

R101 - Amphi05 - C++

Alain Casali

Aix Marseille Univ



1 Avant Propos

- Langage de programmation
- Problème du développeur
- La documentation
- Le C++, cest quoi ?

2 Un premier programme

3 Les commentaires

4 Déclaration, affectation, bloc

5 Entrées / Sorties

6 Opérateur identité et différence

7 Schéma alternatif simple

8 Schéma alternatif complexe

9 Le type booléen

10 Les types entiers

11 Les types réels

Langage de programmation

En informatique, un langage de programmation est une notation conventionnelle destinée à formuler des **algorithmes** et produire des **programmes informatiques** qui les appliquent. D'une manière similaire à une langue naturelle, un langage de programmation est fait d'un **alphabet**, un **vocabulaire**, des **règles de grammaire**, et des **significations**.

Source : http://fr.wikipedia.org/wiki/Langage_de_programmation



Fichiers



Executable

Problème du développeur (1)



C'est une roue !



On ne re-développe **JAMAIS** une fonctionnalité si elle est déjà fournie par le langage (sauf pour des raisons pédagogiques).

Problème du développeur (2)

Exemple : Trier un tableau d'entiers

- Il existe une trentaine d'algorithmes permettant de trier un tableau d'entiers avec des performances différentes.
- Un des algorithmes les plus efficaces est le tri rapide

Taille du tableau	Temps C++	Temps mon implémentation
1 000	~ 0s	~ 0s
10 000	~ 0s	~ 0s
100 000	~ 0s	~ 0s
1 000 000	~ 0s	~ 13s
10 000 000	~ 6s	~ 1200s

La documentation

Elle se lit :

- En ligne (elle est à jour) ;
- En anglais (on évite les erreurs de traduction, si traduction il y a) ;

Elle comporte des exemples.

Objectifs

- Début janvier :
 - Être capable de lire une documentation en anglais ;
 - De la comprendre ;
 - D'adapter les exemples fournis à votre problématique.
- Fin du BUT : Être capable d'écrire une documentation technique en anglais ;



Tous les noms de fonctions, de variables,
doivent être en anglais !

Le C++, cest quoi ?

- 1 Un langage de programmation ;
- 2 Une surcouche (partiellement) objet du C.

Dernière norme du C++ date de 2020.

Plusieurs compilateurs disponibles :

- g++ (support complet)
- clang (support complet)
- VS2017 (support partiel)

En C++ :

- Tout est appel de fonction ou de méthode (maintenant) ;
- Tout est flux ;
- Vive les références (maintenant) et les pointeurs (S3) ;

1 Avant Propos

2 Un premier programme

3 Les commentaires

4 Déclaration, affectation, bloc

5 Entrées / Sorties

6 Opérateur identité et différence

7 Schéma alternatif simple

8 Schéma alternatif complexe

9 Le type booléen

10 Les types entiers

11 Les types réels

Un premier programme (1)

Quel que soit le langage de programmation utilisé, le premier programme est toujours d'afficher, soit :

- Sur un terminal (une console) ;
- Sur une page web ;
- Sur une interface graphique.

la chaîne de caractères "Hello world!" (modulo quelques petits arrangements)

Un premier programme (2)

```
1  /**
2   *
3   * @file    Hello.cxx
4   *
5   * @author  A. CAsali
6   *
7   * @date    12/09/2013
8   *
9   **/
10
11 #include <iostream>
12 using namespace std;
13
14 int main ()
15 {
16     cout << "Hello_World!" << endl;
17     return 0;
18 }
```

1 Avant Propos

2 Un premier programme

3 Les commentaires

4 Déclaration, affectation, bloc

5 Entrées / Sorties

6 Opérateur identité et différence

7 Schéma alternatif simple

8 Schéma alternatif complexe

9 Le type booléen

10 Les types entiers

11 Les types réels

Les commentaires

- ❶ Commentaire sur une unique ligne : utilisation de `//`

```
//this is a one line remark
```

- ❷ Commentaire sur plusieurs lignes : utilisation de `/* */`

```
/* this  
   is a drawn out  
   explanation  
*/
```

- ❸ Autre possibilité :

```
/*this is also a one line remark*/
```

Plus de détail : <http://www.stack.nl/~dimitri/doxygen/>

- 1 Avant Propos
- 2 Un premier programme
- 3 Les commentaires
- 4 **Déclaration, affectation, bloc**
 - Déclaration de variables
 - Affectation
 - Cas des variables constantes
 - Déclaration et initialisation à la volée
 - Bloc et portée de variable
- 5 Entrées / Sorties
- 6 Opérateur identité et différence
- 7 Schéma alternatif simple
- 8 Schéma alternatif complexe
- 9 Le type booléen
- 10 Les types entiers
- 11 Les types réels

Déclaration de variables

Modèle général algorithmique :

```
declarer nomDeVariable : Type;
```

Modèle général C++ :

```
type varIdent; //varIdent (Variable Identifier)
```

Exemple :

```
int i;
```

Affectation

Modèle général algorithmique :

```
nomDeVariable <- valeur ;
```

Modèle général C++ :

```
varIdent = value ;
```

Exemple :

```
i = 10 ;
```

Cas des variables constantes

Modèle général algorithmique :

```
declarer KnomDeVariable : constante type ← valeur;
```

Modèle général C++ :

```
const type KvarIdent = value;
```

Exemple :

```
const int Ki = 10;
```



Tous les noms des constantes doivent commencer par la lettre 'K'

Déclaration et initialisation à la volée

Modèle général algorithmique :

`declarer` nomDeVariable : type \leftarrow Valeur ;

Modèle général C++ :

- ① `type` varIdent = value ;
- ② `type` varIdent (value) ;
- ③ `type` varIdent {value(s)} ;

Exemple :

```
int i = 10;  
int j (5);  
int k {3};
```



Seule la troisième forme permet la déclaration et initialisation à la volée des tableaux.

Bloc et portée de variable

Définition : bloc

Un **bloc** est une suite d'instructions entre { }

Propriété : portée de variable

Une variable n'existe que dans le bloc dans laquelle est déclarée.

Exemple :

```
{
    type i;
    i = value1;
    ...
    {
        type j (value2);
        //i et j existent
        i = value3;
        j = value4;
    }
    //seul i existe
    i = value5;
    j = value6; //<- erreur de compilation
}
```

Bloc externe

Bloc interne

- 1 Avant Propos
- 2 Un premier programme
- 3 Les commentaires
- 4 Déclaration, affectation, bloc
- 5 Entrées / Sorties
 - Saisie clavier
 - Affichage écran
 - Passage à la ligne lors d'un affichage console
- 6 Compression des sorties écran
- 6 Opérateur identité et différence
- 7 Schéma alternatif simple
- 8 Schéma alternatif complexe
- 9 Le type booléen
- 10 Les types entiers
- 11 Les types réels

Saisie clavier

Modèle général algorithmique :

```
saisir (nomDeVariable);
```

Modèle général C++ :

```
cin >> varIdent;
```

cin signifie console input

Le symbole » est appelé un **extracteur**.

Exemple :

```
int i;  
cin >> i; //10
```

Prérequis pour utiliser le clavier :

```
#include <iostream>  
using namespace std;
```

Utilisation de la bibliothèque qui gère les flux (stream) d'entrées / sorties (input / output)

Affichage écran

Modèle général algorithmique :

```
afficher (unLitteral);
afficher (nomDeVariable);
```

Modèle général C++ :

```
cout << literalPattern;
cout << varIdent;
```

`cout` signifie console `output`

Le symbole « est appelé un **injecteur**.

Exemple :

<code>cout << 10;</code>	Littéral entier	10
<code>cout << "une_jolie_chaine";</code>	Littéral chaine de	une jolie chaine
<code>int i;</code>		
<code>i = 10;</code>	caractères	
<code>cout << i;</code>	Variable	10

Prérequis pour utiliser la console :

```
#include <iostream>
using namespace std;
```

Utilisation de la bibliothèque qui gère les flux (stream) d'entrées / sorties (input / output)

Passage à la ligne lors d'un affichage console

Modèle général algorithmique :

```
ligne_suivante;
```

Modèle général C++ :

```
cout << endl;
```

`endl` signifie `end line`

Le symbole `endl` est appelé un **identificateur**.

Exemple :

```
cout << i;  
cout << endl;
```

10



Compression des sorties écran

Il est possible d'injecter plusieurs informations de type différents et donc de compacter l'écriture.

Exemple :

- ```
cout << "Valeur_de_i:_";
cout << i;
cout << endl;
```
- ```
cout << "Valeur_de_i:_"  
  << i  
  << endl;
```



- ❶ Absence du caractère ';' à la fin de la ligne;
- ❷ Tous les injecteurs sont alignés.

- ```
cout << "Valeur_de_i:_ " << i << endl;
```

1 Avant Propos

2 Un premier programme

3 Les commentaires

4 Déclaration, affectation, bloc

5 Entrées / Sorties

6 Opérateur identité et différence

- Opérateur identité
- Opérateur différence

7 Schéma alternatif simple

8 Schéma alternatif complexe

9 Le type booléen

10 Les types entiers

11 Les types réels



# Opérateur identité

## Modèle général algorithmique :

nomDeVariable **vaut** Valeur

## Modèle général C++ :

varIdent == value

## Exemple :

i == 10

Ou mieux

10 == i



On ne compare que des variables et / ou des littéraux du même type.

# Opérateur différence

## Modèle général algorithmique :

nomDeVariable **ne\_vaut\_pas** Valeur

## Modèle général C++ :

varIdent **!=** value

## Exemple :

i **!=** 10

1 Avant Propos

2 Un premier programme

3 Les commentaires

4 Déclaration, affectation, bloc

5 Entrées / Sorties

6 Opérateur identité et différence

7 Schéma alternatif simple

- Schéma alternatif sans condition sinon
- Schéma alternatif avec condition sinon
- Schéma alternatif en cascade

8 Schéma alternatif complexe

9 Le type booléen

10 Les types entiers

11 Les types réels

# Schéma alternatif sans condition sinon

## Modèle général algorithmique :

```
si (condition)
 instruction1;
 instruction2;
 ...
fsi
```

## Modèle général C++ :

```
if (condition)
{
 instruction1;
 instruction2;
 ...
}
```

- condition est une expression booléenne ;
- '{' marque le début d'un bloc d'instruction(s) ;
- '}' marque la fin de ce bloc.



Toutes les instructions à l'intérieur d'un même bloc sont alignées

# Schéma alternatif avec condition sinon (1)

## Modèle général algorithmique :

```
si (condition)
 instruction1;
 instruction2;
 ...
sinon
 instruction3;
 instruction4;
 ...
fsi
```

## Modèle général C++ :

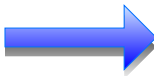
```
if (condition)
{
 instruction1;
 instruction2;
 ...
}
else
{
 instruction3;
 instruction4;
 ...
}
```

# Schéma alternatif avec condition sinon (2)

## Exemple :

```
verifierVie;
verifierArmes;

si (toutEstOK)
 attaquerBoss;
 ...
sinon
 prendrePotion;
 ...
fsi
partir;
```



```
checkLife ();
checkWeapons ();

if (everythingIsOK)
{
 attackBoss ();
 ...
}
else
{
 takePotion ();
 ...
}
leave ();
```

# Schéma alternatif en cascade (1)

## Modèle général algorithmique :

```
si (exprLog1)
 sequ1;
sinon_si (exprLog2)
 sequ2;
sinon_si (exprLog3)
 sequ3;
sinon //tous les autres cas
 sequ4;
fsi
```

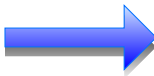
## Modèle général C++ :

```
if (logExp1)
{
 sequ1;
}
else if (logExp2)
{
 sequ2;
}
else if (logExp3)
{
 sequ3;
}
else
{
 sequ4;
}
```

## Schéma alternatif en cascade (2)

### Exemple :

```
si (peuDeVie)
 prendreUnePotion;
sinon_si (jeSuisUnGuerrier)
 attaquerAvecEpee;
sinon_si (jeSuisUnMage)
 lancerUnSort;
sinon
 partir;
fsi
```



```
if (fewOfLife)
{
 takeAPotion ();
}
else if (IAmAWarrior)
{
 attackWithSword ();
}
else if (IAmAWizard)
{
 castASpell ();
}
else
{
 leave ();
}
```



- 1 Avant Propos
- 2 Un premier programme
- 3 Les commentaires
- 4 Déclaration, affectation, bloc
- 5 Entrées / Sorties
- 6 Opérateur identité et différence
- 7 Schéma alternatif simple
- 8 Schéma alternatif complexe
- 9 Le type booléen
- 10 Les types entiers
- 11 Les types réels

# Schéma alternatif complexe (1)

| Condition logique en algo. | Equivalence en C++ |
|----------------------------|--------------------|
| ET                         | &                  |
| OU                         |                    |
| ET_ALORS                   | &&                 |
| OU_SINON                   |                    |

Exemple :

```

si (beaucoupDeVie OU_SINON jeSuisUnGuerrier)
 Attaquer;
fsi

```

se traduit par :

```

if (fullOfLife || IAmAWarrior)
{
 attack ();
}

```

## Schéma alternatif complexe (2)

### Exemple :

```
si (peuDeVie ET_ALORS jeSuisUnGuerrier)
 attaquer;
fsi
```

se traduit par :

```
if (fewOfLife && IAmAWarrior)
{
 attack ();
}
```



- Les règles de propagation de l'opérateur de négation sont les mêmes qu'en algorithmique (amphi2 T11 -> T14);
- Les tableaux de vérité sont les mêmes qu'en algorithmique (amphi2 T15 & T16).

- 1 Avant Propos
- 2 Un premier programme
- 3 Les commentaires
- 4 Déclaration, affectation, bloc
- 5 Entrées / Sorties
- 6 Opérateur identité et différence
- 7 Schéma alternatif simple
- 8 Schéma alternatif complexe
- 9 Le type booléen
  - Identificateur
  - Valeurs
  - Opérations (opérateurs booléens)
  - Opérations (d'identité)
  - Opérateur de négation
- 10 Les types entiers
- 11 Les types réels

# Identificateur

```
bool
```

## Valeurs

```
true false
```

## Opérations (opérateurs booléens)

Produisent un booléen

Voir transparents précédents

```
bool toBe;
bool notToBe;
bool question;
...
```

```
toBe = false;
notToBe = !toBe; // true
question = toBe | notToBe; // always true!
```

# Opérations (d'identité)

Produisent un booléen

`==` `!=`

Exemple :

```
if (toBe == notToBe)
{
 cout << "WTF!";
}
```

## Opérateur de négation

Produit un booléen

Modèle général algorithmique :

```
si (NON condition)
 instruction1;
 instruction2;
 ...
fsi
```

Modèle général C++ :

```
if (!condition)
{
 instruction1;
 instruction2;
 ...
}
```

- 1 Avant Propos
- 2 Un premier programme
- 3 Les commentaires
- 4 Déclaration, affectation, bloc
- 5 Entrées / Sorties
- 6 Opérateur identité et différence
- 7 Schéma alternatif simple
- 8 Schéma alternatif complexe
- 9 Le type booléen
- 10 Les types entiers
  - Identificateur
  - Valeurs
  - Opérations
  - Opérations (de comparaison)
  - Opérations (d'identité)
  - Les différents types d'entiers
- 11 Les types réels

# Identificateur

`int`

## Valeurs

Sous-ensemble des entiers mathématiques

## Opérations

Produisent un entier

`+` `-` `*` `/` `%`



- Troncature du reste de la division ;
- L'opérateur modulo (%) renvoie le reste de la division entière.

## Exemple :

```
int x;
cout << x;
x = 1;
cout << x;
```

0

1

```
x = (x + 2) / 4 * 3;
cout << x;
x = 1;
x = (x + 2) * 3 % 4;
cout << x;
```

0

9



# Opérations (de comparaison)

Produisent un booléen

<    <=    >    >=

## Opérations (d'identité)

Produit un booléen

==    !=

Exemple :

```
int nb1;
nb1 = 12;
int nb2;
nb2 = 5;
int nb3;
nb3 = nb1 / nb2; // nb3 vaut 2
```

```
if (4 == nb3)
{
 cout << "nb3_vaut_4";
}
else if (nb3 >= 2)
{
 cout << ("nb3_>=2");
}
```

# Les différents types d'entiers

| Type                              | Min                        | Max                        | #octets |
|-----------------------------------|----------------------------|----------------------------|---------|
| <code>int</code>                  | -2 147 483 648             | 2 147 483 647              | 4       |
| <code>unsigned [int]</code>       | 0                          | 4 294 967 295              | 4       |
| <code>short [int]</code>          | -32 768                    | 32 767                     | 2       |
| <code>unsigned short [int]</code> | 0                          | 65 535                     | 2       |
| <code>long long [int]</code>      | -9 223 372 036 854 770 000 | 9 223 372 036 854 770 000  | 8       |
| <code>unsigned long</code>        | 0                          | 18 446 744 073 509 551 615 | 8       |
| <code>long [int]</code>           |                            |                            |         |

- 1 Avant Propos
- 2 Un premier programme
- 3 Les commentaires
- 4 Déclaration, affectation, bloc
- 5 Entrées / Sorties
- 6 Opérateur identité et différence
- 7 Schéma alternatif simple
- 8 Schéma alternatif complexe
- 9 Le type booléen
- 10 Les types entiers
- 11 Les types réels
  - Identificateur
  - Valeurs
  - Opérations
  - Opérations (d'identité)
  - Les différents types d'entiers
  - Égalité entre de deux réels

# Identificateur

float

## Valeurs

Sous-ensemble des réels mathématiques

## Opérations

Produisent un réel

+ - \* /



L'opérateur division (/) renvoie la division réelle.

Produisent un booléen

< <= > >=

## Opérations (d'identité)

Produit un booléen

== !=

# Les différents types d'entiers

| Type                       | #octets |
|----------------------------|---------|
| <code>float</code>         | 4       |
| <code>double</code>        | 8       |
| <code>double double</code> | 12      |

Plus il y a doctets, plus la précision est grande (voir cours d'architecture).

# Égalité entre de deux réels



On ne compare jamais deux réels avec l'opérateur d'égalité, même si mathématiquement le résultat est juste !

Exemple : La comparaison suivante a de fortes chances d'être fausse :

```
float four;
four = 4.0;
float squareOfFour;
squareOfFour = ... // Computation of four
if (2.0 == squareOfFour)
{
 cout << "the_value_of_the_square_root_of_4_is_2" << endl;
}
```

Il est préférable de remplacer le test d'égalité par :

```
float eps = 0.000001;
if (abs (2.0 - squareOfFour) < Eps)
{
 cout << "the_value_of_the_square_root_of_4_is_2" << endl;
}
```

# M1102 - Amphi2 C++

Alain Casali    Marc Laporte

Aix Marseille Univ



## 1 Le type caractère

- Identificateur
- Valeurs
- Opérations de comparaison
- Opérations d'identité
- Élément suivant
- Élément précédant
- Fonctions applicables
- Prédicats applicables
- Caractères spéciaux
- Encodage des caractères

## 2 Le type `string`

## 3 Boucle `while`

## 4 Boucle `for`

## 5 Boucle infinie

## 6 Rupture de schéma répétitif

## 7 Schéma séquentiel `switch / case`

## 8 Les sous-programmes

## 9 Cosmétique

## 10 Le type `vector` (tableau de taille variable)



# Identificateur

`char`

## Valeurs

N'importe quel caractère ASCII entre ' '

ASCII : American Standard Code for Information Interchange

## Opérations de comparaison

Produisent un booléen

<   <=   >   >=

## Opérations d'identité

Produisent un booléen

== !=

# Élément suivant

Produit un `car`

La fonction `succ ()` n'existe pas en C++.



Pour passer au caractère suivant, on applique l'opérateur `++` à gauche de la `variable`.

## Exemple

```
char C = 'a';
char Next = ++C;
cout << Next;
```

b

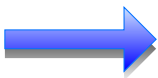
```
char Next = ++'a';
cout << Next;
```

Erreur de compilation

# Élément précédant

Produit un `car`

La fonction `pred ()` n'existe pas en C++.



Pour passer au caractère suivant, on applique l'opérateur `--` à gauche de la **variable**.

## Exemple

```
char C = 'b';
char Prev = --C;
cout << Prev;
```

a

# Fonctions applicables

Produisent un `car`

Pré-requis pour utiliser les fonctions sur les caractères :

```
#include <cctype>
```

- `tolower` (`UnCaract`)

renvoie le caractère en minuscule (uniquement si c'est une majuscule)

- `toupper` (`UnCaract`)

renvoie le caractère en majuscule (uniquement si c'est une minuscule)

## Exemple

```
cout << toupper (Next); B
cout << tolower ('C'); c
```

# Prédicats applicables

Produisent un `bool`

Pré-requis pour utiliser les prédicats sur les caractères :

```
#include <cctype>
```

- `islower` (UnCaract)

renvoie vrai si le caractère est une minuscule

- `isalpha` (UnCaract)

renvoie vrai si le caractère est alphanumérique

D'autres prédicats sont disponibles sur : <http://www.cplusplus.com/reference/cctype/?kw=cctype>

## Exemple

```
if (islower (Next))
{
 cout << "caractere_minuscule";
}
```

# Caractères spéciaux

Liste (non exhaustive) des caractères spéciaux :

| Caractère | Signification                    |
|-----------|----------------------------------|
| '\t'      | tabulation                       |
| '\n'      | Line Feed (saut de ligne)        |
| '\r'      | Carriage Return (retour chariot) |
| '\''      | caractère '                      |

Dans les systèmes compatibles Unix, le caractère matérialisant un retour à la ligne dans un fichier est `'\n'`, alors que dans les systèmes compatibles Windows c'est la combinaison des deux caractères `'\r'` `'\n'`

# Encodage des caractères

Le type `char` est codé sur 1 octet  $\rightarrow$  256 valeurs possibles.

Le type `wchar_t` est codé sur 4 octets  $\rightarrow 2^{32}$  valeurs possibles.

Encodages populaires en France :

- Unicode (compatible avec UTF-8, UTF-16 et UTF-32) ;
- ISO 8859-15.

NB : UTF signifie Universal Transformation Format

1 Le type caractère

2 Le type `string`

- Identificateur
- Valeurs
- Opérations de comparaison
- Opérations d'identité
- Taille d'une `string`
- Accès à un élément
- Opérateur de concaténation `+`

3 Boucle `while`

4 Boucle `for`

5 Boucle infinie

6 Rupture de schéma répétitif

7 Schéma séquentiel `switch / case`

8 Les sous-programmes

9 Cosmétique

10 Le type `vector` (tableau de taille variable)



# Identificateur

string

## Valeurs

N'importe quelle suite de caractères ASCII entre double quotes "

## Opérations de comparaison

Produisent un booléen

<    <=    >    >=

⇒ il existe une relation d'ordre (ordre lexicographique)

## Opérations d'identité

Produisent un booléen

=    !=

# Taille d'une string

Produit un `size_t`  $\Leftrightarrow$  `unsigned long`

Pour connaître la taille d'une `string`, on appelle la méthode `size ()` sur l'objet (la variable) de type `string`.

## Exemple

```
string Str;
cin >> Str; Mario
cout << Str.size(); 5
```

## Accès à un élément

Produit un `char`

On accède au  $(i + 1)^{\text{ème}}$  élément d'une `string` en utilisant `[i]`.



- La première case de la string a pour indice 0;
- La dernière case de la string a pour indice `VarIdent.size () - 1`;
- Aucun contrôle quant à la validité de l'indice n'est effectué lors d'un accès à une case, que ce soit en lecture ou en écriture.

### Exemple

Afficher le contenu d'une `string` en allant à la ligne après chaque caractère

```
pour (i variant_de 0 a Str.size() - 1)
{
 cout << Str [i] << endl;
}
```

# Opérateur de concaténation +

Produit un `string`

L'opérateur de concaténation + est un opérateur binaire. Pour qu'il puisse produire une `string`, on doit avoir :

- À gauche un littéral ou une variable (lhs) ;
- À droite un littéral ou une variable (rhs).

| lhs                 | rhs                 | Compatibilité |
|---------------------|---------------------|---------------|
| <code>char</code>   | <code>string</code> | C++11         |
| <code>string</code> | <code>char</code>   | C++11         |
| <code>string</code> | <code>string</code> | C++           |

## Exemple

```
Str = "0123456789";
cout << Str.size (); 10
Str = Str + "aeiouy\\";
cout << Str.size (); 17
```

1 Le type caractère

2 Le type `string`

3 Boucle `while`

4 Boucle `for`

5 Boucle infinie

6 Rupture de schéma répétitif

7 Schéma séquentiel `switch / case`

8 Les sous-programmes

9 Cosmétique

10 Le type `vector` (tableau de taille variable)

# Boucle `while`

Modèle général algorithmique :

```
tant_que (condition)
faire
 ...
ffaire
```

Modèle général C++ :

```
while (condition)
{
 ...
}
```



Pas de ';' après l'instruction `while`  
(condition).

## Exemple

Algo :

```
declarer i : entier_naturel;
i <- 0;
tant_que (i < taille (Str))
faire
 afficher (Str[i]);
 ligne_suivante;
 i <- i + 1;
ffaire
```

C++ :

```
unsigned i;
i = 0;
while (i < Str.size())
{
 cout << Str [i]
 << endl;
 i = i + 1;
}
```

Modèle général algorithmique :`repetier``...``tant_que (condition)`Modèle général C++ :`do``{``...``}``while (condition);`

Le ';' est **obligatoire** après l'instruction `while` (condition).

**Exemple**

```

unsigned i;
i = 0;
do
{
 cout << Str[i] << endl;
 i = i + 1;
}
while (i < Str.size ());

```



Le code est faux  
si Str est une  
**string** de taille  
nulle !

1 Le type caractère

2 Le type string

3 Boucle while

4 **Boucle for**

- Profil de la boucle for
- Exemple

5 Boucle infinie

6 Rupture de schéma répétitif

7 Schéma séquentiel switch / case

8 Les sous-programmes

9 Cosmétique

10 Le type vector (tableau de taille variable)



# Boucle for

Modèle général algorithmique :

```
pour (NomVar variant_de BorneMin a BorneMax)
faire
 ...
ffaire
```

Cette boucle n'existe pas telle qu'elle en C++, mais la boucle `for` existe.

# Profil de la boucle `for`

## Modèle général C++ :

```
for ([expression1]; [expression2]; [expression3])
{
 instruction(s);
}
```

- `instruction(s)` : un ensemble d'instructions
- `expression1` : une instruction unique, exécutée **avant l'entrée dans l'itération**
- `expression2` : une instruction unique et **booléenne** représentant la **condition de continuation**
- `expression3` : une instruction unique, exécutée **avant la prochaine itération**



Pas de `' ; '` après l'instruction `for ( ; ; )`.

# Exemple

```
for (unsigned i = 0; i < Str.size (); i = i + 1)
{
 cout << Str[i] << endl;
}
```

Équivalent à :

```
for (unsigned i = 0; i < Str.size ();)
{
 cout << Str[i] << endl;
 i = i + 1;
}
```

Équivalent à : ?

```
unsigned i = 0;
for (; i < Str.size (); i = i + 1)
{
 cout << Str[i] << endl;
}
```



Non : portée de variable

1 Le type caractère

2 Le type string

3 Boucle while

4 Boucle for

5 Boucle infinie

6 Rupture de schéma répétitif

7 Schéma séquentiel switch / case

8 Les sous-programmes

9 Cosmétique

10 Le type vector (tableau de taille variable)

# Boucle infinie

Modèle général algorithmique :

```
boucle
 instruction(s);
fboucle
```

Modèle général C++ :

```
for (; ;)
{
 instruction(s);
}
```

```
while (true)
{
 instruction(s);
}
```



Pas de condition de sortie explicite.

1 Le type caractère

2 Le type string

3 Boucle while

4 Boucle for

5 Boucle infinie

6 Rupture de schéma répétitif

- instruction **break**
- instruction **continue**

7 Schéma séquentiel `switch / case`

8 Les sous-programmes

9 Cosmétique

10 Le type vector (tableau de taille variable)

# Rupture de schéma répétitif : instruction `break`

## Modèle général algorithmique :

```
boucle
 instruction (s);
 si (condition) sortie;
 instruction (s);
fboucle
```

## Modèle général C++ :

```
for ([e1]; [e2]; [e3])
{
 instruction(s);
 if (condition) break;
 instruction(s);
}←
```

## Exemple 1 : générer un entier naturel $> 10$

### Exemple

```
declarer N : entier_naturel;
boucle
 afficher ("Entrer une valeur > 10 : ");
 saisir (N);
 si (N > 10) sortie;
 afficher (">10 svp!");
fboucle
```

### Exemple

```
unsigned N;
while (true)
{
 cout << "Entrer une valeur > 10 : ";
 cin >> N;
 if (N > 10) break;
 cout << ">10 svp!";
}
```




# Rupture de schéma répétitif : instruction `continue`

## Modèle général algorithmique :

```
boucle
 instruction (s);
 si (condition) continue;
 instruction (s);
fboucle
```

## Modèle général C++ :

```
for ([e1]; [e2]; [e3]) ← 
{
 instruction(s);
 if (condition) continue;
 instruction(s);
}
```

# Exemple : afficher une lettre sur deux d'un mot saisi

## Exemple

```
string Str;
cin >> Str;
for (unsigned i = 0; i < Str.size (); i = i + 1)
{
 if (0 == i%2) 2 blocs ⇒ 2 indentations
 { ⇒ 2 fois moins lisible
 cout << Str [i];
 }
}
```

## Exemple

```
string Str;
cin >> Str;
for (unsigned i = 0; i < Str.size (); i = i + 1)
{
 if (0 != i%2) continue;
 cout << Str [i]; 1 blocs ⇒ 1 indentations
 ⇒ plus lisible
}
```

# Combinaison de break et de continue

## Exemple

```
while (ICanFrag ())
{
 if (Kill ()) continue;
 if (SomeoneNearMe ()) break;
}
FindAnotherSpot ();
```

1 Le type caractère

2 Le type string

3 Boucle while

4 Boucle for

5 Boucle infinie

6 Rupture de schéma répétitif

7 Schéma séquentiel `switch / case`

- Absence d'instruction `break` dans un `switch/ case`
- Instruction `break` et boucle

8 Les sous-programmes

9 Cosmétique

10 Le type vector (tableau de taille variable)

# Schéma séquentiel switch / case

## Modèle général algorithmique :

```

choix_sur Variable_de_choix
entre
 cas Ens_1 :
 Sequ_1;
 cas Ens_2 :
 Sequ_2;
 cas Ens_3 :
 Sequ_3;
 autre :
 Sequ_4;
fchoix

```

## Modèle général C++ :

```

switch (VarIdent)
{
 case Lit1:
 Instr1;
 break;
 case Lit2:
 Instr2;
 break;
 ...
 default:
 Instr4;
}

```



L'instruction `break` permet la sortie du bloc courant (i.e. la sortie du `switch / case`)



- `VarIdent` doit être de type entier ou caractère;
- Pas de `break` après `Instr4`;

# Exemple : gérer le déplacement (touches uniques)

## Exemple

```
char c;
cin >> c;
switch (c)
{
case 'a':
 MouveLeftward ();
 break;
case 'z':
 MouveUpward ();
 break;
case 'e':
 MouveRightward ();
 break;
case 's':
 MouveDownward ();
 // break;
}
```

Remarque : pas de {} pour les blocs `case`.

# Exemple : gérer le déplacement (touches multiples - 1)

## Exemple

```
char c;
cin >> c;
switch (c)
{
 case 'A':
 case 'a':
 MouveLeftward ();
 break;
 case 'Z':
 case 'z':
 MouveUpward ();
 break;
 case 'E':
 case 'e':
 MouveRightward ();
 break;
 case 'S':
 case 's':
 MouveDownward ();
}
```

# Exemple : gérer le déplacement (touches multiples - 2)

## Exemple

```
char c;
cin >> c;
switch (tolower (c))
{
case 'a':
 MouveLeftward ();
 break;
case 'z':
 MouveUpward ();
 break;
case 'e':
 MouveRightward ();
 break;
case 's':
 MouveDownward ();
}
```



# Exemple : gérer le déplacement (touches multiples - 2)

## Exemple

```
char c;
cin >> c;
switch (tolower (c))
{
case 'a':
 MouveLeftward ();
 // break;
case 'z':
 MouveUpward ();
 break;
case 'e':
 MouveRightward ();
 break;
case 's':
 MouveDownward ();
}
```

a

Déplacement gauche

Déplacement haut

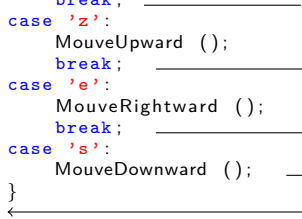


Puisqu'il n'y a pas d'instruction `break`, le `case` est **ignoré**.

# Instruction break et boucle (1)

## Exemple

```
char c;
cin >> c;
for (; 'q' != tolower (c); cin >> c)
{
 switch (tolower (c))
 {
 case 'a':
 MouveLeftward ();
 break;
 case 'z':
 MouveUpward ();
 break;
 case 'e':
 MouveRightward ();
 break;
 case 's':
 MouveDownward ();
 }
 ...
}
```



# Instruction break et boucle (2)

Pas du tout équivalent à :

## Exemple

```
char c;
cin >> c;
for (; 'q' != tolower (c); cin >> c)
{
 if ('a' == tolower (c))
 {
 MouveLeftward ();
 break; _____
 }
 else if ('z' == tolower (c))
 {
 MouveUpward ();
 break; _____
 }
 ...
}←_____
```



Dans une boucle, l'instruction **break** provoque la sortie de la boucle (la plus interne).

- 1 Le type caractère
- 2 Le type string
- 3 Boucle while
- 4 Boucle for
- 5 Boucle infinie
- 6 Rupture de schéma répétitif
- 7 Schéma séquentiel switch / case
- 8 Les sous-programmes
  - Procédure
  - Fonction
  - Passage de paramètres donnés
  - Passage de paramètres [donnés] résultats
- 9 Cosmétique
- 10 Le type vector (tableau de taille variable)

# Procédure

## Modèle général algorithmique :

```
procedure NomProc (Variable1 : [in|out|in_out] Type1 ,
 Variable2 : [in|out|in_out] Type2 , ...);
```

## Modèle général C++ :

```
void ProcName (VarIdent1 : [in|out|in_out] Type1 ,
 VarIdent2 : [in|out|in_out] Type2 , ...);
```

# Exemple

Procédure qui affiche "bonjour" En algorithmique :

```
procedure AffichBonjour ()
debut
 afficher ("bonjour");
 ligne_suivante;
fin
```

En C++ :

```
void ShowHello ()
{
 cout << "bonjour" << endl;
}
// ShowHello ()
```

# Fonction

## Modèle général algorithmique :

```
fonction NomFonct (Variable1 : [in|out|in_out] Type1 ,
 Variable2 : [in|out|in_out] Type2 , ...)
renvoie TypeDeRetour ;
```

## Modèle général C++ :

```
ReturnType FctName (VarIdent1 : [in|out|in_out] Type1 ,
 VarIdent2 : [in|out|in_out] Type2 , ...);
```



A l'intérieur d'une fonction C++, l'instruction algorithmique **renvoie** se traduit par **return**.

# Exemple

Fonction qui affiche "bonjour" et renvoie cette **string**

En algorithmique :

```
fonction AffichEtRenvoieBonjour () renvoie string
debut
 afficher ("bonjour");
 ligne_suivante;
 renvoie "bonjour";
fin
```

En C++ :

```
string ShowHello ()
{
 cout << "bonjour" << endl;
 return "bonjour";
} // ShowHello ()
```



# Passage de paramètres données

## Modèle général algorithmique :

```
SP NomSP (Variable1 : in Type1, Variable2 : in Type2, ...)
[renvoie TypeDeRetour];
```

## Modèle général C++ :

```
[ReturnType] FctName (const Type1 & VarIdent1,
 const Type2 & VarIdent2, ...);
```

Exemple : renvoyer le nombre de caractères d'espacement d'une `string` passée en paramètre

### Exemple

```
unsigned SpaceCount (const string & Str)
{
 unsigned Cpt = 0;
 for (unsigned i = 0; i < Str.size (); i = i + 1)
 {
 if (isspace (Str[i])) Cpt = Cpt + 1;
 }
 return Cpt;
} //SpaceCount ()
```

# Passage de paramètres [donnés] résultats

## Modèle général algorithmique :

```
SP NomSP (Variable1 : [in_] out Type1, Variable2 : [in_] out Type2, ...)
[renvoie TypeDeRetour];
```

## Modèle général C++ :

```
[ReturnType] FctName (Type1 & VarIdent1, Type2 & VarIdent2, ...);
```

Exemple : écrire la procédure qui calcule un entier aléatoire entre les bornes  
Min et Max

### **Exemple**

```
void Rand (const int & Min, const int & Max, int & Res)
{
 srand (time (NULL));
 Res = Min + rand () % (Max - Min);
} // Rand ()
```

1 Le type caractère

2 Le type string

3 Boucle while

4 Boucle for

5 Boucle infinie

6 Rupture de schéma répétitif

7 Schéma séquentiel switch / case

8 Les sous-programmes

9 **Cosmétique**

- Manipulateur setw () ou cadrage à droite
- Manipulateur setfill ()

10 Le type vector (tableau de taille variable)

# Manipulateur setw ( ) ou cadrage à droite

## Pré-requis :

```
#include <iomanip>
```

## Profil :

```
/*undefined*/ setw (int n);
```

## Définition :

L'injection, dans un flux de sortie (`cout`), du manipulateur `setw` (Nb), provoque l'affichage de la variable ou du littéral qui le suit sur Nb caractères (cadrés à droite). Les caractères manquants sont remplacés par des espaces.

## Exemple

```
int a (10), b (20), c (30), d (40);
cout << setw (4) << a << setw (4) << b __10__20
 << endl;
cout << setw (4) << c << setw (4) << d __30__40
 << endl;
```

# Manipulateur `setfill ()`

## Pré-requis :

```
#include <iomanip>
```

## Profil :

```
/*undefined*/ setfill (char c);
```

## Définition :

Le manipulateur `setfill ()` s'utilise avant `setw ()`. Il provoque le remplacement des espaces par le caractère passé en paramètre. `setfill ()` a un effet permanent.

## Exemple

```
int a (10), b (20), c (30), d (40);
cout << setfill ('x') << setw (4) << a
 << setfill ('?') << setw (4) << b << endl xx10??20
 << setw (4) << c
 << setfill ('_') << setw (4) << d
 << endl; ??30__40
```

1 Le type caractère

2 Le type string

3 Boucle while

4 Boucle for

5 Boucle infinie

6 Rupture de schéma répétitif

7 Schéma séquentiel switch /  
case

8 Les sous-programmes

9 Cosmétique

10 Le type vector (tableau de taille  
variable)

- Pré-requis
- Identificateur
- Taille d'un vecteur
- Insertion en fin de vecteur
- Redimensionnement d'un vecteur
- Accès à un élément en lecture / écriture
- La boulette du débutant
- Recopie d'un tableau dans un autre

# Pré-requis

```
#include <vector>
using namespace std;
```

## Identificateur

Modèle général algorithmique :

tableau\_de **UnType**;

Modèle général C++ :

vector <**Type**>

### Exemple

```
vector <int> VInt;
vector <float> VFloat;
```



La taille du vecteur est nulle lors de sa création

## Taille d'un vecteur

Pour connaître la taille d'un `vector` (), on appelle la **méthode** () `size` () `size` () **sur l'objet** (la variable) de type `vector` (même nom que pour connaître la taille d'une `string` ()).

### Exemple

```
cout << VInt.size (); // 0
```

## Insertion en fin de vecteur

Pour insérer des éléments compatibles en fin d'un `vector`, on appelle la **méthode** () `push_back` () **sur l'objet** (la variable) de type `vector` (cette méthode existe aussi pour les `string`). Le vecteur est redimensionné automatiquement.

### Exemple

```
VInt.push_back (0);
VInt.push_back (1);
...
VInt.push_back (9);
cout << VInt.size (); // 10
```



- Même exemple que précédemment en utilisant une boucle `for` :

### Exemple

```
for (int InsVal (0); InsVal < 10; InsVal = InsVal + 1)
{
 VInt.push_back (InsVal);
}
cout << VInt.size (); //10
```

- Insertion de 10 valeurs saisies au clavier :

### Exemple

```
int InsVal;
for (unsigned NbVal = 0; NbVal < 10; NbVal = NbVal + 1)
{
 cin >> InsVal;
 VInt.push_back (InsVal);
}
cout << VInt.size (); //10
```

# Redimensionnement d'un vecteur

Pour redimensionner un `vector`, on appelle la **méthode** `() resize ()` **sur l'objet** (la variable) de type `vector`.

## Exemple

```
VInt.resize (42);
```

Si la taille de l'ancien vecteur  $<$  taille du nouveau (i.e. on ne fait pas de troncature), alors les nouvelles valeurs sont :

| Type                | Valeur par défaut |
|---------------------|-------------------|
| Entier              | 0                 |
| Réel                | 0.0               |
| <code>string</code> | chaîne vide       |
| <code>char</code>   | '\0'              |

## Accès à un élément en lecture / écriture

On accède au  $(i+1)^{\text{ème}}$  élément d'un **vector** en utilisant la notation `[i]`.



- La première case de lu **vector** a pour indice 0;
- La dernière case du **vector** a pour indice `VarIdent.size () - 1`;
- Aucun contrôle quant à la validité de l'indice n'est effectué lors d'un accès à une case, que ce soit en lecture ou en écriture.

### Exemple

Affichage du contenu d'un vecteur

```
for (unsigned i = 0; i < VInt.size (); i = i + 1)
{
 cout << VInt [i];
}
```

Et surtout pas `cout << VInt;`

# La boulette du débutant

## Exemple

```
vector <int> VInt;
VInt [0] = 0;
```

Compilation : pas d'erreur

Exécution : Segmentation fault : 11 (anciennement écran bleu)



A l'aide

VInt est déclaré, mais non dimensionné

⇒ La case d'indice 0 n'existe pas!!

# Recopie d'un tableau dans un autre

On souhaite recopier le contenu de VInt dans un second tableau (VInt2)

```
vector <int> VInt2;
```

- Solution 1 :

```
for (unsigned i = 0; i < VInt.size (); i = i + 1)
{
 VInt2.push_back (VInt[i]);
}
```

- Solution 2 :

```
vector <int> VInt2;
VInt2 = VInt;
// vector <int> VInt2 = VInt;
```

- Solution 3 :

```
vector <int> VInt2 (VInt);
```

# RI.01 - INITIATION AU DÉVELOPPEMENT - AMPHI#07

A. Casali

# PLAN

A. Divers

B. Les types caractères

C. Transtypage

D. Allègement de code pour bloc à instruction unique

E. Allègement de code pour opérateurs mathématiques

F. Opérateurs ++ et --

G. Opérateur ternaire

H. QT Creator

# I. Laser run



La coupe est a nous,  
et on la garde !





# 2. Tuteurat

- Pour qui ?
  - Tous ceux qui le souhaite
    - Si vos notes sont trop basses => obligatoire
  - Quand ?  
<https://amubox.univ-amu.fr/s/mcM4sBDSXGAqRbo>  
Voir Discord du dept

# 3. Changements planning

- Groupe 3 : 1 jour de plus pour le rendu (code) de la prochaine SAE
- Samedi 16 oct : test;
- Lundi 18 oct : 17h45 -> 18h45 - ORE (G1 & G2) : correction du test #1- partie 1 ;
- Mardi 19 oct : 17h45 -> 18h45 - ORE (G3 & G4) : correction du test #1- partie 1 ;
- Lundi 25 oct : 17h45 -> 18h45 - ORE (G1 & G2) : correction du test #1- partie 2 ;
- Mercredi 27 oct : 17h45 -> 18h45 - ORE (G3 & G4) : correction du test #1- partie 2 ;

# 4. Modifications amphi #6

- Ajout boucle `for_each`

# 5. OpenClassRooms



- Vous n'y allez pas;
- Quelles sont les failles de ce tuto (tableau de taille dynamique / variable) :  
<https://openclassrooms.com/fr/courses/1894236-programmez-avec-le-langage-c/1896212-manipulez-les-tableaux>
- Quelles sont les failles de ce tuto (type string) :  
<https://openclassrooms.com/fr/courses/1894236-programmez-avec-le-langage-c/1897113-decouvrez-la-notion-de-programmation-orientee-objet-poo>
- Deal R1.01 :
  - 0.1 point sur moyenne mini-tests #1 (entre 1 et 6(+1)) par incohérence trouvée, les premiers arrivés auront les points;
  - 1 mail (ou équivalent discord) / incohérence, 2 personnes max;
  - Si vous avez faux, -0.1 pt sur moyenne mini-tests #1;
  - Une variable mal nommée n'est pas une erreur;
  - Deadline : prochain amphi.
  - Tout le module : une personne visite ce site => bye bye => absence => -0.1 sur la moyenne

# 6. Mini tests

- Max 48h après votre *rentrée*
- Vos notes : lien à venir

# PLAN

- A. Les types caractères
- B. Transtypage
- C. Allégement de code pour bloc à instruction unique
- D. Allégement de code pour opérateurs mathématiques
- E. Opérateurs ++ et --
- F. Opérateur ternaire
- G. QT Creator

# A. Les types caractères

Il y a plusieurs types de caractères "simples" :

- codés (stockés) sur 1 octet = 8 bits : code ASCII

8 bits =  $2^8$  possibilités = 256 caractères au maximum

128 caractères de base

code ASCII de base

128 caractères supplémentaires

code ASCII étendu

- codés sur 2 octets (16 bits) : code Unicode

16 bits =  $2^{16}$  possibilités = 65536 caractères au maximum

caractères ASCII  
éditables

| binnaire | décimal |                                                             |      |   |      |
|----------|---------|-------------------------------------------------------------|------|---|------|
| 0000     | 0       |                                                             |      |   |      |
| 0000     | ...     |                                                             |      |   |      |
| ...      | 9       | caractère TAB ( <i>TABulation</i> )                         |      |   | '\t' |
| 0000     | 1       | caractère LF ( <i>Line Feed</i> ) et NL ( <i>New Line</i> ) |      |   | '\n' |
| 1001     | 0       |                                                             |      |   |      |
| 0000     | ...     | caractère CR ( <i>Carriage Return</i> )                     |      |   | '\r' |
| 1010     | 3       |                                                             |      |   |      |
| ...      | 1       |                                                             |      |   |      |
| 0000     | 3       | <i>espac</i>                                                | 0100 | 6 | 'A'  |
| 0000     | 2       | <i>e</i>                                                    | 0001 | 5 | 'B'  |
| 0010     | 3       | '!'                                                         | 0100 | 6 | 'C'  |
| 0001     | 3       | '''                                                         | 0010 | 6 | ...  |
| 0010     | 3       |                                                             | 0100 | 6 |      |
| 0010     | 4       | '0'                                                         | 0010 | 7 | 'a'  |
| 0000     | 8       | ' '                                                         | 0001 | 7 | 'b'  |
| 0011     | 4       | 'l'                                                         | 0110 | 9 | 'c'  |
| 0001     | 9       | ' '                                                         | 0010 | 8 | ...  |
| 0011     | 5       | '2'                                                         | 0110 | 9 |      |
| 0010     | 0       | ' '                                                         | 0011 | 9 |      |
| ...      |         |                                                             |      |   |      |

pas de caractère accentué dans le code ASCII de base



Il y a trois types de caractères "simples" codés sur 1 octet

char

unsigned char

signed char

Nous n'utiliserons pour commencer que le type char

```
cout << sizeof (char);
```



1 octet de 8 bits

Nombre d'octets pris par la structure de données. Ne pas confondre avec .size()

⇒

Il y a trois types de caractères "simples" codés sur 1 octet

char                  unsigned char                  signed char

Nous n'utiliserons pour commencer que le type char

```
cout << sizeof (char); 1 octet de 8 bits
```

En C++, les caractères, quel que soit leur type, sont des valeurs entières

⇒

- ☐ il existe une relation d'ordre
- ☐ on peut faire des opérations arithmétiques dessus

```
char carac = 'A';
```

carac "vaut" 65 (code ASCII de A)

```
cout << carac;
```

affichage de A car

l'injecteur << "sait bien" que c'est le symbole A que  
l'utilisateur veut voir apparaître !

```
carac = 65;
cout << carac;
```

affichage de A

```
carac = 0x41;
cout << carac;
```

affichage de A

```
cout << carac + 1;
```

affichage de 66

```
carac = carac + 1;
cout << carac;
```

affichage de B

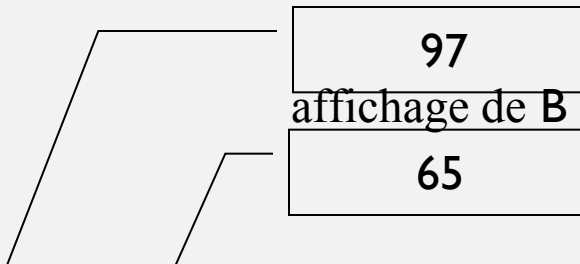
```
const char kShift = 'a' - 'A';
```

```
carac = 'C';
cout << carac + kShift;
```

affichage de 99

```
carac = carac + kShift;
cout << carac;
```

affichage de c



# PLAN

- A. Les types caractères
- B. Transtypage
- C. Allègement de code pour bloc à instruction unique
- D. Allègement de code pour opérateurs mathématiques
- E. Opérateurs ++ et --
- F. Opérateur ternaire
- G. QT Creator

# B. Transtypage

## B.1 Définition

"*Conversion*" d'une valeur d'un type (source) dans un autre (cible).

## B.2 Déclaration

```
typeCible (valeurDuTypeSource);
```

### Exemple :

```
carac = 'C';
cout << carac + kShift << endl 99
 << int (carac + kShift) << endl 99
 << char (carac + kShift) << endl; c
```

### Exemple : Initialisation d'une string contenant toutes les minuscules

```
string minusc;
for (unsigned i (0); i < 26; i = i +1)
{
 minusc = minusc + char ('a' + i);
}
cout << minusc << endl; abcdefg...
```

Transtypage explicite



### Exemple : Affichage des codes ACSII d'une string


```
for (const unsigned & val : minusc)
{
 cout << val << '/'; 97/98/99/100/101/102/103/...
}
```

Transtypage implicite



### B.3 Transtypage implicite impossible

```
for (int i (0); i < minusc.size (); i = i + 1)
{
 cout << minusc [i];
}
```



warning: comparison between signed and unsigned integer expressions

**Solution** : utiliser un transtypage explicite

```
for (int i (0); i < int (minusc.size ()); i = i + 1)
{
 cout << Minusc [i];
}
```

ou

```
for (int i (0); unsigned (i) < minusc.size (); i = i + 1)
{
 cout << Minusc [i];
}
```

Ou encore mieux :

```
for (size_t i (0); i < minusc.size (); i = i + 1)
{
 cout << minusc [i];
}
```



Pourquoi s'embêter à faire des transtypes.



# PLAN

- A. Les types caractères
- B. Transtypage
- C. Allègement de code pour bloc à instruction unique
- D. Allègement de code pour opérateurs mathématiques
- E. Opérateurs ++ et --
- F. Opérateur ternaire
- G. QT Creator

# C. Bloc à unique instruction

## C.I Règle

Lorsqu'un bloc contient une unique instruction, les accolades { } l'entourant sont superflues

Exemple (dans un bloc alternatif) :

```
int i;
cin >> i;
if ((i % 2) == 0)
{
 cout << "le chiffre"
 << " est pair";
}
else
{
 cout << "le chiffre"
 << " est impair";
}
```

```
int i;
cin >> i;
if ((i % 2) == 0)
 cout << "le chiffre est pair";
else
 cout << "le chiffre est impair";
```

### Exemple (dans un bloc répétitif) :

```
for (unsigned i (0); i < minusc.size (); i = i + 1)
 cout << minusc [i];
```

### Exemple (bloc alternatif dans un bloc répétitif) :

```
for (unsigned i (0); i < 42; i = i + 1)
```

```
 if (rand (0, 2) == 0)
```

```
 cout << "le chiffre est pair";
```

```
 else
```

```
 cout << "le chiffre est impair";
```

Vu comme une unique  
instruction par le  
schéma alternatif

Vu comme une unique instruction par le schéma répétitif

# PLAN

- A. Les types caractères
- B. Transtypage
- C. Allègement de code pour bloc à instruction unique
- ~~D. Allègement de code pour opérateurs mathématiques~~
- E. Opérateurs ++ et --
- F. Opérateur ternaire
- G. QT Creator

# D. Opérateurs mathématique

```
i = i + (expression_entiere);
```

peut être écrite :

```
i += (expression_entiere);
```

ou plus simplement :

```
i += expression_entiere;
```

Exemple :

```
int i = 25;
int incr;
cin >> incr;
// ...
i += incr * 2; // ou i = i + (incr * 2);
```

existe aussi pour autres opérateurs binaires



arithmétiques

```
i = i - (expression_entiere);
```

peut être écrite :

```
i -= (expression_entiere);
```

```
i = i / (expression_entiere);
```

peut être écrite :

```
i /= (expression_entiere);
```

`*=`, `%=` existent aussi

## Examples :

```
i = incr;
```



```
i = i - incr;
```

De même :

```
i /= incr;
```



```
i = i / incr;
```



```
i -= incr - 1;
```



```
i = i - (incr - 1);
```

三

```
i = i - incr + 1;
```

et :

```
i /= incr * 2;
```

≡

```
i = i / (incr * 2);
```



```
i = i / incr / 2;
```



`i *= incr / 2;`  $\equiv$  `i = i * (incr / 2);`

mais n'est pas du tout équivalent à

`i = i * incr / 2;`

### Exemple :

`i = 10;`

`incr = 5;`

`i *= incr / 2;`

$\equiv$

`i = i * (incr / 2);`  
`= 10 * (5 / 2);`  
`= 10 * 2;`  
`= 20;`

$\neq$

`i = i * incr / 2;`  
`= 10 * 5 / 2;`  
`= 50 / 2;`  
`= 25;`





L'opérateur `+=` existe aussi pour les strings  
avec la même sémantique :

```
chaine1 += chaine2; ≡ chaine1 = chaine1 + (chaine2);
```

~~surcharge de l'opérateur `+` : concaténation~~

surcharge de l'opérateur `+=` arithmétique

```
string s1 ("Coucou");
```

```
string s2 ("moi");
```

```
s1 += " c'est " + s2;
```

≡

```
s1 = s1 + (" c'est " + s2);
```

# PLAN

- A. Les types caractères
- B. Transtypage
- C. Allègement de code pour bloc à instruction unique
- D. Allègement de code pour opérateurs mathématiques
- E. Opérateurs ++ et --
- F. Opérateur ternaire
- G. QT Creator

# E. Opérateurs ++ et --

## E.I Pré-incrémentation - Pré-décrémentation

```
i = i + 1; // incrémentation
```

peut être écrite :

```
i += 1; // incrémentation
```

ou même :

```
++i; // incrémentation
```

L'expression `++i` a un double effet :

`i` est incrémenté de 1

l'expression a pour valeur la **nouvelle valeur** de `i`

`++` est ici l'opérateur de **pré-incrémentation**

`--i` : opérateur de **pré-décrémentation** similaire

## Exemple d'utilisation

```
string str ("Bonjour");
unsigned i = 3;
cout << str [++i];

cout << str [++i];
```

```
i vaut 3;
i vaut 4
affichage de str [4] : 'o'
i vaut 5
affichage de str [5] : 'u'
```



## Effet de bord (*side effect*)

```
i = 3;
cout << str [++i];
cout << str [++i];
```

affichage de 'o'  
affichage de 'u'

pas équivalent à :

```
i = 3;
cout << str [++i] << str [++i];
```

ordre **d'évaluation** des opérandes : de droite à gauche

2

1

⇒

i = 5

i = 4

⇒

ordre **d'injection** des opérandes : de gauche à droite

3

4

⇒

cout << str [5] << str [4];      affichage de 'u' suivi de 'o'

autre exemple : afficher un mot lu au clavier  
**caractère/caractère à l'envers**

```
string motLu;
cin >> motLu;
```

```
unsigned i = motLu.size ();
```

```
for (; ;)
{
```

```
 if (0 == i) break;
```

```
 i = i - 1;
```

```
 cout << motLu [i];
```

```
}
```

```
// suite du programme
```

≡      cout << motLu [--i];

⇒

```
string motLu;
cin >> motLu;
for (unsigned i = motLu.size (); i > 0;)
{
 cout << motLu [--i];
}
// suite du programme
```

**ou même :**

```
string motLu;
cin >> motLu;

for (unsigned i = motLu.size (); i > 0;)
 cout << motLu [--i];

// suite du programme
```

## E.2 Post-**i**ncrémentation - Post-**dé**crémentation

L'expression `i++` a un double effet :

l'expression a pour valeur l'**ancienne valeur** de `i`

**puis** `i` est incrémenté de 1

**++** est ici l'opérateur de **post-incrémentation**

**--** : opérateur de **post-décrémentation** similaire

```
string str ("Bonjour");
unsigned i = 3;
cout << str [i++];

cout << str [i++];
```

```
i vaut 3;
affichage de str [3] : 'j'
i vaut 4
affichage de str [4] : 'o'
i vaut 5
```



exemple : afficher un mot lu au clavier  
**caractère/caractère à l'envers**

```
string motLu;
cin >> motLu;

for (unsigned i = motLu.size (); i-- > 0;)
 cout << motLu [i];
```

# PLAN

- A. Les types caractères
- B. Transtypage
- C. Allègement de code pour bloc à instruction unique
- D. Allègement de code pour opérateurs mathématiques
- E. Opérateurs ++ et --
- F. Opérateur ternaire
- G. QT Creator

# F. Opérateur ternaire

## F.I Utilité

Eviter les redondances

```
char C;
cin >> C;

if (' ' == C || '\t' == C)
 cout << C << " est un espace" << endl;
else
 cout << C << " n'est pas un espace" << endl;
```

```
char C;
cin >> C;
cout << C;

if (' ' == C || '\t' == C)
 cout << " est";
else
 cout << " n'est pas";

cout << " un espace" << endl;
```

## F.2 Simplification du code C++ : opérateurs ? :

### Syntaxe



### Sémantique

*expression\_1* ? *expression\_2* :  
*expression\_3*

La valeur renvoyée par cet opérateur est égale à  
*expression\_2* ou *expression\_3*  
selon que *expression\_1* est vrai ou faux

⇒

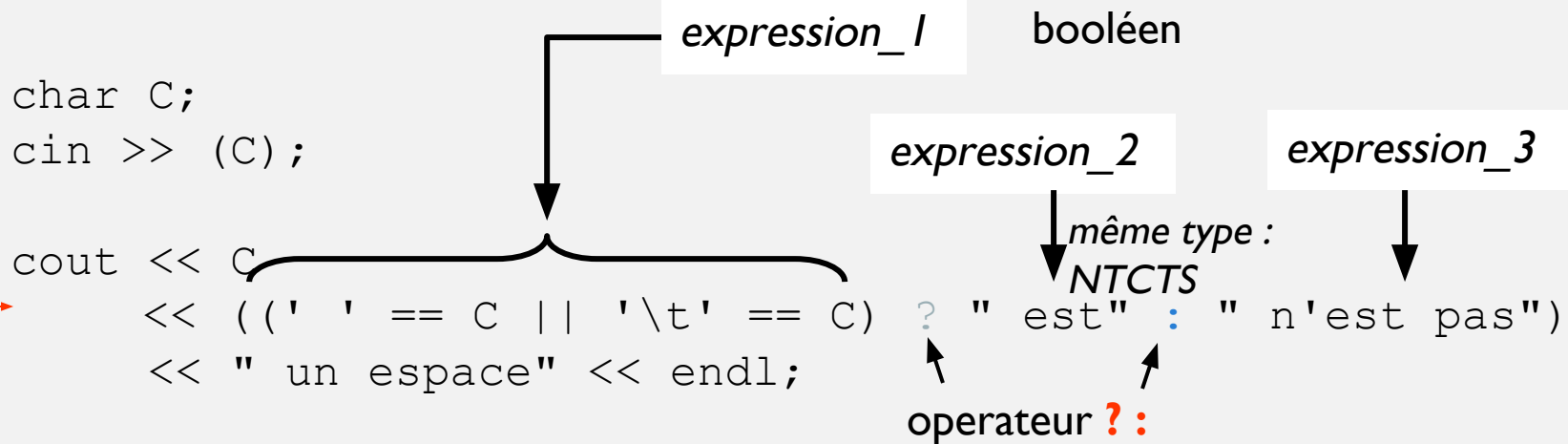
*expression\_1* doit être de **type booléen**,

*expression\_2* et *expression\_3* doivent être **de même type**,  
qui est aussi celui du résultat

## Exemple :

```
char C;
cin >> C;

cout << C;
if (' ' == C || '\t' == C)
 cout << " est";
else
 cout << " n'est pas";
cout << " un espace" << endl;
```



résultat injecté : Littéral chaîne de caractères

Un des avantages : **une seule instruction**

⇒ `if ... , for ...`

### Example :

```
double X, AbsX;
cin >> X;
```

```
if (X < 0.0)
 AbsX = -X;
else
 AbsX = X;
```

```
double X, AbsX;
cin >> X;
```

```
AbsX = (X < 0.0) ? -X : X;
```

### Exemple :

```
unsigned validRang (char codeOp, unsigned rang, unsigned
max)
{
 if ('I' == codeOp)
 return min (rang, max);
 else
 return min (rang, max - 1);

} // validRang()
```

mieux :

```
return ('I' == codeOp) ? min (rang, max)
 : min (rang, max - 1);
```

encore mieux :

```
return min (Rang, ('I' == codeOp) ? Max : Max - 1);
```



Ne pas perdre en lisibilité<sup>1</sup>

## Exemple :

```
#include <cctype> // toupper(), isxdigit(), isalpha()

short hexa2Dec (const char C)
{
 if (! isxdigit (C)) return -1;
 if (isalpha (C))
 return toupper (C) - 'A' + 10;
 else
 return C - '0';
} // hexa2Dec ()
```

```
if (! isxdigit (C)) return -1;
```

```
return isalpha (C) ? toupper (C) - 'A' + 10 : C - '0';
```

```
return (!isxdigit (C)) ? -1
```

```
 : isalpha (C) ? toupper (C) - 'A' + 10
 : C - '0';
```

← 62 colonnes →



### F.3 Attention

```
cout << X;
if (X >= 0) cout << " non";
cout << " négatif";
```

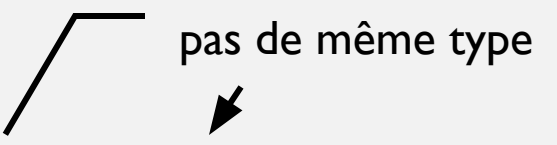
```
cout << X << ((X >= 0) ? " non" :) << " négatif";
```



pas de même type

FAUX

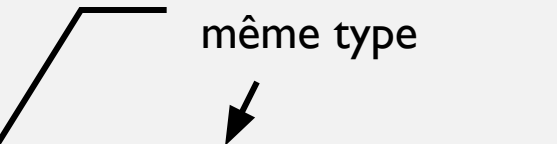
```
cout << X << ((X >= 0) ? " non " : ' ') << "négatif";
```



pas de même type

FAUX

```
cout << X << ((X >= 0) ? " non " : " ") << "négatif";
```



même type

JUSTE

# PLAN

- A. Les types caractères
- B. Transtypage
- C. Allègement de code pour bloc à instruction unique
- D. Allègement de code pour opérateurs mathématiques
- E. Opérateurs ++ et --
- F. Opérateur ternaire
- G. QT Creator

# RI.01 - INITIATION AU DÉVELOPPEMENT - AMPHI#08

A. Casali

# PLAN

**A. Les structs**

**B. minGL 2.0**

**C. Lectures au clavier**

**D. Conversion**

# A Les struct

## A.1 Objectif

Créer son propre type permettant de regrouper différents types.

## A.2 Définition

```
struct newName {
 type1 varIdent1;
 type2 varIdent2;
 type3 varIdent3;
```

Attention au ‘;’

```
}; //newName
```

## A.3 Exemple

```
struct pos {
 unsigned abs;
 unsigned ord;
}; //pos
```

```
struct RGBcolor
{
 short Red;
 short Green;
 short Blue;
};
```

```
struct perso
{
 string name;
 string biblio;
 short age;
 unsigned XP;
 int lifePoint;
};
```

## A.4 Déclaration

```
nameOfTheStruct varIdentOfTheStruct;
```

```
pos aPos;
RGBColor color;
perso bibi;
```

## A.5 Accès à un élément

Notation pointée

```
varIdentOfTheStruct.varIdent
```

```
aPos.abs = 10;
aPos.ord = 10;
```

Autant d'accolades que de  
champs à remplir

## A.6 Déclaration et initialisation à la volée

```
nameOfTheStruct varIdentOfTheStruct {valueList};
```

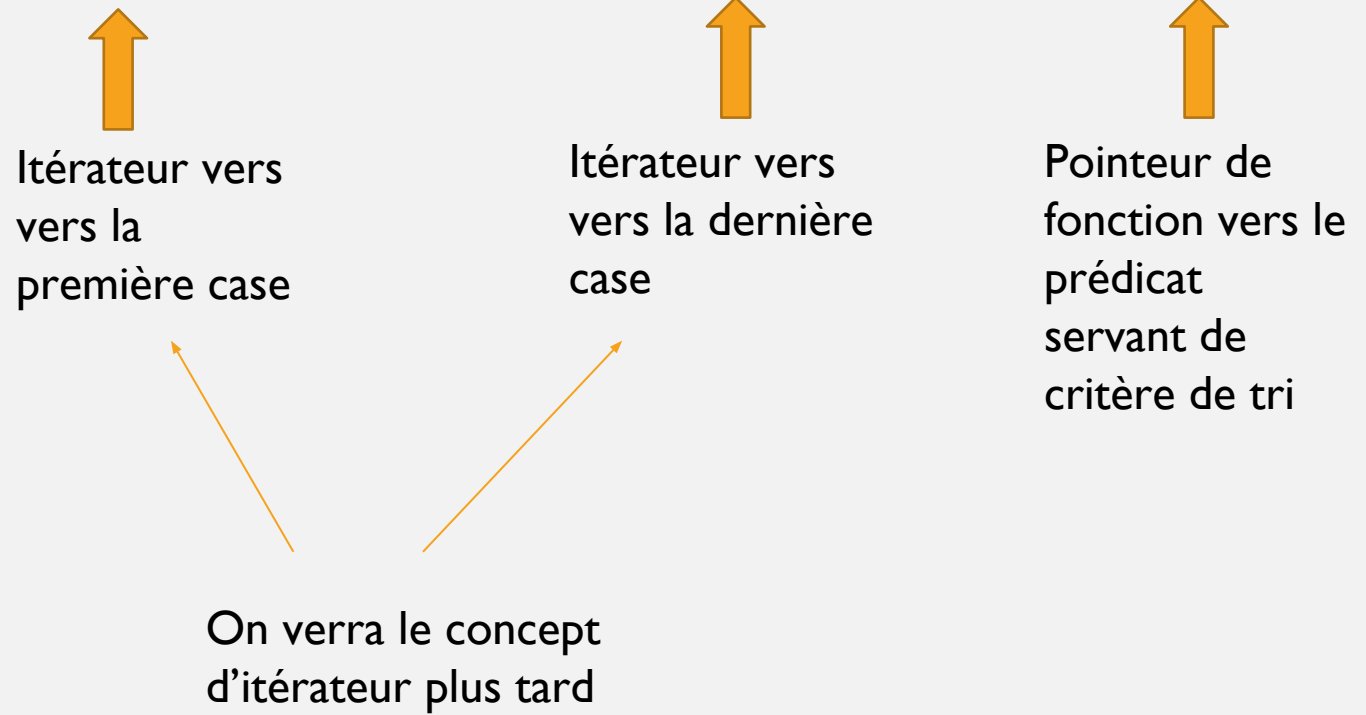
```
pos aPos {10, 10};
RGBColor color {0, 0, 0};
perso bibi {"casali", "", 44, 100, -1};
```

## A.7 Trier un vecteur de struct

```
vector<pos> vPos;
vector<RGBColor> vColor;
vector<perso> vPerso;
```

Comment fait on pour tirer ce tableau ?

```
sort (varIdentOfTheStruct.begin(), varIdentOfTheStruct.end(), criteria);
```



## A.7 Trier un vecteur de struct

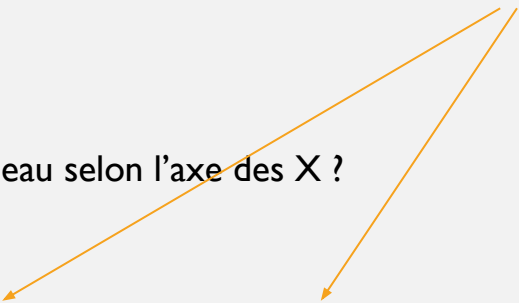
```
vector<pos> vPos;
```

Comment fait on pour tirer ce tableau selon l'axe des X ?

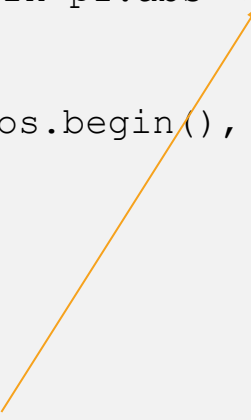
```
bool orderByX (const pos & p1, const pos & p2){
 return p1.abs <= p2.abs;
}
```

```
sort (vPos.begin(), vPos.end(), orderByX);
```


const / référence obligatoires



L'égalité est possible



Pas de parenthèse –  
uniquement le nom de la  
fonction





## A.7 Trier un vecteur de struct

```
vector<pos> vPos;
```

Comment fait on pour tirer ce tableau selon l'axe des Y ?

```
bool orderByX (const pos & p1, const pos & p2){
 return p1.abs <= p2.abs;
}
```

```
bool orderByY (const pos & p1, const pos & p2){
 return p1.ord <= p2.ord;
}
```

```
sort (vPos.begin(), vPos.end(), orderByY);
```

Même paramètres que précédemment

Seul le critère de tri change

## A.7 Trier un vecteur de struct – multi critères

```
vector<pos> vPerso;
```

**Comment fait on pour tirer ce tableau selon le nom ?**

```
bool orderByName (const perso & p1, const perso & p2){
 return p1.name <= p2.name;
}
```

```
sort (vPerso.begin(), vPerso.end(), orderByName);
```

**Comment fait on pour tirer ce tableau selon l'âge?**

```
bool orderByAge (const perso & p1, const perso & p2){
 return p1.age <= p2.age;
}
```

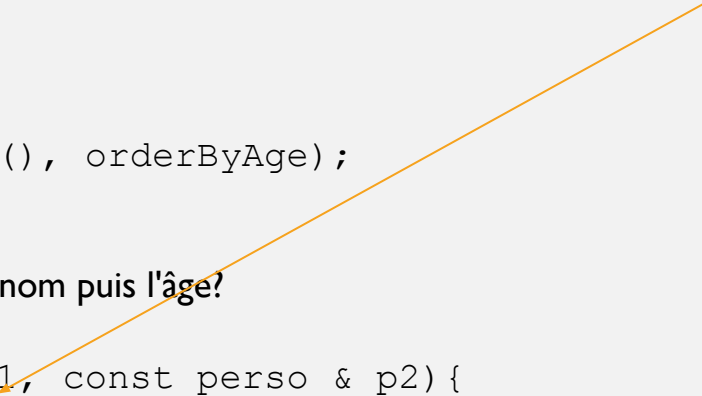
```
sort (vPerso.begin(), vPerso.end(), orderByAge);
```

**Comment fait on pour tirer ce tableau selon le nom puis l'âge?**

```
bool orderByAge (const perso & p1, const perso & p2){
 return p1.name <= p2.name && p1.age <= p2.age;
}
```

```
sort (vPerso.begin(), vPerso.end(), orderByAge);
```

**On enchaîne les conditions de tri**



## A.8 struct vs COO

Une struct est une classe pour laquelle tout (donnée membre / fonction public) est public

- ➡ on peut déclarer des fonctions dans une `struct`;
- ➡ à ne pas faire si vous ne maîtrisez pas !

# PLAN

**A. Les structs**

**B. minGL 2.0**

**C. Lectures au clavier**

**D. Conversion**

# B. minGL V2.0

## B.1 C'est quoi? Pourquoi?

- Surcouche de freeGLUT (surcouche d'OpenGL) sous linux;
- Gestion des sorties simplifiée : affichage de type primitif & composé;
- Gestion des entrées / sorties;
- Pas de gestion des pointeurs.

## B.2 Réalisations attendues

- Sans animation :
  - Spirale de ULAM.
- Avec animation :
  - Tri des vecteurs;
  - Tours de Hanoi;
  - SAEI.02 : *space invaders*



Commencer à travailler ?

# B. minGL V2.0

## B.3 Prérequis mathématiques

- Savoir se positionner dans un repère orthonormé d'entiers naturels.  $(0, 0)$

## B.4 Prérequis informatiques (Linux)

```
sudo apt-get install freeglut3-dev
```

[https://github.com/alain13100/MinGL2\\_IUT\\_AIX.git](https://github.com/alain13100/MinGL2_IUT_AIX.git)

## B.5 Prochaine SAE

- On ne développe plus des jeux comme cela en 2021 ! (source : papa de Mario sur un doc Netflix) – cf. p. tut 20/21 & 21/22;
- Correction par un dev Ubisoft pour ceux qui le souhaitent;
  - ➡ Prévoir les binômes de la 1<sup>ère</sup> SAE de dev en fonction si vous voulez vous faire plaisir;
  - ➡ si (NON minGL) : vous n'aurez accès qu'au terminal Unix.

## B.6 Création de la fenêtre

```
minGL(const unsigned & width = 640,
 const unsigned & height = 480,
 const string & name = string());
```

Valeurs par  
défaut

```
minGL window (400, 200, "une fenetre");
```



Ne pas redimensionner la fenêtre

## B.7 Initialisation des composants graphiques

Il faut appeler successivement les méthodes `initGLUT()` et `initGraphic()` sur l'objet de type `minGL`.

```
window.initGlut();
window.initGraphic();
```

## B.8 Affichage

Il faut appeler la méthode `updateGraphic()` sur l'objet de type `minGL`.  
Equivalent du `<< endl;` dans le terminal.

```
window.updateGraphic();
```

## B.9 Taille de la fenêtre

- **Longueur** : appeler la méthode `getWindowWidth()` sur l'objet de type `minGL`.
- **Largeur** : appeler la méthode `getWindowHeight()` sur l'objet de type `minGL`.

```
cout << window.getWindowWidth() << '\t'
 << window.getWindowHeight();
```



On pourrait définir un pixel comme suit :

```
struct pixel {
 pos aPos;
 RGBcolor color;
};
```

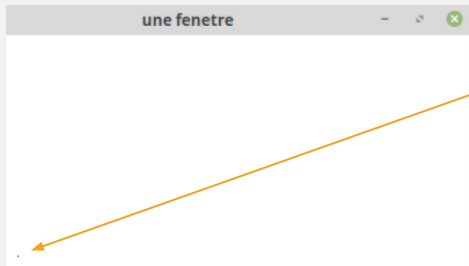


L'élément en position (0,0) se trouve en bas à gauche de la fenêtre

appeler la méthode `setPixel()` sur l'objet de type `minGL`. Cette méthode prend 2 paramètres :

1. Une position;
2. Une couleur de type RGB

```
window.setPixel(pos (10, 10), RGBcolor (0, 0, 0));
```



Le pixel est ici!!

## B.11 Couleurs prédéfinies

```
const RGBcolor KBlack { 0, 0, 0};
const RGBcolor KWhite {255, 255, 255};
const RGBcolor KRed {255, 0, 0};
const RGBcolor K Lime {0 , 255, 0};
const RGBcolor KBlue {0 , 0, 255};
const RGBcolor KYellow {255, 255, 0};
const RGBcolor KCyan {0 , 255, 255};
const RGBcolor KMagenta {255, 0, 255};
const RGBcolor KSilver {192, 192, 192};
const RGBcolor KGray {128, 128, 128};
const RGBcolor KMaroon {128, 0, 0};
const RGBcolor KOlive {128, 128, 0};
const RGBcolor KGreen {0 , 128, 0};
const RGBcolor KPurple {128, 0, 128};
const RGBcolor KTeal {0 , 128, 128};
const RGBcolor K Navy {0 , 0, 128};
```

## Dessin d'un carré de 10 pixel de côté en bas à gauche

```
minGL window (400, 200, "une fenetre");
window.initGlut();

window.initGraphic();

for (unsigned i (0); i < 10; ++i)
 for (unsigned j(0); j < 10; ++j)
 window.setPixel(pos (i,j), KRed);

window.updateGraphic();
window.get_key();
```

↑  
Demande à l'utilisateur  
d'appuyer sur une  
touche pour faire  
l'action suivante.



### B.13 Figure primitive : triangle

Pour dessiner un triangle, on a besoin :

- Des coordonnées des 3 points;
- De la couleur de la bordure;
- De la couleur du fond.

```
triangle(const pos & pos1, const pos &pos2, const pos &
pos3, const RGBcolor & borderCol, const RGBcolor & fillCol);
```

```
triangle tri (pos (50, 50), pos (100, 50), pos (83, 15),
 KBlack, KBlack);
window << tri;
window.updateGraphic();
```

On demande l'affichage  
du triangle



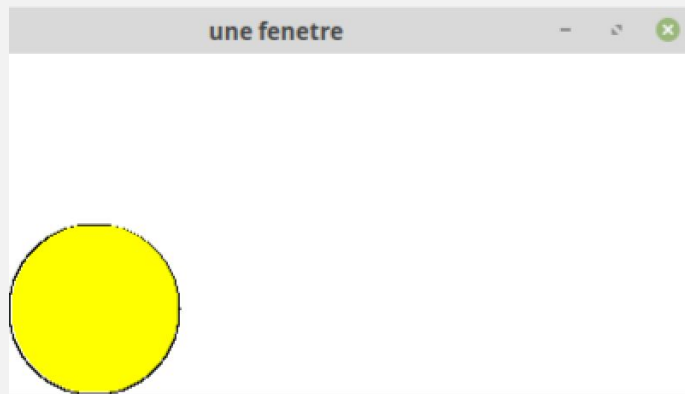
## B.14 Figure primitive : cercle

Pour dessiner un cercle, on a besoin :

- Des coordonnées de son centre;
- De son rayon;
- De la couleur de sa bordure;
- De la couleur du fond.

```
circle(const pos & _pos, const unsigned & rad, const
RGBcolor & borderCol, const RGBcolor & inCol);
```

```
circle cir (pos (50, 50), 50, KBlack, KYellow);
window << cir;
window.updateGraphic();
```



## B.15 Figure primitive : rectangle (1)

Pour dessiner un rectangle, on a besoin :

- Des coordonnées de 2 points non consécutifs;
- De la couleur de bordure;
- De la couleur de fond.

```
rectangle(const pos & pos1, const pos & pos2, const RGBcolor
& borderCol, const RGBcolor & inCol);
```

```
rectangle rec (pos (1,1), pos (10, 10), KBlue, KRed);
window << rec;
window.updateGraphic();
```



## B.15 Figure primitive : rectangle (2)

Pour dessiner un rectangle, on a besoin :

- Des coordonnées du point en bas à gauche;
- De la longueur et la largeur;
- De la couleur de bordure;
- De la couleur de fond.

```
rectangle(const pos & pos, const int & width, const int & height, const RGBcolor & borderCol, const RGBcolor & inCol);
```

```
rectangle rec (pos (1,1), 9, 9, KBlue, KRed);
window << rec;
```



## Tableau de figure *simple*

### Déclaration :

```
figure varIdent;
figure pacMan;
```

### Ajout d'une figure primitive :

Il faut appeler la méthode `.add ()` sur un objet de type `figure` avec comme paramètre une figure primitive (triangle, cercle, rectangle).

```
pacMan.Add(cir);
pacMan.Add(circle (pos (75, 75), 5, KBlack, KBlack));
pacMan.Add(triangle(pos (50, 50), pos (100, 50), pos (83, 15),
 KBlack, KBlack));
```

L'œil

### Injection possible dans une fenêtre graphique

```
window << pacMan;
```

La bouche





## B.17 Opérateur mathématique +

Opérateur mathématique +, appliqué entre une figure et une position, provoque le décalage de la figure d'autant de pixel que les valeurs de la position (considérée ici comme un vecteur).

```
window << PacMan + pos (100, 100);
```

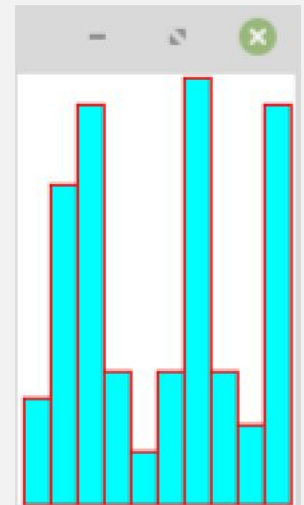


**L'opérateur + existe aussi entre 2 positions :**

```
pos P1 (a,b) , P2 (c,d) ;
pos P3 = P1 + P2 ;
P3.abs = P1.abs + P2.abs ;
P3.ord = P1.ord + P2.ord ;
```

**Exemple : afficher l'histogramme d'un tableau d'entier naturel**

```
pos aPos (2,0) ;
for (const unsigned & val : V)
{
 rectangle unRectangle (aPos, longueurDUnCarre,
val*longueurDUnCarre, KREd, KCyan) ;
 window << unRectangle ;
 aPos = aPos + pos (longueurDUnCarre, 0) ;
}
```



## B.18 Opérateur mathématique \*

Opérateur mathématique \*, appliqué entre une figure et un réel, provoque le grossissement / la réduction de la figure du facteur du réel.

```
window << PacMan * 0.25;
```



```
window << pos (100, 100) + PacMan * 0.25 ;
```



Il faut appeler la méthode `clearscreen()` sur l'objet de type `minGL`.

```
for (unsigned i (0); i < 100; ++i)
 window << pos (100 + i, 100) + PacMan * 0.25 ;
```



```
for (unsigned i (0); i < 100; ++i) {
 window. clearscreen();
 window << pos (100 + i, 100) + PacMan * 0.25 ;
}
```

- A vous d'explorer la bibliothèque (vous avez les sources) ;
- L'anti-aliasing est aussi pris en compte;
- Merci à C. Drift et A. Sollier (étudiant 2018-2020).

# PLAN

**A. Les structs**

**B. minGL 2.0**

**C. Lectures au clavier**

**D. Conversion**

# C. Lectures claviers

## C.1 Lire un caractère

```
char c;
c = cin.get();
```

## C.2 Lire un mot

```
string s;
cin >> s;
```

## C.3 Lire plusieurs mots

```
string s;
getline (cin, s);
```

Obligatoirement  
de type string



# PLAN

**A. Les structs**

**B. minGL 2.0**

**C. Lectures au clavier**

**D. Conversion**



# D. Conversion C++11

Pour convertir une `string` vers un `int` on utilise la fonction `stoi` () de profil :

```
int stoi (string, unsigned*, int)
```

1<sup>er</sup> paramètre : la chaîne à convertir

2<sup>ème</sup> paramètre : un pointeur (=> `nullptr`, par défaut `nullptr`)

3<sup>ème</sup> paramètre : la base utilisée pour la conversion (par défaut 10)

Exemple :

```
string str1 = "10";
```

```
string str2 = "3.14159";
```

```
string str3 = "10xyz";
```

```
cout << stoi (str1) << ' ' 10
 << stoi (str2) << ' ' 3
 << stoi (str3) ; 10
```

```
cout << stoi ("10", nullptr, 2); 2
```

Sur le même profil, on trouve les fonctions :

| Nom                   | Type de retour     |
|-----------------------|--------------------|
| <code>stol()</code>   | long               |
| <code>stoll()</code>  | long long          |
| <code>stoul()</code>  | unsigned long      |
| <code>stoull()</code> | unsigned long long |
| <code>stof()</code>   | float              |
| <code>stod()</code>   | double             |
| <code>stold()</code>  | long double        |



La string passée en paramètre ne doit pas commencer par une erreur

Exemple :

```
string str4 = "xyz10";
int myint4 = stoi(str4);
```

```
terminate called after throwing an instance of
'std::invalid_argument'
 what(): stoi
Abort trap: 6
```



La base doit être compatible avec la chaîne à extraire

Exemple :

```
string str5 = "42";
int myint5 = stoi(str5, nullptr, 2);
```

```
terminate called after throwing an instance of
'std::invalid_argument'
 what(): stoi
Abort trap: 6
```

On peut convertir n'importe quel type « *intégral* » en string en utilisant un des fonctions `to_string` () de profil :

```
string to_string (int val);
string to_string (long val);
string to_string (long long val);
string to_string (unsigned val);
string to_string (unsigned long val);
string to_string (unsigned long long val);
string to_string (float val);
string to_string (double val);
string to_string (long double val);
```

```
int x;
cin >> x;
cout << "le chiffre contient " << to_string (x).size () << "
caractère(s)" << endl;
```

```
12345
le chiffre contient 5 caractère(s)
```

# RI.01 - INITIATION AU DÉVELOPPEMENT - AMPHI#09

A. Casali

# Z Test de samedi

- Porte principalement sur la partie algorithmique de la ressource RI.01
- Les parties :
  1. Corrections d'erreurs (20 min)
  2. Traduction d'algorithme (algo -> C++ : 20 min)
  3. Un premier retour sur la SAE SI.01
    1. Thèmes : les minis jeux :
      1. Pile ou face – équiprobable
      2. Pile ou face – pas équiprobable
      3. Pile ou face ou tranche – pas équiprobable
      4. Tic tac toe
      5. Master Pendu : un mix entre le master mind et le pendu.
  4. Plus de mini test après cette semaine

# PLAN

**A. Les structs**

B. Le type `string`

C. Le type `vector`

D. Flux sur un fichier de sortie

E. Flux sur un fichier d'entrée

F. Demo : copie d'un fichier et lien avec R1.02

# A Les struct

## A.1 Objectif

Créer son propre type permettant de regrouper différents types.

## A.2 Définition

```
struct newName {
 type1 varIdent1;
 type2 varIdent2;
 type3 varIdent3;
```

Attention au ‘;’

```
}; //newName
```

## A.3 Exemple

```
struct pos {
 unsigned abs;
 unsigned ord;
}; //pos
```

```
struct RGBcolor
{
 short Red;
 short Green;
 short Blue;
};
```

```
struct perso
{
 string name;
 string biblio;
 short age;
 unsigned XP;
 int lifePoint;
};
```



## A.4 Déclaration

```
nameOfTheStruct varIdentOfTheStruct;
```

```
pos aPos;
RGBColor color;
perso bibi;
```

## A.5 Accès à un élément

Notation pointée

```
varIdentOfTheStruct.varIdent
```

```
aPos.abs = 10;
aPos.ord = 10;
```

Autant d'accolades que de  
champs à remplir

## A.6 Déclaration et initialisation à la volée

```
nameOfTheStruct varIdentOfTheStruct {valueList};
```

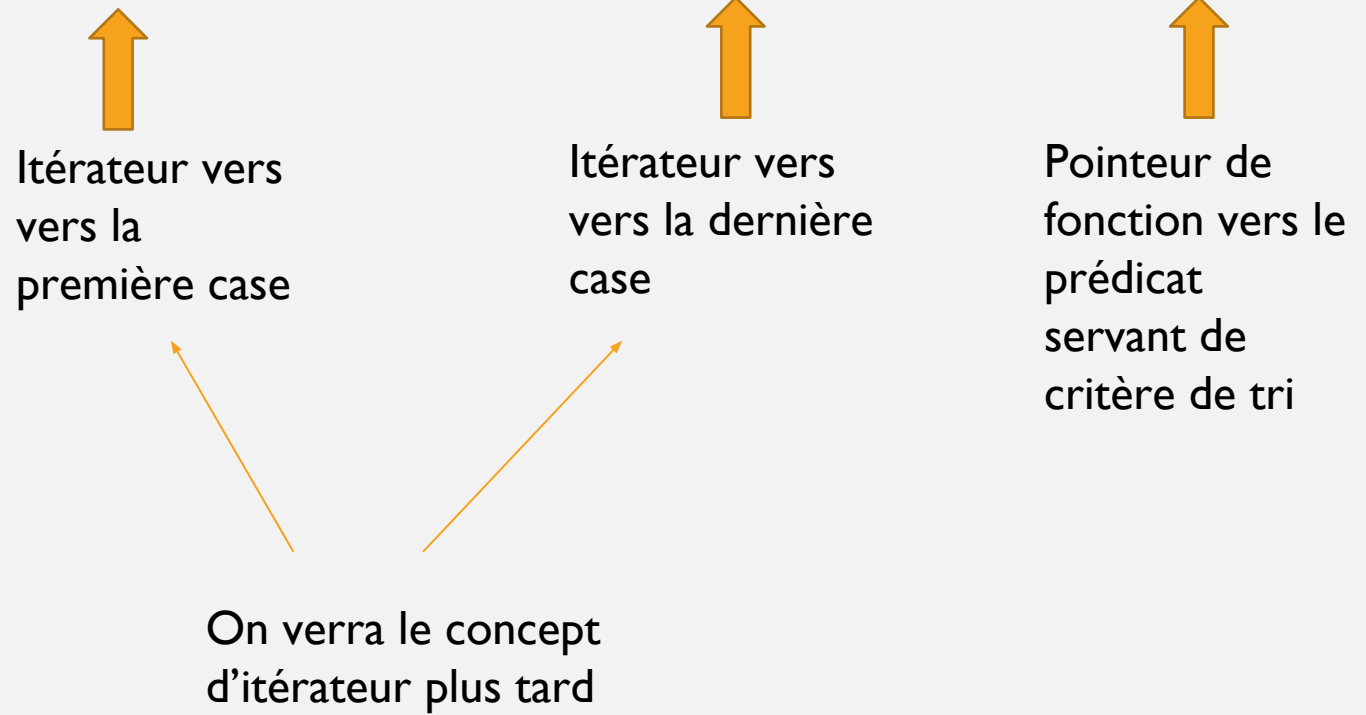
```
pos aPos {10, 10};
RGBColor color {0, 0, 0};
perso bibi {"casali", "", 44, -1, 100};
```

## A.7 Trier un vecteur de struct

```
vector<pos> vPos;
vector<RGBColor> vColor;
vector<perso> vPerso;
```

Comment fait on pour tirer ce tableau ?

```
sort (varIdentOfTheStruct.begin(), varIdentOfTheStruct.end(), criteria);
```



## A.7 Trier un vecteur de struct

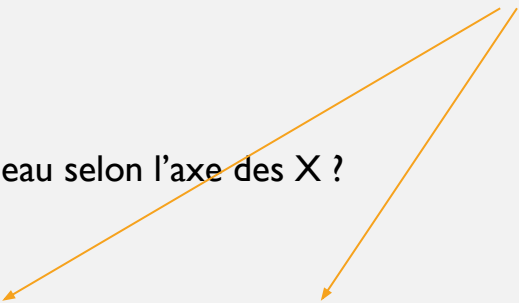
```
vector<pos> vPos;
```

Comment fait on pour tirer ce tableau selon l'axe des X ?

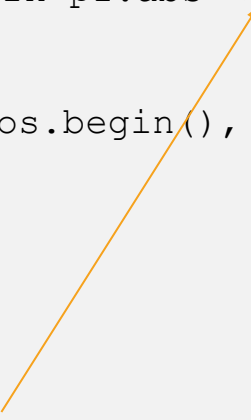
```
bool orderByX (const pos & p1, const pos & p2){
 return p1.abs <= p2.abs;
}
```

```
sort (vPos.begin(), vPos.end(), orderByX);
```


const / référence obligatoires



L'égalité est possible



Pas de parenthèse –  
uniquement le nom de la  
fonction



## A.7 Trier un vecteur de struct

```
vector<pos> vPos;
```

Comment fait on pour tirer ce tableau selon l'axe des Y ?

```
bool orderByX (const pos & p1, const pos & p2){
 return p1.abs <= p2.abs;
}
```

```
bool orderByY (const pos & p1, const pos & p2){
 return p1.ord <= p2.ord;
}
```

```
sort (vPos.begin(), vPos.end(), orderByY);
```

Même paramètres que précédemment

Seul le critère de tri change

## A.7 Trier un vecteur de struct – multi critères

```
vector<pos> vPerso;
```

**Comment fait on pour tirer ce tableau selon le nom ?**

```
bool orderByName (const perso & p1, const perso & p2){
 return p1.name <= p2.name;
}
```

```
sort (vPerso.begin(), vPerso.end(), orderByName);
```

**Comment fait on pour tirer ce tableau selon l'âge?**

```
bool orderByAge (const perso & p1, const perso & p2){
 return p1.age <= p2.age;
}
```

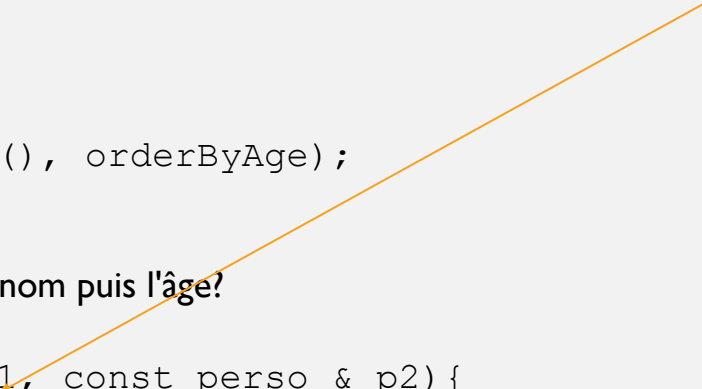
```
sort (vPerso.begin(), vPerso.end(), orderByAge);
```

**Comment fait on pour tirer ce tableau selon le nom puis l'âge?**

```
bool orderByAge (const perso & p1, const perso & p2){
 return p1.name <= p2.name && p1.age <= p2.age;
}
```

```
sort (vPerso.begin(), vPerso.end(), orderByAge);
```

**On enchaîne les conditions de tri**



## A.8 struct vs COO

Une struct est une classe pour laquelle tout (donnée membre / fonction public) est public

- ➡ on peut déclarer des fonctions dans une `struct`;
- ➡ à ne pas faire si vous ne maîtrisez pas !

# PLAN

**A. Les structs**

B. Le type string

C. Le type vector

D. Flux sur un fichier de sortie

E. Flux sur un fichier d'entrée

F. Demo : copie d'un fichier et lien avec R1.02

# B. Le type string

## B.1 Initialisation

- `string s1;`
- `string s2(10);`
- `string s2(10, 'c');`

Déclaration d'une chaîne vide (*ie.* de taille 0)

Déclaration d'une chaîne qui contient 10 fois le caractère `'c'`



# PLAN

**A. Les structs**

B. Le type `string`

C. Le type `vector`

D. Flux sur un fichier de sortie

E. Flux sur un fichier d'entrée

F. Demo : copie d'un fichier et lien avec R1.02

# C. Le type vector

## C.1 Initialisation

- `vector<int> v1;`
- `vector<int> v2(10);`
- `vector<int> v3(10, -1);`

Déclaration d'un tableau d'entier de taille 0

Déclaration d'un tableau d'entier de taille 10

Déclaration d'un tableau d'entier de taille 10 dont chaque case a pour valeur -1

- `vector<string> v4(10, string(100));`

Déclaration d'un tableau de `string` de taille 10 dont chaque case a pour valeur une `string` de taille 100

- `vector<string> v5(10, string(100, 'c'));`

Déclaration d'un tableau de `string` de taille 10 dont chaque case a pour valeur une `string` de taille 100 dont chaque case a pour valeur `'c'`

# C. Le type `vector`

## C.1 Initialisation

- `vector <pos> v6 (10, pos);`
- `vector <pos> v7 (10, pos (10,10));`
- `vector <pos> v8;`  
`for (size_t i (0); i < 10; ++i)`  
`v8.push_back (pos (i,10));`

Déclaration d'un tableau de `pos` de taille 10 dont chaque case a pour valeur une position (0,0)

Déclaration d'un tableau de `pos` de taille 10 dont chaque case a pour valeur une position (10, 10)

Déclaration d'un tableau de `pos` de taille 10 dont chaque case a pour valeur une position (i, 10)

# PLAN

- A. **Les structs**
- B. Le type `string`
- C. Le type `vector`
- D. Flux sur un fichier de sortie
- E. Flux sur un fichier d'entrée
- F. Demo : copie d'un fichier et lien avec R1.02

# D. Flux de sortie

On souhaite écrire le résultat de l'application dans un fichier de sortie

Solution 1 : rediriger la sortie standard vers un fichier

```
a.out > OutFile
```

Problèmes :

- Comment fait-on si on souhaite écrire dans un fichier depuis une interface graphique (ie, on n'a pas accès à la console);
- Comment peut on écrire dans plusieurs fichiers en même temps (on ne peut faire que 2 redirections : la sortie standard et le flux d'erreur).

Solution 2 : utiliser les flux de sortie

## D.1 Declaration

```
#include <fstream>
using namespace std;
```

```
ofstream ofs;
```



Output File STREAM

## D.2 Ouverture

```
ofs.open (const string & fileName [, Mode XXX]);
```

On lie le flux de sortie `ofs` au fichier `fileName` selon le mode spécifier.

La chaine `fileName` est le chemin relatif au fichier souhaité. Le « *point de départ* » du chemin relatif est l'exécutable.

Par défaut (aucun mode n'est spécifié) :

- Le fichier texte est accessible en écriture;
- Si le fichier n'existe pas, il est créé;
- S'il existe, les informations sont ajoutées avant celles qui existaient déjà.

### Exemple :

```
ofs.open ("test.txt");
```

## D.3 Déclaration et Ouverture

En même temps qu'on déclare un flux sur un fichier, il est possible de spécifier:

- le fichier en question;
- Son mode d'ouverture.

```
ofstream ofs (const string & filename [,Mode XXX]);
```

Exemple :

```
ofstream ofs ("test.txt");
```

## D.4 Les modes d'ouverture

Tous les modes sont des constantes de la classe `ios_base`.

On accède à un mode en utilisant la notation `ios_base::modeName`

| ModeName | Signification         | Accès                                                              |
|----------|-----------------------|--------------------------------------------------------------------|
| out      | output (sortie)       | Accès au flux en écriture                                          |
| in       | input (entrée)        | Accès au flux en lecture                                           |
| binary   | binary (binaire)      | Effectue les opérations d'E/S en mode binaire et non en mode texte |
| ate      | at end (à la fin)     | Position la tête (de lecture) à la fin du fichier                  |
| app      | append (ajout)        | Les ajouts se font à la fin du fichier et non au début             |
| trunc    | truncate (troncature) | Supprime le contenu du fichier                                     |

### Exemple :

```
ofstream ofs ("test.txt", ios_base::out|ios_base::binary);
```

Le caractère '|' permet d'enchaîner les options.





Un flux de sortie ne peut avoir, en même temps, les modes `trunc` et `app`!

## D.5 Fermeture d'un flux de sortie

On appelle la méthode `close ()` sur le flux de sortie.

Exemple :

```
ofs.close ();
```

## D.6 Ecriture dans un flux de sortie

On utilise l'injecteur (<<) pour pouvoir écrire dans un flux de sortie, de la même manière qu'on l'utilise pour pouvoir écrire dans la console (qui est un cas particulier des flux de sortie).

L'injecteur fonctionne :

- Pour tous les types (`int`, `char`, `bool`, ...);
- Pour certaines classes (`string`).

### Exemple :

```
ofs << 10 << endl;
ofs << true << endl;
ofs << 'a' << endl;
ofs << "coucou" << endl;
```

```
more test.txt
```

```
10
```

```
1
```

```
a
```

```
coucou
```



De la même façon qu'on ne peut afficher un vecteur (`vector`) sur la console, on ne peut pas l'injecter directement dans un flux de sortie.

### Exemple :

```
vector<int> v (10);
ofs << v;
```

```
XXX.cxx:YYY:Z: error: cannot bind 'std::basic_ostream<char>'
lvalue to 'std::basic_ostream<char>&&'
 ofs << v;
 ^
```

```
In file included from /opt/local/include/gcc48/c++/iostream:39:0,
 from hello.cpp:9:
/opt/local/include/gcc48/c++/ostream:602:5: error: initializing
argument 1 of 'std::basic_ostream<_CharT, _Traits>&
std::operator<<(std::basic_ostream<_CharT, _Traits>&&, const
_Tp&) [with _CharT = char; _Traits = std::char_traits<char>; _Tp
= std::vector<int>] '
 operator<<(basic_ostream<_CharT, _Traits>&& __os, const _Tp&
__x)
 ^
```

La seule et unique solution consiste à injecter chaque élément du vecteur les uns à la suite des autres en utilisant une boucle.

Exemple :

```
for (const int & x : v)
 ofs << setw (3) << x;
for (size_t i (0); i < v.size(); ++i)
 ofs << setw (3) << v[i];
```

more test.txt

0 0 0 0 0 0 0 0 0 0

# PLAN

- A. **Les structs**
- B. Le type `string`
- C. Le type `vector`
- D. Flux sur un fichier de sortie
- E. Flux sur un fichier d'entrée
- F. Demo : copie d'un fichier et lien avec R1.02

# E. Flux d'entrée

On souhaite lire des données pour une application depuis un fichier d'entrée

Solution 1 : rediriger l'entrée standard depuis un fichier

```
a.out < InFile
```

Problèmes :

- Comment fait-on si on souhaite lire des données depuis le Net (voir S3);
- Comment peut-on lire depuis plusieurs fichiers en même temps (on ne peut faire qu'une redirection : l'entrée standard).

Solution 2 : utiliser les flux d'entrée

## E.1 Declaration

```
#include <fstream>
using namespace std;
```

```
ifstream ifs;
```



Input File STREAM

## E.2 Ouverture

```
ifs.open (const string & fileName [, Mode XXX]);
```

On lie le flux d'entrée `ifs` au fichier `fileName` selon le mode spécifier.

La chaîne `fileName` est le chemin relatif au fichier souhaité. Le « *point de départ* » du chemin relatif est l'exécutable.

Par défaut (aucun mode n'est spécifié) :

- Le fichier texte est accessible en lecture;
- Si le fichier n'existe pas, il n'est pas créé (aucun contrôle quant à la validité du fichier est effectué);
- S'il existe, la tête de lecture est positionnée au début du fichier.

Exemple :

```
ifs.open ("test.txt");
```

## E.3 Déclaration et Ouverture

En même temps qu'on déclare un flux sur un fichier, il est possible de spécifier:

- le fichier en question;
- Son mode d'ouverture.

```
ifstream ifs (const string & filename [,Mode XXX]);
```

Exemple :

```
ifstream ifs ("test.txt");
```

## E.4 Les modes d'ouverture

Les différents mode d'ouverture sont les mêmes que pour les flux de sortie.

On ne peut toujours pas associer les mode `trunc` et `ate`.

De même l'ouverture échoue si le mode est `trunc`, mais pas `out`.

## E.5 Fermeture d'un flux d'entrée

On appelle la méthode `close ()` sur le flux d'entrée.

Exemple :

```
ifs.close ();
```



## E.6 Existence d'un flux d'entrée

On appelle la méthode `is_open ()` sur le flux d'entrée. Cette méthode renvoie vrai si le flux existe, faux sinon.

### Exemple :

```
if (!ifs.is_open ())
{
 cerr << "File not found" << endl;
 return;
}
//le fichier existe et est accessible
```

## E.7 Lecture depuis un flux d'entrée

On utilise l'extracteur (>>) pour pouvoir lire depuis un flux d'entrée, de la même manière qu'on l'utilise pour pouvoir lire depuis le clavier (qui est un cas particulier des flux d'entrée).

L'extracteur fonctionne :

- Pour tous les types (int, char, bool, ...);
- Pour certaines classes (string).

Exemple (I):

```
string str;
ifs >> str;
cout << str << endl;
int x;
ifs >> x;
cout << x << endl;
bool b;
ifs >> b;
cout << b << endl;
char c;
ifs >> c;
cout << c << endl;
;
```

test.txt :

```
coucou
10
1
a
```

```
coucou
10
```

```
1
```

```
a
```

### Exemple (2):

Tête de lecture

```
int x;
ifs >> x;
cout << x << endl;
bool b;
ifs >> b;
cout << b << endl;
char c;
ifs >> c;
cout << c << endl;
string str;
ifs >> str;
cout << str << endl;
```

0

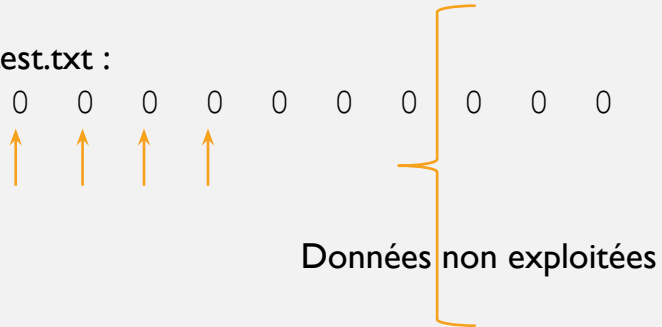
0

0

0

test.txt :

0 0 0 0 0 0 0 0 0 0



Données non exploitées

## E.7 Lecture du contenu d'un fichier

On aimerait avoir un algorithme du type :

```
tant_que (NON fin_de_fichier)
faire
 unType resultat <- lireQuelqueChose;
 manipuler (resultat);
ffaire
```

**PB : on ne peut détecter la fin d'un fichier qu'après avoir fait une tentative de lecture.**

Solution :

```
boucle
 unType resultat <- lireQuelqueChose;
 si (finDeFichier) sortie;
 manipuler (resultat);
fboucle
```

Pour détecter la fin d'un fichier, on appelle la méthode `eof ()` sur le flux d'entrée. Cette méthode renvoie vrai si la fin d'un fichier a été atteint, faux sinon.

**Exemple :**

```
while (true)
{
 aType varIdent = readSomething ();
 if (ifs.eof ()) break;
 manipulate (varIdent);
}
```

**Problème (I) :**

Lire un vecteur d'entier préalablement stocké

```
vector<int> V;
while (true)
{
 int x;
 ifs >> x;
 if (ifs.eof ()) break;
 V.push_back (x);
}
```

### Problème (2) :

Connaître le nombre de mots d'un fichier texte

Commande `wc -w` sous Linux

```
unsigned wc (0);
string str;
while (true)
{
 ifs >> str;
 if (ifs.eof ()) break;
 ++wc;
}
cout << setw (8) << wc << ' ' << __FILE__ << endl;
```



Macro du C permettant de connaître le nom du fichier courant.

### Problème (3) :

Connaitre le nombre de caractères d'un fichier texte

Commande `wc -c` sous Linux

### Solution I :

```
unsigned wc (0);
char c;
while (true)
{
 ifs >> c;
 if (ifs.eof ()) break;
 ++wc;
}
cout << setw (8) << wc << ' ' << __FILE__ << endl;
```

**Pb :** quid de l'encodage des caractères ☹

```
wc -c hello.cxx
```

```
wc: hello.cxx: Illegal byte sequence
```

```
8230 hello.cxx
```

```
./a.out
```

```
6414 hello.cxx
```

**Solution 2** : utiliser la méthode `get (char &)` de la classe `ifstream`.

```
unsigned wc = 0;
char c;
while (true)
{
 ifs.get (c);
 if (ifs.eof ()) break;
 ++wc;
}
cout << setw (8) << wc << ' ' << __FILE__ << endl;
```



#### Problème (4) :

Connaitre le nombre de lignes d'un fichier texte

Commande `wc -l` sous Linux

Lorsqu'on souhaite lire une ligne **complète** depuis la console :

```
~void getline (cin, string &);
```

Or on souhaite lire depuis un flux d'entrée. Donc :

```
~void getline (ifstream &, string &);
```



Le flux est passé par référence car il y a consommation de l'information (déplacement de la tête de lecture).

```
unsigned wc = 0;
string str;
while (true)
{
 getline (ifs, str);
 if (ifs.eof ()) break;
 ++wc;
}
cout << setw (8) << wc << ' ' << __FILE__ << endl;
```

En utilisant une boucle for :

```
unsigned wc = 0;
string str;
for (getline (ifs, str); !ifs.eof (); getline (ifs, str))
 ++wc;

cout << setw (8) << wc << ' ' << __FILE__ << endl;
```

# PLAN

**A. Les structs**

B. Le type `string`

C. Le type `vector`

D. Flux sur un fichier de sortie

E. Flux sur un fichier d'entrée

F. Demo : copie d'un fichier et lien avec R1.02

# RI.01 - INITIATION AU DÉVELOPPEMENT - AMPHI#10

A. Casali

# PLAN

A. Gestion des erreurs de lecture

B. Positionnement dans un flux

C. Etat d'un flux

D. Tri de tableau


E. Tester ses programmes (V2)

# A. Gestion des erreurs de lecture

## A.I Echec de lecture

Ca aurait pu être n'importe quel `ifstream`, le résultat aurait été le même

```
int i;
for (; ;)
{
 cout << "Taper un entier suivi de <CR> : ";
 cin >> i;
 if (cin.eof ()) break;
 cout << "i = " << i << endl;
}
```

Taper un entier suivi de <CR> : x10 

i = 1075991360

Taper un entier suivi de <CR> : i = 1075991360

Taper un entier suivi de <CR> : i = 1075991360

Taper un entier suivi de <CR> : i = 1075991360

...



1. L'utilisateur n'a pas entré un `int`;
2. La chaîne de caractère "`x10`" reste dans le tampon;
3. Le tampon n'a pas pu se vider;
4. A chaque tour de boucle, l'extracteur essaye de convertir ce qu'il y a dans le tampon en un `int` (retour en 2)


### Solution :

Appeler la méthode `fail ()` de la classe `istream` pour savoir si l'extraction a pu se faire.

```
int i;
for (;;) {
 cout << "Taper un entier suivi de <CR> : ";
 cin >> i;
 if (cin.eof ()) break;
 if (cin.fail ())
 cout << "echec de lecture" << endl;
 else
 cout << "i = " << i << endl;
}
```

Classe pour tous les flux d'entrée (`cin`,  
`ifstream`, ...)



Taper un entier suivi de <CR> : x10   
echec de lecture

Taper un entier suivi de <CR> : echec de lecture

Taper un entier suivi de <CR> : echec de lecture

Taper un entier suivi de <CR> : echec de lecture

Taper un entier suivi de <CR> : echec de lecture



1. On a bien détecté l'erreur sur le flux;
2. Mais, le caractère invalide n'étant pas lu, il reste dans le **tampon** et est retrouvé à la lecture suivante ⇒  
**purger le tampon**

...

```
if (cin.fail ())
{
 cout << "echec de lecture" << endl;
 string Buf;
 getline (cin, Buf);
}
```

même exécution ⇒


**même résultat !!!**




## A.2 Restauration

Pour pouvoir restaurer un flux positionné dans un état `fail` (), il faut appeler la méthode `clear` () sur ce flux.

```
...
if (cin.fail ())
{
 cout << "echec de lecture" << endl;
 cin.clear (); = remise en état du flux
 string Buf;
 getline (cin, Buf);
}
```

Taper un entier suivi de <CR> : x10 

echec de lecture

Taper un entier suivi de <CR> : 10 

i = 10

Taper un entier suivi de <CR> : Ctrl+D

**Problème :** tous les entiers sur la ligne après le caractère invalide sont perdus.

**Solution :** purger **caractère/caractère**

### A.3 Lecture dans le tampon sans consommation de l'information

On souhaite supprimer tous les caractères invalides du tampon qui sont avant l'entier à extraire.

```
int i;
for (;;) {
 cout << "Taper un entier suivi de <CR> : ";
 cin >> i;
 if (cin.eof ()) break;
 if (cin.fail ())
 {
 cout << "echec de lecture" << endl;
 char c;
 cin.clear ();
 for (cin.get (c); !isdigit (c); cin.get (c));
 }
 else
 cout << "i = " << i << endl
}
```

Taper un entier suivi de <CR> : xyz10

echec de lecture

Taper un entier suivi de <CR> : i = 0

`get ()` consomme l'information qui est dans le tampon




### Solution :

Ne consommer l'information que si elle est invalide.

⇒ On est obligé de lire dans le tampon sans le toucher. Pour cela, il faut utiliser la méthode `peek ()`.

...

```
if (cin.fail ())
{
 cout << "echec de lecture" << endl;
 cin.clear ();
 char c;
 for (c = cin.peek (); !isdigit (c); c = cin.peek ())
 cin.get ();
}
```

Taper un entier suivi de <CR> : xyz10 

echec de lecture

Taper un entier suivi de <CR> : i = 10

Taper un entier suivi de <CR> : Ctrl + D

On souhaite supprimer tous les caractères invalides du tampon qui sont après la `string` à extraire.

```
string email;
cout << "Entrer votre email : ";
cin >> email;
cout << email << endl;
unsigned age;
cout << "Entrer votre age : ";
cin >> age;
cout << age << endl;
```

```
Entrer votre email : alain.casali@univ-amu.fr
marc.laporte@univ-amu.fr
alain.casali@univ-amu.fr
Entrer votre age : 0
```

=> Il faut purger le reste du tampon après avoir fait la lecture.

Solution 1 : utiliser (et adapter) la méthode qui utilise `peek` ()

Solution 2 : utiliser la méthode `ignore` () de la classe `istream` de profil

```
~void ignore (streamsize n, int delim);
```

`~unsigned` (Par défaut 1)

Par défaut EOF

1<sup>er</sup> paramètre : nombre de caractères maximum à ignorer dans le flux courant. Ces caractères seront supprimés du tampon.

2<sup>ème</sup> paramètre : ignorer les caractères jusqu'à ce qu'on trouve ce caractère.

```
string email;
cout << "Entrer votre email : ";
cin >> email;
cout << email << endl;
cin.ignore (numeric_limits<streamsize>::max(), '\n');
unsigned age;
cout << "Entrer votre age : ";
cin >> age;
cout << age << endl;
```

# PLAN

- A. Gestion des erreurs de lecture
- B. Positionnement dans un flux
- C. Etat d'un flux
- D. Tri de tableau
- E. Tester ses programmes (V2)

# B. Positionnement dans un flux

On appelle la méthode `seekg ()` de la classe `istream` de profil  
`~void seekg (streamoff, ios_base::seekdir) ;`

1<sup>er</sup> paramètre : (`~int`) nombre d'octet du déplacement

2<sup>ème</sup> paramètre : position à partir de laquelle on fait le déplacement

- `ios_base::beg` : déplacement à partir du début du flux;
- `ios_base::cur` : déplacement à partir de la position courante du flux;
- `ios_base::end` : déplacement à partir de la fin du flux.

## Lecture à l'envers dans un flux

```
ifstream ios ("Relecture", ios_base::ate |
ios_base::in);
if (0 == ios.tellg()) return 0;

ios.seekg (-1, ios_base::cur);

for (char c; ;)
{
 cout << ios.tellg();
 ios.get (c) ;
 cout << ' ' << c << '\n';

 if (1 == ios.tellg()) break;
 ios.seekg (-2, ios_base::cur);
}
return 0;
```

*at end*

|   |   |
|---|---|
| 5 | 5 |
| 4 | 4 |
| 3 | 3 |
| 2 | 2 |
| 1 | 1 |
| 0 | 0 |



# PLAN

- A. Gestion des erreurs de lecture
- B. Positionnement dans un flux
- C. Etat d'un flux
- D. Tri de tableau
- E. Tester ses programmes (V2)

# C. Etat d'un flux

Un flux peut avoir 4 états :

- `eof` () : fin de fichier;
- `fail` () : erreur E/S
- `bad` () : erreur sur le flux (propagation d'exception, ...)
- `good` () : le flux est OK

Les 2 tests suivants sont équivalents :

1. `if (ifs)`
2. `if (ifs.good () [==true])`

Profil de `getline` () (amphi 2.2)

```
~void getline (ifstream &, string &);
```

Nouveau profil:

```
ifstream getline (ifstream &, string &);
```


⇒ Lecture jusqu'à la fin d'un fichier :

Amphi 9 :

```
string str;
for (getline (ifs, str); !ifs.eof (); getline (ifs, str))
 Manipulate (str);
```

En utilisant le nouveau profil de getline () :

```
string str;
for (; getline (ifs, str);)
 Manipulate (str);
```



Appel sous-jacent à `.good ()` et on n'effectue l'appel à la fonction uniquement si le flux est dans un état valide.

# PLAN

- A. Gestion des erreurs de lecture
- B. Positionnement dans un flux
- C. Etat d'un flux
- D. Tri de tableau
- E. Tester ses programmes (V2)

# D. Tri de tableau

## D.1 Tri par sélection

A la  $i^{\text{ème}}$  itération, on cherche le plus petit élément du tableau ( $V$ ) entre les positions  $[i, V.size() [$  et on le permute avec l'élément en position  $i$ .

|   |   |   |    |    |    |    |   |
|---|---|---|----|----|----|----|---|
| 5 | 1 | 2 | 12 | 24 | 13 | 10 | 2 |
|---|---|---|----|----|----|----|---|

|   |   |   |    |    |    |    |   |
|---|---|---|----|----|----|----|---|
| 1 | 5 | 2 | 12 | 24 | 13 | 10 | 2 |
|---|---|---|----|----|----|----|---|

|   |   |   |    |    |    |    |   |
|---|---|---|----|----|----|----|---|
| 1 | 2 | 2 | 12 | 24 | 13 | 10 | 5 |
|---|---|---|----|----|----|----|---|



Complexité :  $O(n^2)$

## D.2 Tri par insertion

Chaque élément est placé à sa position finale dans le vecteur : on cherche à placer le  $i^{\text{ème}}$  élément à sa place dans le sous vecteur  $[0, i[$

|   |   |   |    |    |    |    |   |
|---|---|---|----|----|----|----|---|
| 5 | 1 | 2 | 12 | 24 | 13 | 10 | 2 |
|---|---|---|----|----|----|----|---|

|   |   |   |    |    |    |    |   |
|---|---|---|----|----|----|----|---|
| 1 | 5 | 2 | 12 | 24 | 13 | 10 | 2 |
|---|---|---|----|----|----|----|---|

|   |   |   |    |    |    |    |   |
|---|---|---|----|----|----|----|---|
| 1 | 2 | 5 | 12 | 24 | 13 | 10 | 2 |
|---|---|---|----|----|----|----|---|



Complexité :  $O(n^2)$

### D.3 Tri à bulles

Si l'élément d'indice  $i$  est plus grand que celui d'indice  $i + 1$ , on les permute

|   |   |   |    |    |    |    |   |
|---|---|---|----|----|----|----|---|
| 5 | 1 | 2 | 12 | 24 | 13 | 10 | 2 |
|---|---|---|----|----|----|----|---|

|   |   |   |    |    |    |   |    |
|---|---|---|----|----|----|---|----|
| 1 | 2 | 5 | 12 | 13 | 10 | 2 | 24 |
|---|---|---|----|----|----|---|----|

|   |   |   |    |    |   |    |    |
|---|---|---|----|----|---|----|----|
| 1 | 2 | 5 | 12 | 10 | 2 | 13 | 24 |
|---|---|---|----|----|---|----|----|



Optimisation : faire une passe sur indice croissant et une passe sur indice décroissant dans chaque boucle.



Complexité :  $O(n^2)$

## D.4 Tri par comptage

On crée un tableau externe de taille  $\text{max} - \text{min}$  dans lequel on compte chaque occurrence de chaque nombre.

tabInit

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 6 | 1 | 2 | 2 | 4 | 3 | 1 | 2 |
|---|---|---|---|---|---|---|---|

tabExterne

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 2 | 3 | 1 | 1 | 0 | 1 |
|---|---|---|---|---|---|

L'élément  $\text{min} + 1$  apparaît 3 fois

On reconstitue le tableau initial

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 2 | 2 | 3 | 4 | 6 |
|---|---|---|---|---|---|---|---|



Complexité :  $O(n)$



# PLAN

- A. Gestion des erreurs de lecture
- B. Positionnement dans un flux
- C. Etat d'un flux
- D. Tri de tableau
- E. Tester ses programmes (V2)

# E. Tester ses programmes (V2)

## E.1 Test interne



La première qualité d'un programme est sa justesse !  
Il doit répondre aux spécifications demandées.

Fonction à tester pour savoir si un vecteur est trié :

```
void estVecteurTrie (const vector<int> & V)
{
 for (size_t i (0); i < V.Size () -1 ; ++i)
 assert (V[i] <= V[i + 1]);
}
```

## E.2 Test de performance



On ne commence les tests de performance qu'à partir du moment où le programme est juste!

Idée :

```
Temps1 <- DemanderHeure ();
appeler la fonction;
Temps2 <- DemanderHeure ();
CalculerDifference (Temps2, Temps1);
```

Sémantiquement juste, mais complètement faux dans la pratique : si on exécute plusieurs fois le programme de test, on n'obtiendra jamais le même résultat (l'OS n'étant pas dans le même état) => il faut faire N (suffisamment grand selon le temps disponible) tests et faire la moyenne des N tests.

Stocke dans timer1 l'horodatage courant.

Nécessite une adresse mémoire => passage par référence dans l'appel.

```
#include <ctime>
```

```
void Test (const unsigned & nbTests)
{
 vector<time_t> vTime (NbTests);
 //functionToTest () params initialization
 for (unsigned (i (0); i < NbTests; ++i) {
 time_t timer1, timer2;
 time (&timer1);
 fonctionToTest (ListOfParams);
 time (&timer2);
 //manipulate (difftime (timer2, timer1));
 vTime [i] = difftime (timer2, timer1);
 }
 manipulate (makeAvergae (vTime));
}
```

makeMedian () serait plus parlant!

Renvoie la différence (en secondes) entre timer2 et timer1.

Ou mieux si nbTest est suffisamment grand (à votre guise)

```
#include <ctime>
```

```
void Test (const unsigned & NbTests, size_t & NbToRemove)
{
 vector<time_t> vTime(nbTests);
 for (unsigned (i (0); i < NbTests; ++i) {
 time_t timer1, timer2;
 time (&timer1);
 functionToTest (listOfParams);
 time (&timer2);
 vTime [i] = difftime (timer2, timer1);
 }
 for (size_t i (0); i < NbToRemove / 2; ++i) {
 vTime.erase (max_element (VTime.begin (), VTime.end ()));
 vTime.erase (min_element (VTime.begin (), VTime.end ()));
 }
 manipulate (makeAvergae (VTime));
}
```

Est-ce suffisant?



NON : on n'a pas fait varier les paramètres de la fonction à tester  
!!

Exemple:

On souhaite étudier l'impacte du passage par référence d'un vecteur dans une fonction d'affichage.

```
void showVectV1 (const vect & V) ;
void showVectV2 (const vect V) ;
```

| NbElem        | NbIter  | TempsMoyenV1 | TempsMoyenV2 |
|---------------|---------|--------------|--------------|
| 1 000         | 100 000 | 0            | 0            |
| 10 000        | 100 000 | 0            | 0            |
| 100 000       | 100 000 | 0            | 0            |
| 10 000 000    | 100 000 | 2            | 3            |
| 1 000 000 000 | 100 000 | 209          | 245          |