **référence** est une variable qui coïncide avec une autre variable.
La déclaration d'une référence est de la forme suivante :

```
<type> &<nom> = <nom_var>;
```

Par exemple :

```
int n = 10;
int &r = n; // r est une référence sur n
```

`n` et `r` ont la **même adresse** : `r` peut être vue comme un alias de `n`.

```
r = 20; // modification de la valeur de n
```

Contrairement aux pointeurs, une référence ne peut être initialisée qu'une seule fois : à la déclaration. Toute autre affectation modifie en fait la variable référencée.

Passage des paramètres dans une fonction

Les pointeurs

Version naïve (passage par valeur) :

```
void swap(int a, int b) // ne fonctionne pas
{
  int aux; // variable auxiliaire servant pour l'échange
  aux = a; a = b; b = aux;
}
```

Version par références :

```
void swap(int &a, int &b) // style C++
{
  int aux;
  aux = a; a = b; b = aux;
}
```

Version par pointeurs :

```
void swap(int *a, int *b) // style C
{
  int aux;
  aux = *a; *a = *b; *b = aux;
}
```

Usages de *, &

Les pointeurs

- &, quand utilisé avec une déclaration de variable (un type)
  = **référence sur...**

```
int &ra = a; // ra est une référence sur a (un alias de a)
```

- *, quand utilisé avec une déclaration de variable (un type)
  = **pointeur sur...**

```
int *pa; // pa est un pointeur sur un entier
```

- &, quand utilisé avec une variable déjà déclarée
  = **adresse de...**

```
&a // adresse de la variable a
```

- *, quand utilisé avec un pointeur déjà déclaré
  = **valeur pointée par...**

```
cout << *pa << endl; // affiche la valeur pointée par pa
```

On dit aussi que * est un opérateur de déréférencement.