Séance 1
Extensions non orientées objet

Valeur par défaut

```
void f (int x, float y, int z = 5);      5 est déclarée valeur par défaut pour z

void f (int x, float y, int z) {         la valeur par défaut est déclarée seulement sur le prototype
      …
}
```

A l'appel on peut omettre z :

```
f (2, 3.1, 4);
f (2, 3.1);                z vaut 5
```

Restriction : valeur par défaut sur un paramètre => valeur par défaut sur les paramètres suivants

```
void f (int x, float y = 2.3, int z = 5);      OUI
void f (int x, float y = 2.3, int z);          NON
```

Surcharge de fonction

Fonctions de même nom, paramètres différents :

```
void afficher(int x);
void afficher(char* ch):

void afficher(int x) {
	printf(%d\n", x);
}

void afficher (char* ch) {
	printf(%s\n", ch);
}
```

```
int a = 12;
afficher(a);
afficher("bonjour");
```

Variables locales

Peuvent être déclarées au milieu des instructions :

```
void f (…) {
      int x;
      float y = 3.1;
      …instructions…
      int z = 2;
      …instructions…
}
```

Application aux boucles for :

```
for (int i = 0; i < 50; i++) {        i est local à la boucle for
      …
}
…
for (int i = 1; i < 20; i++) {
      …
```

Structure

Une structure est un type => définition sans typedef :

```
struct INDIVIDU {
	int age;
	char nom[20];
};
```

```
INDIVIDU x;
x.age = 25;
…
```

Allocation dynamique (new)

```
INDIVIDU* p;
p = new INDIVIDU;              allocation d'une structure INDIVIDU
p->age = 25;
…

int* t;
t = new int [100];             allocation d'un tableau d'entiers
t[0] = 5;
…
```

Libération de la mémoire :

```
delete p;
delete [] t;                   [] dans le cas d'un tableau
```

Référence : définition

```
int x;
int& y = x;
```

int& : type référence sur int
y est une référence de x : y est un alias de x

```
y = 2;          => x et y valent 2
x = 4;          => x et y valent 4
```

Référence : passage de paramètre

Passage de paramètre par référence

```
void saisir (INDIVIDU& indiv) {
      scanf("%d", &indiv.age);
      scanf("%s", indiv.nom);
}
```

le paramètre est une référence sur la variable à saisir
le passage par référence permet de modifier la variable passée

```
INDIVIDU x;
saisir (x);
```

une référence de x est automatiquement passée
pas de copie
le passage par référence est performant en temps d'exécution

Pour les structures, privilégier passage par référence ou par adresse par rapport à passage par valeur.

Utiliser const si variable non modifiée par la fonction : void afficher (const INDIVIDU& indiv);
Utilisable aussi avec le passage par adresse : void afficher (const INDIVIDU* indiv);

Affichages et saisies

```
int x = 124;
char ch[50];
strcpy(ch, "bonjour");

cout << "entier: " << x << "chaine: " << ch << endl;
```

cout : le flux de sortie (le terminal)
<< x : envoie la valeur de la variable vers le flux
endl : retour à la ligne

```
int x;
char ch[50];

cin >> x >> ch;
```

cin : le flux d'entrée (le clavier)
>> x : lit sur le flux et range dans la variable
séparateur entre les saisies : espace ou retour à la ligne

Types de variables reconnues par cin/cout : types du langage (int, float, …) et chaines de caractères.

Include et espace de nom

```
#include <iostream>                pour cin/cout
#include <string.h> ou <cstring>   les fichiers de la bibliothèque C si besoin
…
using namespace std;               utilisation de l'espace de nom std (sinon il faut préfixer cin/cout)
```