

## *Chapitre 3*

# ENTRÉES ET SORTIES

POLYTECHNIQUE  
MONTRÉAL

LE GÉNIE  
EN PREMIÈRE CLASSE

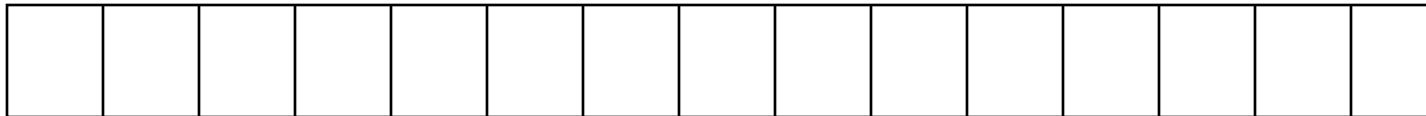


## Fonctions d'entrée/sortie

- 📖 Les entrées/sorties s'appuient sur des flots (stream).
- 📖 Un flot est un flux de données abstrait qui part d'une source et qui va vers une cible.
- 📖 Source et cible peuvent être des fichiers usuels, des périphériques ou des emplacements mémoires.
- 📖 L'objet *cout* représente le canal de sortie standard.
- 📖 << est l'opérateur d'insertion pour envoyer des données vers un flot. Ex.: `cout << x;`
- 📖 L'objet *cin* représente le canal d'entrée standard.
- 📖 >> est l'opérateur d'extraction qui lit les données d'un flot. Ex.: `cin >> x;`

# Les Entrées et Sorties

- Nous appelons, tampon d'entrée, une portion de l'espace mémoire que nous représenterons:



- Chaque case représente un octet pouvant contenir un caractère.
- ↵ représente un saut de ligne (carriage return + line feed) illustré à l'aide de deux cases dans le livre.

## Lecture avec l'opérateur >>

- ❑ Les séparateurs délimitant les valeurs à lire sont: l'espace, le tabulateur, la fin de ligne et la fin de fichier.
- ❑ Les séparateurs précédant la valeur à lire sont ignorés (ou sautés).
- ❑ La lecture s'effectue jusqu'à la rencontre du premier séparateur suivant la valeur à lire, ou du premier caractère invalide pour le type de valeur.

## Déclaration des variables pour les exemples suivants

```
double  nbreReel;  
int     entier1, entier2;  
char    car1, car2;  
string  chaine1, chaine2, chaine3;  
int     numero;  
string  rue;
```

tous les exemples de lecture suivants sont dans `III_lecture.cpp`

# Lecture de valeurs numériques

Contenu du tampon d'entrée initial

		2	.	3	4		5	7				-	8	1	↵
--	--	---	---	---	---	--	---	---	--	--	--	---	---	---	---



```
cin >> nbreReel;
```

		2	.	3	4		5	7				-	8	1	↵
--	--	---	---	---	---	--	---	---	--	--	--	---	---	---	---



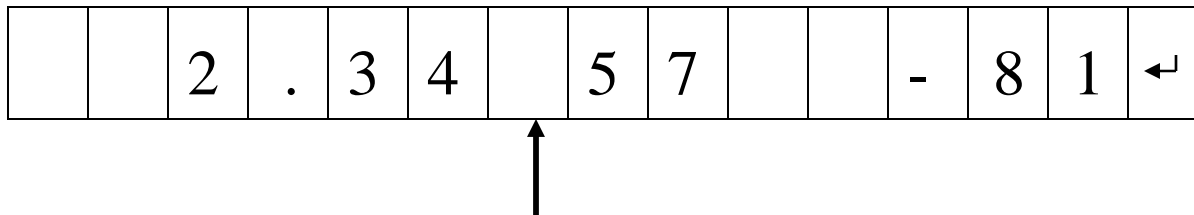
nbreReel

?

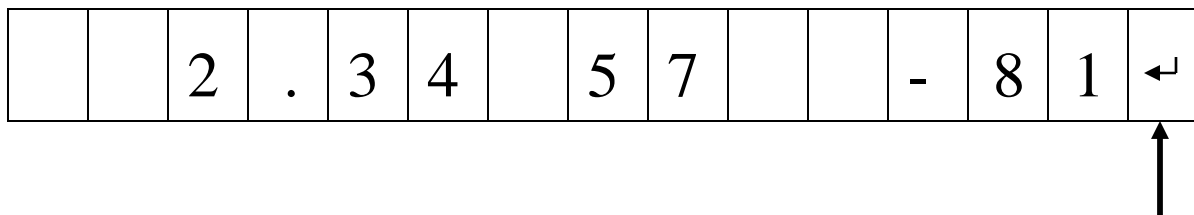
nbreReel

2.34

# Lecture de valeurs numériques



```
cin >> entier1 >> entier2;
```



entier1 entier2

?

?

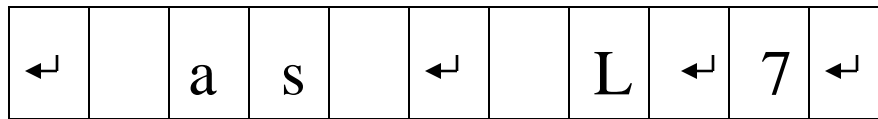
entier1 entier2

57

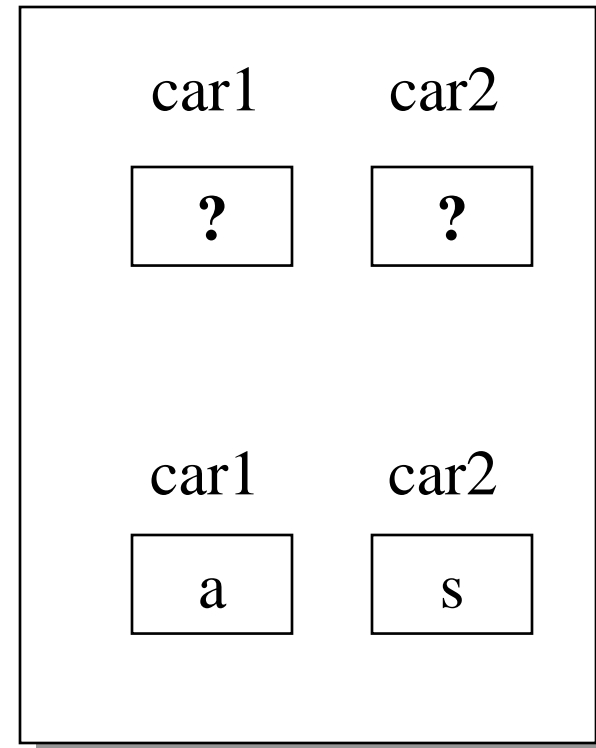
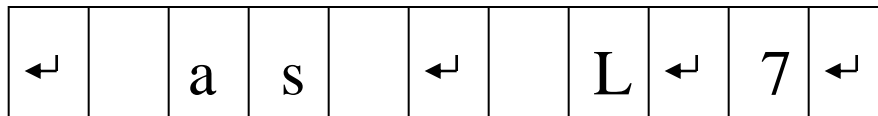
-81

# Lecture de caractères

Contenu du tampon d'entrée initial

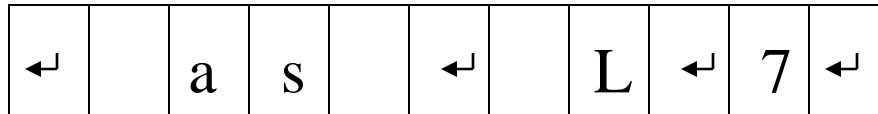


```
cin >> car1 >> car2;
```

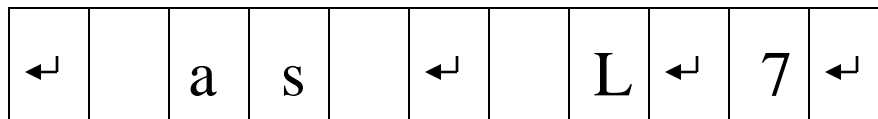




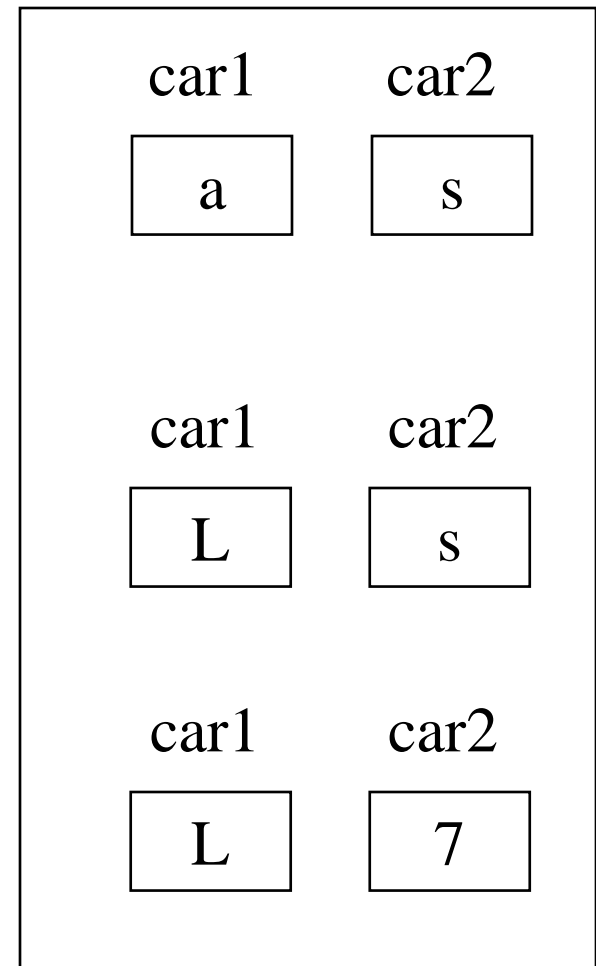
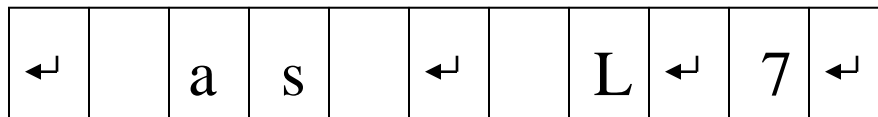
# Lecture de caractères



```
cin >> car1;
```



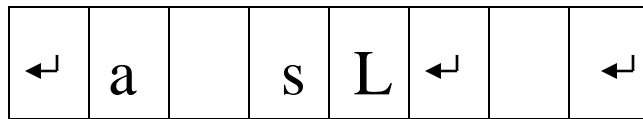
```
cin >> car2;
```



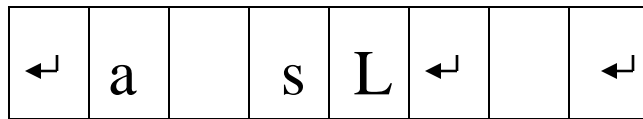
## Lecture avec `istream& get(char& carLu)`

- La fonction lit un caractère depuis un flot et le range dans la variable `carLu`.
- Permet de lire tous les caractères standards présents dans le tampon.
- Le caractère mémorisé lors de la lecture d'un ENTER est le caractère line feed (#10).

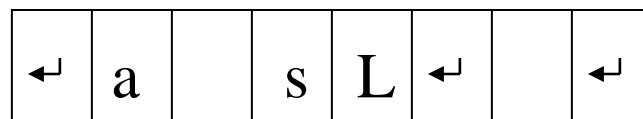
# Lecture de caractères



`cin.get(car1);` ou `car1 = cin.get();`



`cin.get(car2);` ou `car2 = cin.get();`



car1

car2

?

?

car1

car2

←

?

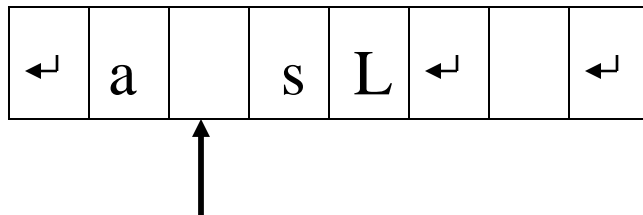
car1

car2

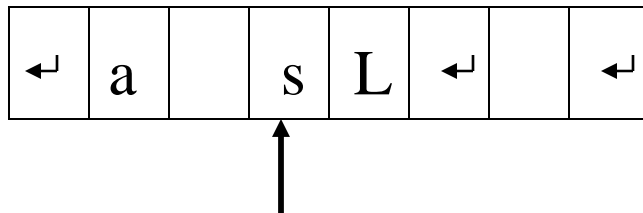
←

a

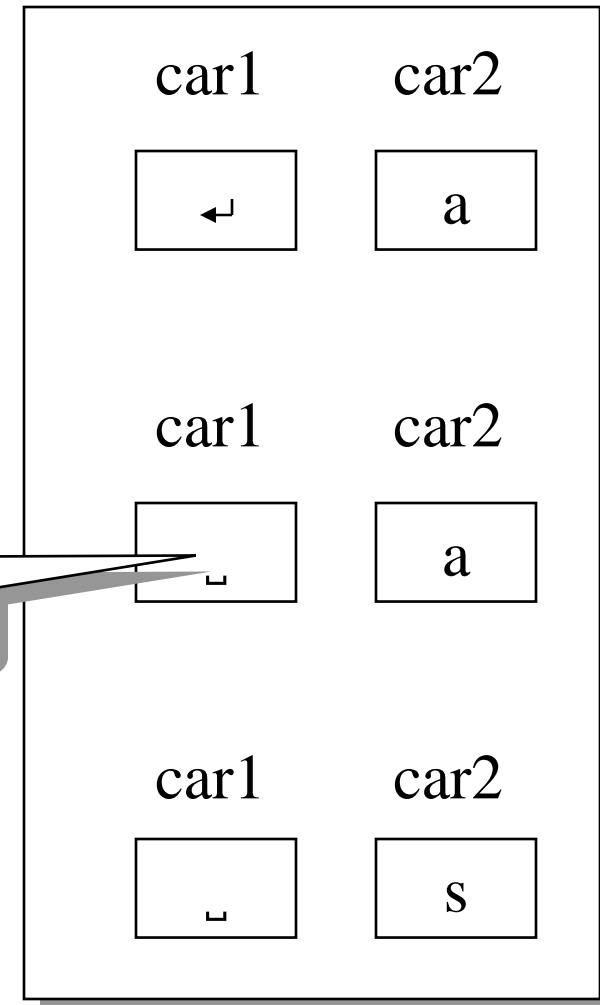
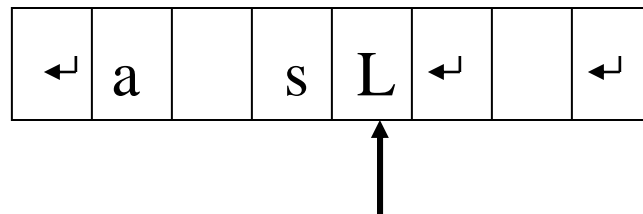
# Lecture de caractères



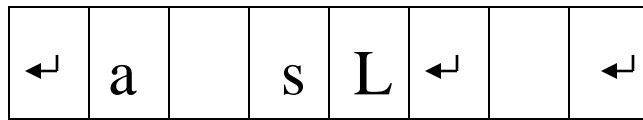
`cin.get(car1);` ou `car1 = cin.get();`



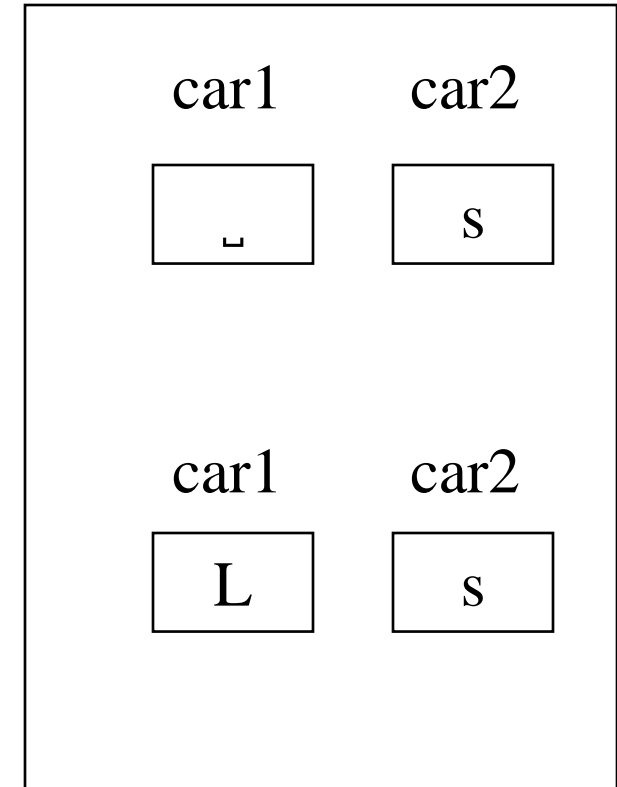
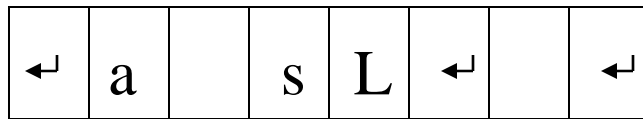
`cin.get(car2);` ou `car2 = cin.get();`



# Lecture de caractères



`cin.get(car1);` ou `car1 = cin.get();`



# Lecture de chaînes de caractères

Contenu du tampon d'entrée initial

F	e	n	ê	t	r	e				o	u	v	e	r	t	e	↵
---	---	---	---	---	---	---	--	--	--	---	---	---	---	---	---	---	---



```
cin >> chaine1;
```

F	e	n	ê	t	r	e				o	u	v	e	r	t	e	↵
---	---	---	---	---	---	---	--	--	--	---	---	---	---	---	---	---	---



chaine1

<sup>N<sub>U</sub></sup> <sub>L</sub>
---------------------------------------

chaine1

Fenêtr  <sup>N<sub>U</sub></sup> <sub>L</sub>
---

# Lecture de chaînes de caractères

F	e	n	ê	t	r	e			o	u	v	e	r	t	e	↵
---	---	---	---	---	---	---	--	--	---	---	---	---	---	---	---	---



```
cin >> chaine1;
```

F	e	n	ê	t	r	e			o	u	v	e	r	t	e	↵
---	---	---	---	---	---	---	--	--	---	---	---	---	---	---	---	---



chaine1

Fenêtr|<sup>N<sub>U</sub><sub>L</sub></sup>

chaine1

ouverte|<sup>N<sub>U</sub><sub>L</sub></sup>

Lecture avec  
`istream& getline(istream& buf, string& chaineLue,  
char fin='\n')`

- Lit une suite de caractères dans un flot jusqu'à:
  - Le caractère `fin` a été lu (ENTER par défaut)
  - EOF a été lu
- Les caractères lus sont mémorisés dans la variable `chaineLue`.
- La marque de fin (`fin`) a été lue, mais n'est pas placée dans la `chaineLue`.



# Lecture de chaînes de caractères

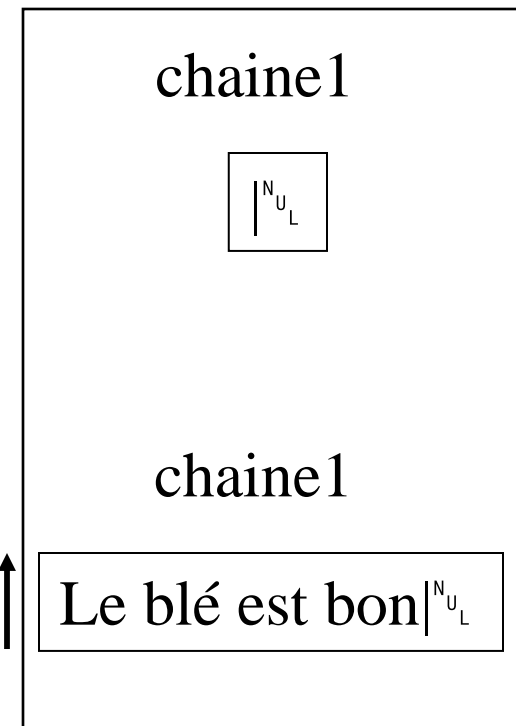
Contenu du tampon d'entrée initial

L	e		b	l	é		e	s	t		b	o	n	↵
---	---	--	---	---	---	--	---	---	---	--	---	---	---	---



```
getline(cin, chaine1);
```

L	e		b	l	é		e	s	t		b	o	n	↵
---	---	--	---	---	---	--	---	---	---	--	---	---	---	---



# Lecture de chaînes de caractères

L	e	↵	b	l	é		↵	e	s	t	↵			
---	---	---	---	---	---	--	---	---	---	---	---	--	--	--

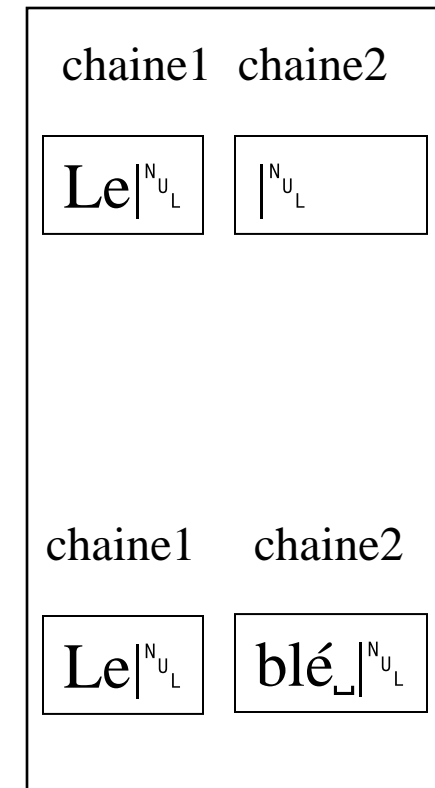


```
getline(cin, chaine1);
```

L	e	↵	b	l	é		↵	e	s	t	↵			
---	---	---	---	---	---	--	---	---	---	---	---	--	--	--



```
getline(cin, chaine2);
```



```
istream& ignore(int n = 1, int fin = EOF);
```

- Saute une suite de caractères du flot d'entrée jusqu'à:
  - n caractères ont été sautés
  - la marque de fin est rencontrée; lorsqu'elle est rencontrée la marque (le caractère) est sauté également.
- Peut être utilisé pour sauter le <ENTER> présent dans le tampon afin d'effectuer correctement la lecture avec la fonction `getline()` par la suite.

# Lecture de chaînes de caractères

 Lecture d'une chaîne vide !

```
cin >> numero;
```

3	5	3	7	↵	G	i	l	f	o	r	d	↵
---	---	---	---	---	---	---	---	---	---	---	---	---



```
getline(cin, rue);
```

3	5	3	7	↵	G	i	l	f	o	r	d	↵
---	---	---	---	---	---	---	---	---	---	---	---	---




numero

3537

rue

|<sup>N</sup><sub>U</sub><sub>L</sub>

# Lecture

 Solution: utilisation de la fonction `ignore(nombre)` pour « sauter » le caractère `↵`.

3	5	3	7	↵	L	a		b	i	s	e	↵
---	---	---	---	---	---	---	--	---	---	---	---	---



```
cin >> numero;
```

3	5	3	7	↵	L	a		b	i	s	e	↵
---	---	---	---	---	---	---	--	---	---	---	---	---



```
cin.ignore(1);
```

3	5	3	7	↵	L	a		b	i	s	e	↵
---	---	---	---	---	---	---	--	---	---	---	---	---



numero

3537

rue

|<sup>N</sup><sub>U</sub><sub>L</sub>

# Lecture

3	5	3	7	↵	L	a		b	i	s	e	↵
---	---	---	---	---	---	---	--	---	---	---	---	---



```
getline(cin, rue);
```

3	5	3	7	↵	L	a		b	i	s	e	↵
---	---	---	---	---	---	---	--	---	---	---	---	---



numero

3537

rue

La\_bise|<sup>N<sub>U</sub><sub>L</sub></sup>

# Affichage

```
int entier = 123;
double reel = 45.6;
char caractere = 'X';
string chaine = "bonjour";

cout << entier;
cout << reel << caractere;
cout << chaine;
// Malgré qu'il y ait plusieurs lignes d'instructions dans le programme,
// les affichages précédents seront sans espacement.

cout << endl; // endl indique une fin de ligne (end of line).
cout << entier << endl;
cout << reel << " " << caractere << endl;
cout << chaine << endl;

// Résultat:
// 12345.6Xbonjour
// 123
// 45.6 X
// bonjour
```

III\_affichage.cpp

# Traitement d'erreur

```
void clear(int n = 0);
```

- Cette fonction est utilisée lorsqu'une erreur de lecture est commise puisque toute instruction de lecture subséquente à une erreur de lecture est ignorée pour le flot d'entrée concerné.
- On réactive le flot d'entrée standard cin à l'aide de l'instruction:

```
cin.clear();
```

- Si on ne veut pas traiter les erreurs mais s'assurer que le programme arrête lors d'une erreur, ajouter simplement au début du main:

```
cin.exceptions(ios::failbit);
```

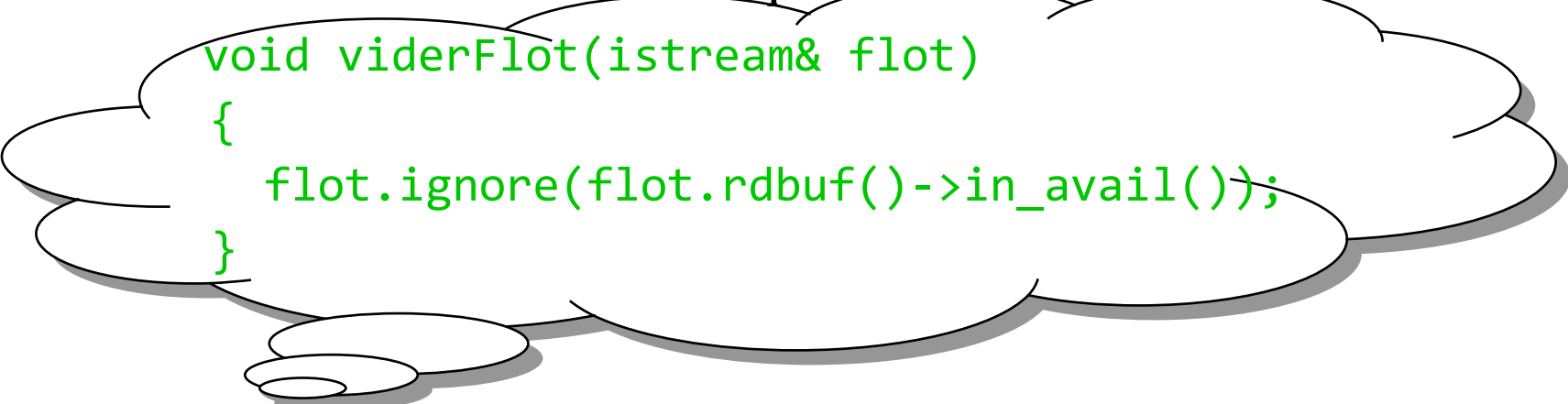


# Traitement d'erreur

- Il est nécessaire de nettoyer le tampon d'entrée lorsqu'une erreur de lecture survient. L'instruction suivante permet d'enlever les 80 premiers caractères du flot d'entrée cin ou jusqu'à la rencontre d'une fin de ligne:

```
cin.ignore(80, '\n');
```

- Enlever tous les caractères présents dans un flot d'entrée:



```
void viderFlot(istream& flot)
{
    flot.ignore(flot.rdbuf()->in_avail());
}
```

La notion de fonction sera abordée au chapitre 5; après avoir entré cette fonction, écrire « `viderFlot(cin);` » est équivalent à écrire « `cin.ignore(cin.rdbuf()->in_avail());` »

## Les fichiers textes

- ✓ Les fichiers textes sont des fichiers séquentiels.
- ✓ Les opérations de lecture et d'écriture sont exclusives, on peut lire OU écrire et non les deux à la fois.
- ✓ Analogie avec les rubans audio ou vidéo.
- ✓ Pour utiliser un fichier texte on inclut préalablement le fichier d'en-tête *fstream*.
- ✓ Les opérateurs et fonctions de lecture ou écriture (clavier ou écran) s'appliquent intégralement à un fichier texte.

	LECTURE	ÉCRITURE
Déclaration	<code>ifstream ficLire;</code>	<code>ofstream ficEcrire;</code>
<b>Déclaration + ouverture</b>	<code>ifstream ficLire("Nom");</code> <code>ifstream ficLire("Nom", mode);</code>	<code>ofstream ficEcrire("Nom");</code> <code>ofstream ficEcrire("Nom", mode);</code>
Ouverture	<code>ficLire.open("Nom");</code> <code>ficLire.open("Nom", mode);</code>	<code>ficEcrire.open("Nom");</code> <code>ficEcrire.open("Nom", mode);</code>
Vérification	<code>if (ficLire.fail())</code>	<code>if (ficEcrire.fail())</code>
Lecture	<code>ficLire &gt;&gt; variable;</code> <code>getline(ficLire, mot); etc.</code>	<i>Impossible</i>
Écriture	<i>Impossible</i>	<code>ficEcrire &lt;&lt; variable;</code> <code>ficEcrire.put(car); etc.</code>
Fin de fichier	<code>if (ficLire.eof())</code>	<code>if (ficEcrire.eof())</code>
Fermeture	<code>ficLire.close();</code>	<code>ficEcrire.close();</code>

# Mode d'ouverture

MODE	SIGNIFICATION
<code>ios::in</code>	- Ouvrir un fichier texte en mode lecture seulement, le fichier doit exister (option par défaut des fichiers de type <i>ifstream</i> ).
<code>ios::out</code>	- Ouvrir un fichier texte en mode écriture, le contenu du fichier est détruit si le fichier existe ou le fichier est créé s'il n'existe pas (option par défaut des fichiers de type <i>ofstream</i> ).
<code>ios::app</code>	- ( <b>append</b> ) Ouvrir un fichier texte pour écriture uniquement à partir de la fin du fichier. Tous les ajouts sont faits à la fin du fichier.
<code>ios::ate</code>	- ( <b>at end</b> ) Ouvrir un fichier texte et se positionner à la fin du fichier pour écriture. Ensuite, l'écriture se fait selon la position courante.
<code>ios::trunc</code>	- ( <b>truncate</b> ) Ouvrir un fichier texte et effacer son contenu si le fichier existe (option par défaut si <i>ios::out</i> est spécifié et aucune des options <i>ios::app</i> , <i>ios::ate</i> ou <i>ios::in</i> ne l'est).

# Fichier Texte

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

int main()
{
    ifstream entree("III_original.txt"); // Ouverture du fichier d'entrée à lire.
    ofstream sortie("III_copie.txt");    // Ouverture du fichier de sortie à créer.

    string mot;
    entree >> mot;                      // Lecture d'un mot de entree.
    sortie << mot << endl;              // Écriture du mot lu dans sortie.
    int nombre;
    entree >> nombre;                   // Lecture d'un nombre de entree.
    sortie << nombre << endl;          // Écriture du nombre lu dans sortie.
    string phrase;
    getline(entree, phrase);            // Lecture d'une phrase de entree.
    sortie << phrase << endl;          // Écriture de la phrase lue dans sortie.

    entree.close();                    // Fermeture d'un
    sortie.close();                    // et de l'autre fichier.
}
```

III\_fichier\_texte.cpp

## eof() et fail() lors de lectures

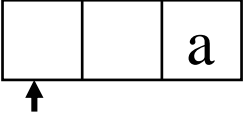
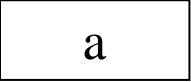
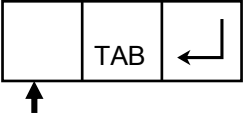

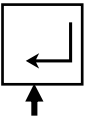
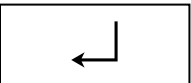

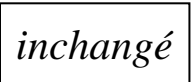
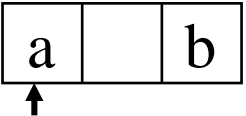
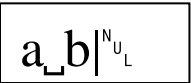
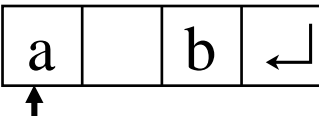
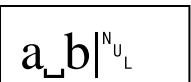
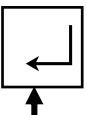
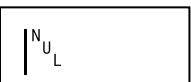


- eof() (« end of file ») devient vrai si une opération tente de lire après le dernier caractère d'un fichier.
  - Le caractère suivant un mot/nombre doit être lu pour détecter la fin du mot/nombre.
    - Aucun caractère après un mot/nombre valide → eof() vrai et fail() faux.
- fail() devient vrai si la lecture échoue.
  - Mauvais caractère pour une valeur (ex. lettre au lieu d'un nombre)
    - Alors fail() vrai et eof() faux.
  - Aucune valeur à lire (fin de fichier atteinte sans valeur)
    - Alors fail() et eof() sont vrais.
- Toute lecture échoue si une de ces conditions est déjà vraie.
  - Utiliser `entree.clear()` pour pouvoir lire à nouveau.

## Exemples eof() et fail()

entree		nombre	entree.fail()	entree.eof()
<div>entree</div> <div>1 2 3</div> <div>↑                      ↑</div>	entree >> nombre;	123	0	1
<div>1 2 3</div> <div>↑                      ↑</div>	entree >> nombre;	123	0	0
<div>1 2 3 a</div> <div>↑                      ↑</div>	entree >> nombre;	123	0	0
<div>TAB ↵</div> <div>↑                      ↑</div>	entree >> nombre;	<i>inchangé</i>	1	1
<div>a b c</div> <div>↑                      ↑</div>	entree >> nombre;	<i>inchangé</i>	1	0
		mot		
<div>a b c</div> <div>↑                      ↑</div>	entree >> mot;	abc  <sup>N<sub>u</sub><sub>L</sub></sup>	0	1
<div>a b c</div> <div>↑                      ↑</div>	entree >> mot;	abc  <sup>N<sub>u</sub><sub>L</sub></sup>	0	0
<div>TAB ↵</div> <div>↑                      ↑</div>	entree >> mot;	<i>inchangé</i>	1	1

Les exemples sont indépendants. Les flèches noires/grises sont les positions avant/après la lecture.

## Exemples eof() et fail()

entree		car	entree.fail()	entree.eof()
	<code>entree &gt;&gt; car;</code>		0	0
	<code>entree &gt;&gt; car;</code>		1	1
	<code>entree.get(car);</code>		0	0
	<code>entree.get(car);</code>		1	1
	<code>getline(entree,           phrase);</code>		0	1
	<code>getline(entree,           phrase);</code>		0	0
	<code>getline(entree,           phrase);</code>		0	0
	<code>getline(entree,           phrase);</code>		1	1

Les exemples sont indépendants. Les flèches noires/grises sont les positions avant/après la lecture.



```
#include <iostream> // Pour l'utilisation de cin et cout.
#include <fstream>   // Pour l'utilisation des fichiers.
#include <string>     // Pour l'utilisation des strings.
using namespace std;

int main()
{
    cout << " Quel est le nom du fichier a lire: ";
    string nomFichier;
    cin >> nomFichier;
    ifstream entree(nomFichier); // Ouverture en lecture de entree.
    //NOTE: Avant C++11 il fallait utiliser nomFichier.c_str()

    string mot;
    entree >> mot;
    cout << "Le premier mot du fichier est: " << mot << endl;
}
```

# Format d’affichage / écriture de fichiers

Expression	Description	Exemple
endl	★ « end line », ajoute un caractère de fin de ligne; passe à la prochaine ligne.	cout << "a" << endl << "b";
setw(nb_col)	★ Précise le nombre de colonnes (caractères) ★ <b>Actif uniquement pour la prochaine donnée affichée</b>	cout << setw(5) << 123;
setf(ios::mode) setiosflags(ios::mode)	★ Précise un ou des modes d’affichage	cout.setf(ios::fixed); cout << setiosflags(ios::fixed);
unsetf(ios::mode) resetiosflags(ios::mode)	★ Permet d’enlever une ou des spécifications actives	cout.unsetf(ios::fixed); cout << resetiosflags(ios::fixed);
flags()  flags(ios::mode)	★ Permet de connaître les spécifications courantes ★ Permet de changer pour des nouvelles spécifications	ios::fmtflags ancien = cout.flags(); cout << fixed << 1.4; cout.flags(ancien); cout << 1.4;

## Format d'écriture des réels

Expression	Description	Exemple
<code>setprecision(nb_dec)</code>	★ Précise le nombre de décimales à afficher pour une donnée réelle	<code>cout &lt;&lt; setprecision(6);</code>
<code>fixed</code>	★ Nombre de décimales fixes après le point; ex: -12.345000	<code>cout &lt;&lt; fixed &lt;&lt; -12.345;</code>
<code>scientific</code>	★ Notation scientifique avec précision fixe; ex: -1.234500e+001	<code>cout &lt;&lt; scientific &lt;&lt; -12.345;</code>
<code>resetiosflags( ios::floatfield)</code>	★ S'assure que les modes fixed ou scientific sont désactivés ★ Nombre total de chiffres (avant et après le point) borné; ex: -12.345 -1.23457e+006	<code>cout &lt;&lt; resetiosflags( ios::floatfield);</code> <code>cout &lt;&lt; -12.345;</code> <code>cout &lt;&lt; -1234567.89;</code>

## Affichage de nombres entiers

```
cout << 12;
```

```
cout << setw(1) << 12;
```

```
cout << setw(7) << 12;
```

```
cout << -12;
```

```
cout << setw(5) << -12;
```

1	2	3	4	5	6	7	8	9	10	11	12
1	2										
1	2										
					1	2					
-	1	2									
		-	1	2							

Entre chaque instruction nous ajoutons: `cout << endl << endl;`

## Affichage de nombres réels

```
cout.setf(ios::fixed);
cout << setprecision(2);
```

```
cout << 1.2;
```

```
cout << setw(10) << 1.237;
```

1	2	3	4	5	6	7	8	9	10	11	12
1	.	2	0								
						1	.	2	4		

```
cout << setprecision(5);
```

```
cout << setw(10) << -5.4321;
```

```
cout << setw(6)
    << -1.212125321;
```

		-	5	.	4	3	2	1	0		
-	1	.	2	1	2	1	3				

Entre chaque instruction nous ajoutons: `cout << endl << endl;`

## Affichage de nombres réels

```
cout.setf(ios::scientific);
cout << setprecision(2);
```

```
cout << 1.2;
```

```
cout << setw(10) << 1.237;
```

```
cout << setprecision(5);
```

```
cout << setw(10) << -5.4321;
```

```
cout << setw(6)
    << -1.212125321;
```

1	2	3	4	5	6	7	8	9	10	11	12
1	.	2	0	e	+	0	0				
		1	.	2	4	e	+	0	0		

-	5	.	4	3	2	1	0	e	+	0	0
-	1	.	2	1	2	1	3	e	+	0	0

Entre chaque instruction nous ajoutons: `cout << endl << endl;`

## Affichage de caractères

```
cout << 'a';
```

```
cout << 'a'
```

```
<< setw(3) << 'b'
```

```
<< setw(2) << 'c';
```

1	2	3	4	5	6	7	8	9	10	11	12
a											
a			b		c						

Entre chaque instruction nous ajoutons: `cout << endl << endl;`

## Affichage de chaîne de caractères

```
cout << "Salut";  
cout << setw(8)  
      << "l'ami";
```

1	2	3	4	5	6	7	8	9	10	11	12
S	a	l	u	t							
			l	'	a	m	i				

Entre chaque instruction nous ajoutons: `cout << endl << endl;`



## `ostream& put(char car)`

- Le caractère passé en paramètre est ajouté au flot de sortie sans influence des manipulateurs
- Exemple:
  - `cout.put('B');`
  - `cout.put(carLu); // carLu est une var. déclarée`
  - `// char carLu;`

# Formatage – Spécification

(les plus utilisées)

Toutes les spécifications doivent être utilisées sous la forme `ios::spécification`, avec `setf`, `unsetf` ...

<code>boolalpha</code>	Lit/écrit les <i>bool</i> en texte "true" et "false".
<code>dec / oct / hex</code>	Conversion décimale / octale / hexadécimale pour les entiers.
<hr/>	
	<b>Pour les sorties:</b>
<code>left / right</code>	Cadrage à gauche ou droite ou avec espaces au centre (après le signe ou la base).
<code>fixed / scientific</code>	Format des réels avec décimales fixes ou notation scientifique.

# Formatage – Spécification

(les autres)

internal	Espaces au centre (après le signe ou la base), en sortie. S'oppose à left / right.
uppercase	Affichage hexadécimal avec majuscules.
showbase	Affichage de la base (0 pour octal, 0x pour hexadécimal).
showpoint	Affichage du point décimal, toujours.
showpos	Affichage du signe (+) des valeurs positives.
unitbuf	Vide le tampon du flot après chaque opération d'écriture.
skipws	Saute les espaces lors de lectures avec >>.
adjustfield	Équivalent à (left   right   internal).
basefield	Équivalent à (dec   oct   hex).
floatfield	Équivalent à (scientific   fixed).

# Manipulateurs

- Une spécification est activable par un manipulateur de même nom.
  - Existent pour toutes les spécifications sauf les ...field (qui incluent plusieurs spécifications).
  - ex: cout << boolalpha << showpos << fixed << right;
  - cin >> skipws >> hex;
- Une spécification est désactivable par un manipulateur no...
  - Existent pour les spécifications indépendantes, pas pour dec/hex/oct, fixed/scientific, internal/left/right.
  - ex: cout << noboolalpha << noshowpos;
  - cin >> noskipws;

# Manipulateurs (suite)

⇒ set... définis dans le fichier **iomanip**

Mots clés	Description
setbase(n)	Changement de base pour la conversion; <i>n</i> doit être 8, 10 ou 16.
setiosflags(s)	Active une ou plusieurs spécifications.
resetiosflags(s)	Désactive une ou plusieurs spécifications.
	<b>Pour les entrées:</b>
ws	Saute les espaces maintenant.
	<b>Pour les sorties:</b>
endl	Ajout d'un saut de ligne. (Équivalent à '\n' puis flush.)
ends	Ajout d'un caractère NUL. (Équivalent à '\0'.)
flush	Vidage du tampon d'un flot vers la sortie associée.
setprecision(n)	Modification de la précision utilisée pour les nombres réels.
setw(n)	Modification de la largeur du prochain affichage.
setfill(c)	Modification du caractère de remplissage.

# Formatage

```
int main()
{
    int    index    = -23;
    double distance = 12.345;
    char   lettre   = 'X';
    string nom      = "Jeremi Desir";
    bool   vrai     = true;

    cout << "La valeur de Index est "    << index    << endl;
    cout << "La valeur de Distance est " << distance << endl;
    cout << "La valeur de Lettre est "   << lettre   << endl;
    cout << "La valeur de Nom est "      << nom      << endl;

    index = 31;
    cout << "La valeur decimale/octale/hexadecimale de Index est "
        << dec << index << '/' << oct << index << '/' << hex << index << endl;
    cout << "La valeur hexadecimale de Lettre est " << (unsigned)lettre << endl;
    cout << "vrai est " << vrai << " ou " << boolalpha << vrai << endl;

    cout << "Donner une valeur decimale --> ";
    cin  >> index;
    cout << "La valeur lue en hexadecimal est " << index << endl;

    cout << "Entrez des espaces suivis de deux caracteres, le deuxieme pouvant etre un espace -->
";
    char car1, car2;
    cin >> car1 >> noskipws >> car2;
    cout << "Les deux caracteres lus sont '" << car1 << "' et '" << car2 << "'" << endl;
}
```

III\_formatage.cpp

## Formatage (suite)

Résultat de l'exécution:

La valeur de Index est -23

La valeur de Distance est 12.345

La valeur de Lettre est X

La valeur de Nom est Jeremi Desir

La valeur decimale/octale/hexadecimale de Index est 31/37/1f

La valeur hexadecimale de Lettre est 58

vrai est 1 ou true

Donner une valeur decimale --> 999

La valeur lue en hexadecimal est 3e7

Entrez des espaces suivis de deux caracteres, le deuxieme  
pouvant etre un espace -->    x

Les deux caracteres lus sont 'x' et ' '