Information, Calcul, Communication (partie programmation) : Entrées/Sorties

Jean-Cédric Chappelier

Laboratoire d'Intelligence Artificielle
Faculté I&C

Rappel du calendrier

		MOOC	décalage / MOOC	exercices prog. 1h45 Jeudi 9-11	cours prog. 45 min. Jeudi 11-12
1	22.09.22	--	-1	prise en main	Bienvenue/Introduction
2	29.09.22	1. variables	0	variables / expressions	variables / expressions
3	06.10.22	2. if	0	if – switch	if – switch
4	13.10.22	3. for/while	0	for / while	for / while
5	20.10.22	4. fonctions	0	fonctions (1)	fonctions (1)
6	27.10.22		1	fonctions (2)	fonctions (2)
7	03.11.22	5. tableaux (vector)	1	vector	vector
8	10.11.22	6. string + struct	1	array / string	array / string
9	17.11.22		2	structures	structures
10	24.11.22	7. pointeurs	2	pointeurs	pointeurs
11	01.12.22		-	entrées/sorties	entrées/sorties
12	08.12.22		-	erreurs / exceptions	erreurs / exceptions
13	15.12.22		-	révisions	théorie : sécurité
14	22.12.22	8. étude de cas	-	**Examen final (2h45)**	

(ne sont pas sur le MOOC)

Objectifs du cours d'aujourd'hui

- Présenter les bases des entrées/sorties en C++ :
  - les entrées/sorties clavier/écran (`cin`/`cout`)
  - les fichiers
  - formatage des entrées/sorties

Interagir avec le monde : les entrées/sorties

Jusqu'à présent, les possibilités d'interaction de vos programmes sont restées très limitées :

> quelques exemples rudimentaires de saisie au clavier (`cin`) et d'affichage à l'écran (`cout`).

On peut évidemment faire beaucoup plus !

Les interactions d'un programme avec « l'extérieur » sont gérées par des **instructions d'entrée/sortie** et ce qu'on appelle des « **flots** ».

Un flot correspond à un canal d'échange de données entre le programme et l'extérieur.

`cin` est le nom de la variable associée par défaut au flot d'entrée.
On l'appelle « *entrée standard* ».

`cout` est le nom de la variable associée par défaut au flot de sortie.
On l'appelle « *sortie standard* ».

Jean-Cédric Chappelier
& Jamila Sam

Buts des entrées/sorties

- *intégrer*, au sein du programme, des *données issues de l'extérieur*, par exemple saisies au clavier ou lues dans un fichier
  - ☞ nécessité de **fonctions d'entrée**, permettant d'associer des informations externes aux variables du programme
- *communiquer des données internes*
  - ☞ nécessité de **fonctions de sortie**, permettant de visualiser (en spécifiant éventuellement le format) les valeurs des variables du programme
- *sauvegarder les résultats produits* par les programmes, par exemple pour des utilisations ultérieures
  - ☞ nécessité de **fonctions de lecture/écriture**, permettant
    1. de stocker dans des **fichiers** les données produites par les programmes
    2. de récupérer dans le programme les données contenues dans de tels fichiers

Entrées-sorties clavier/écran

À l'heure actuelle, les interactions les plus simples (« naturelles » ?)
avec un programme se font via l'écran pour la sortie et le clavier pour les entrées.

En C++, ces « **flots** » sont représentés respectivement par `cout` et `cin`.

`cin` et `cout` sont définis dans le fichier de définitions `iostream`. Pour les utiliser il faut donc faire

```
#include <iostream>
```

au début du programme.

Note : leur vrai nom est « `std::cin` » et « `std::cout` ». C'est le

```
using namespace std;
```

qui permet de n'écrire que « `cin` » et « `cout` ».

Affichage à l'écran

```
cout << expr1 << expr2 << ... ;
```

affiche à l'écran les valeurs des **expressions** *expr1*, *expr2*, ...

Remarque : par défaut, les valeurs affichées ne sont pas séparées par des espaces
☞ ajoutez des `' '` (1 espace), si nécessaire.

Exemple d'affichage :

```
int a(1);
cout << "intervalle : [" << -a << ", " << a << "]" << endl;
```

➪ `intervalle : [-1, 1]`

Remarque : `endl` permet à l'affichage d'aller à la ligne.

Jean-Cédric Chappelier
& Jamila Sam

Lecture au clavier

`cin >> var1 >> var2 >> ... ;`

permet à l'utilisateur de saisir au clavier une liste de valeurs `val1`, `val2`, ... qui seront stockées dans les variables `var1`, `var2`, ...

Remarque : Lorsque plusieurs valeurs sont lues à la suite, le **caractère séparateur** de ces valeurs est le retour à la ligne **ou l'espace**.

Attention ! On ne peut donc pas lire en une fois une ligne entière contenant des blancs (par exemple saisie d'un nom composé) avec `cin`

☞ il faut utiliser `getline` pour lire une ligne entière (contenant des espaces)
Conseil : faire précéder les « `getline(cin, ...)` » de :

`cin >> ws;`

Exemple :

```
string phrase_a_lire;
...
cin >> ws;
getline(cin, phrase_a_lire);
```


Jean-Cédric Chappelier
& Jamila Sam

Exemple simple d'entrée-sortie

```
#include <iostream>
int main () {
  int i;
  double x;
  cout << "Valeurs pour i et x : ";
  cin >> i >> x ;
  cout << "lus : " << i << ", " << x << endl;
  return 0;
}
```

Contrôle strict de l'entrée clavier

Vous avez peut être remarqué que la lecture sur `cin` est parfois « *capricieuse* » (p.ex. quand on ne lui donne pas ce qu'elle attend).

Exemple tiré des séries d'exercices :

```
int lu;
do {
  cout << "entrez un nombre entre 1 et 10 : ";
  cin >> lu;
} while ((lu < 1) or (lu > 10));
```

Si vous tapez `'a'` (ou n'importe quoi qui ne soit pas un nombre)
☞ boucle infinie ! !

Comment éviter cela ?

☞ en contrôlant l'état du flot `cin` (et en « jetant à la poubelle » ce qui ne convient pas).

Jean-Cédric Chappelier
& Jamila Sam

Contrôle strict de l'entrée clavier

```
#include <limits>
...
int lu(0);
do {
  cout << "entrez un nombre entre 1 et 10 : ";
  cin >> lu;
  if (cin.fail()) {          // teste si cin "ne va pas bien"
    cout << "Je vous ai demandé d'entrer un nombre, "
         << "pas du charabia !" << endl;
    // remet cin dans un état lisible
    cin.clear();
    // "jette" tout le reste de la ligne
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
  }
} while ((lu < 1) or (lu > 10));
```

Remarque : Une autre solution consiste à ne **lire** l'entrée que **caractère par caractère** avec la fonction `get()` (`c = cin.get()`) et à traiter ensuite la séquence lue, de façon totalement contrôlée par notre programme.

Mais cette bonne solution (« tokenization », `flex`) est trop avancée dans sa mise en œuvre pour être détaillée dans ce cours.

Sortie erreur standard

En plus de `cin` et `cout`, il existe une **sortie d'erreur standard**, `cerr`.

Par défaut `cerr` est envoyée sur le terminal, comme `cout`. Mais il s'agit bien d'un flot séparé !

(Note : dans plusieurs systèmes d'exploitation, on peut effectivement séparer la sortie standard de la sortie d'erreur standard.)

☞ Conseil : Pour afficher des messages d'erreur depuis votre programme, préférez `cerr` plutôt que `cout`.

Jean-Cédric Chappelier
& Jamila Sam

Les entrées/sorties avec des fichiers

Pour sauvegarder de façon permanente (c'est-à-dire rendre disponible après la fin de l'exécution du programme) les données produites par un programme, une solution est d'utiliser le **système de fichiers** fourni par le système d'exploitation.

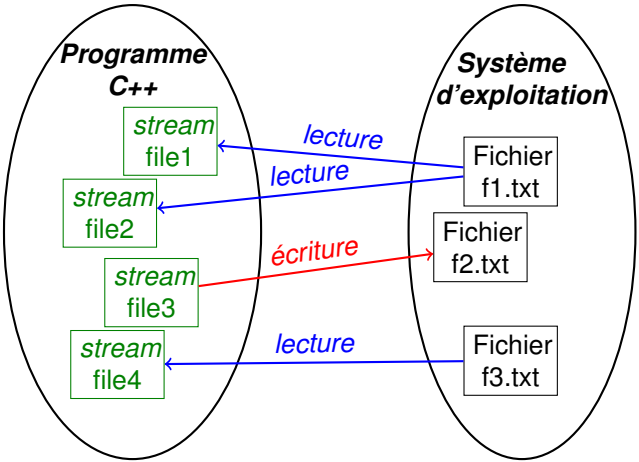
Il est donc nécéssaire d'avoir un objet informatique permettant de gérer les entrées et les sorties du programme vers le système de fichiers.

Cet objet informatique (appelé un **flot**, « stream » en anglais) **représente** en fait *un lien à un fichier physique* stocké dans le disque et géré par le système de fichiers.

L'idée de base des flots est de **séparer l'aspect logique** des entrées-sorties (c'est-à-dire *leur intégration dans des programmes*) **de** leur **aspect physique** (c'est-à-dire *leur réalisation par le biais de périphériques particuliers*).

Jean-Cédric Chappelier
& Jamila Sam

Les entrées/sorties avec des fichiers (2)



Les types « stream »

Comme pour de nombreuses autres notions C++, pour pouvoir utiliser les flots, il faut tout d'abord inclure les fichiers de définitions correspondant, `iostream` et `fstream` :

```
#include <iostream>
#include <fstream>
```

Deux nouveaux types sont alors disponibles :

- `ifstream` (pour *input file stream*) qui définit un flot d'**entrée** (similaire à `cin`)
- `ofstream` (pour *output file stream*) qui définit un flot de **sortie** (similaire à `cout`)

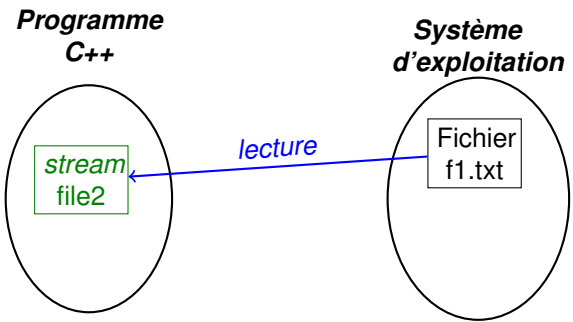
Attention ! On ne peut jamais copier de flot (`ifstream`, `ofstream`, ...), ni les passer par valeur. Il faut impérativement les passer par référence !

Jean-Cédric Chappelier
& Jamila Sam

Schéma général d'utilisation des flots

Mécanisme général pour la mise en œuvre d'entrées-sorties :

- **création d'un flot** (d'entrée ou de sortie) par la **déclaration** d'une variable du type correspondant (`ifstream` ou `ofstream`)
- **lien** de la variable déclarée **avec un dispositif** d'entrée-sortie **physique** (fichier)



Jean-Cédric Chappelier
& Jamila Sam

Schéma général d'utilisation des flots

Mécanisme général pour la mise en œuvre d'entrées-sorties :

- **création d'un flot** (d'entrée ou de sortie) par la **déclaration** d'une variable du type correspondant (`ifstream` ou `ofstream`)
- **lien** de la variable déclarée **avec un dispositif** d'entrée-sortie **physique** (fichier)
- **utilisation** de la variable déclarée et liée pour effectivement réaliser les entrées-sorties
- **fermeture** du flot

La plupart de ces actions se font en applicant des fonctions spécifiques définies pour les types `stream`.

Ces fonctions sont appelées avec la syntaxe suivante :

```
nom_de_flot.nom_de_fonction(arg1, ...);
```


Jean-Cédric Chappelier
& Jamila Sam

Création d'un flot

Pour créer un flot, il suffit de déclarer une variable du type correspondant.

Exemple :

```
ifstream entree;
ofstream sortie;
```

déclare deux variables `entree` et `sortie`, respectivement de type `ifstream` et `ofstream`.

Jean-Cédric Chappelier
& Jamila Sam

Lien avec un fichier

Dans le cas des fichiers textes (fichiers lisibles par les humains), l'association d'un flot d'entrée-sortie avec le fichier se fait par le biais de la fonction spécifique `open()`.

Exemple :

```
ifstream entree;
entree.open("test.txt");
```

associe le stream `entree` avec le fichier physique `test.txt`.

Dans le cas des fichiers binaires, il faut ajouter un argument supplémentaire :

`ios::in|ios::binary` pour la lecture
`ios::out|ios::binary` pour l'écriture

Exemple :

```
ifstream entree;
entree.open("a_lire.zip", ios::in|ios::binary);

ofstream sortie;
sortie.open("a_ecrire.exe", ios::out|ios::binary);
```


Jean-Cédric Chappelier
& Jamila Sam

Lien avec un fichier (2)

Par défaut, un « `ofstream` » ouvre le fichier en mode « *écrasement* »
(c'est-à-dire détruit le contenu du fichier si il existe déja)

On souhaite parfois pouvoir ouvrir le fichier en mode « *ajout* » (« append »)
(c'est-à-dire écrire en fin de fichier)

Cela se fait aussi en ajoutant un argument supplémentaire à `open` :

```
ofstream sortie;
sortie.open("a_poursuire.txt", ios::app);
```

Dans le cas de fichiers en binaire :

```
sortie.open("a_completer.data",
            ios::binary|ios::app);
```


Jean-Cédric Chappelier
& Jamila Sam

Lien avec un fichier (3)

L'association du stream d'entrée ou de sortie avec un fichier particulier peut aussi se faire directement par une **initialisation** lors de la déclaration du stream :

Exemple :

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

déclare la variable `entree` et l'associe au fichier texte `test.txt`.

 Dans le cas de fichiers en binaire :

```
ifstream entree("a_lire.zip", ios::in|ios::binary);

ofstream sortie("a_ecrire.exe", ios::out|ios::binary);
```

Dans le cas de fichiers en écriture en mode « ajout » :

```
ofstream sortie("a_poursuivre.txt", ios::app);
```


Jean-Cédric Chappelier
& Jamila Sam

Validité des accès fichier

Il est **important** de **vérifier** que le lien du flot avec le fichier **s'est bien passé** (p.ex. le fichier existe, il est bien lisible, ...).

Ceci est fait en testant l'état du flot, typiquement après l'appel à la fonction `open()` :

```
ifstream entree("test.txt") ;
if (not entree.fail()) { ...\\ Utilisation de entree... }
```

En cas de problème, la fonction `fail()` du flot retourne la valeur `true` si le flot n'est pas dans un état correct pour la prochaine opération (lecture ou écriture). Elle renvoie `false` si le flot est prêt.

(voir aussi exemples complets plus loin)

Jean-Cédric Chappelier
& Jamila Sam

Utilisation des flots

L'utilisation des variables de type `ifstream` ou `ofstream` dans les programmes pour réaliser les entrées-sorties se fait de la même façon que pour les flots particuliers `cin` et `cout`, à l'aide des opérateurs `<<` et `>>`.

```
ifstream entree("fichier_entree.txt");
if (entree.fail()) { ... ; return; }

ofstream sortie("fichier_sortie.txt");
if (sortie.fail()) { ... ; return; }

...
string mot;
entree >> mot; // lit un mot dans le fichier "fichier_entree.txt"
sortie << mot; // écrit ce mot dans "fichier_sortie.txt"
...
entree >> ws; // "nettoye" les blancs pour la lectutre avec getline
getline(entree, mot);
...
```


Jean-Cédric Chappelier
& Jamila Sam

Utilisation des flots – Remarques

Dans un contexte de test, les opérateurs `<<` et `>>`, ainsi que `getline`, retournent en fait « not *flot*`.fail()` »,

Exemples :

```
if (cin >> x) { // on a pu lire x }

while (entree >> nom >> age) {
  // on a eu un nom (1 seul mot) et un age
}
while (getline(entree, ligne)) {
  // on peut traiter la ligne
}
```

En cas d'erreur, une fonction utile pour tester si l'erreur est liée à la fin du fichier (ou à autre chose) est la fonction `eof()`.

Exemple :

```
while (entree >> age) { ...  }
if (not entree.eof()) {
  cerr << "je voulais un entier" << endl;
  // etc...
}
```


Jean-Cédric Chappelier
& Jamila Sam

Fermeture des flots

La fermeture du stream se fait par la fonction `close()`.

Exemple :

```
ifstream entree("fichier_entree");
...
entree.close();
```

Attention ! NE PAS oublier de fermer tout fichier ouvert !
En particulier en écriture : vous risqueriez sinon d'avoir des surprises...

Note : Ceci dit la norme C++ garantit l'appel de `close()` en fin de vie d'une variable de type `stream`...
...mais attention en cas d'utilisations multiples (faire `close` avant de refaire `open`) !

Jean-Cédric Chappelier
& Jamila Sam

Exemple de lecture à partir d'un fichier

Exemple de programme de lecture d'un fichier texte de nom « `test` » :

```
int main() {
  string nom_fichier("test");
  ifstream entree(nom_fichier);

  if (entree.fail()) {
    cerr << "Erreur : impossible de lire le fichier "
         << nom_fichier << endl;
  } else {
    string mot;

    while (entree >> mot) {
      cout << "lu : \"" << mot << '"' << endl;
    }
    entree.close() ;
  }
  return 0;
}
```


Jean-Cédric Chappelier
& Jamila Sam

Écriture dans un fichier

Exemple de programme d'écriture dans un fichier texte :

```
int main() {
  string nom_fichier;

  cout << "Dans quel fichier voulez vous écrire ? ";
  cin >> ws;
  getline(cin, nom_fichier);

  ofstream sortie(nom_fichier);
  if (sortie.fail()) {
    cerr << "Erreur : impossible d'écrire dans le fichier "
         << nom_fichier << endl;
  } else {
    string phrase;

    cout << "Entrez une phrase : ";
    cin >> ws;
    if (getline(cin, phrase)) {
      sortie << phrase << endl;
    }
    sortie.close();
  }
  return 0;
}
```


Jean-Cédric Chappelier
& Jamila Sam

Sorties formatées

Un certain nombre de paramètres pour le format des sorties peuvent être explicitement spécifiés.
Il peuvent être de deux formes :

- soit des **manipulateurs** appliqués à l'instruction `<<`
- soit des **options de configurations** pour une variable de type `ostream` (dont `cout` en particulier)

(Note : certains formatages peuvent s'obtenir indifféremment des deux façons.)

Pour pouvoir utiliser **les manipulateurs**, il faut utiliser le fichier de définitions `iomanip` :

```
#include <iomanip>
```

Les **manipulateurs** s'utilisent de la façon suivante :

```
cout << manipulateur << expression << ...
```


Jean-Cédric Chappelier
& Jamila Sam

Manipulateurs

Exemple de quelques manipulateurs :

① `setprecision` : précision désirée dans l'affichage des nombres.

Exemple :

```
for (int i(2); i <= 5; ++i) {
  cout << setprecision(i) << i << " " << 123.46
       << endl;
}
```

```
2 1.2e+02
3 123
4 123.5
5 123.46
```

② `boolalpha` : affiche les booléens en forme textuelle (`true`, `false`).

Exemple : `cout << boolalpha << est_premier;`

Manipulateurs (2)

④ `setw` : longueur d'une chaîne de caractères.
La donnée qui suit ce manipulateur est affichée sur (au moins) le nombre de caractères spécifiés avec un cadrage à droite.
Ceci est pratique pour représenter des nombre en colonnes :

```
constexpr int col1(14);
constexpr int col2(5);
cout << setw(col1) << "un :"
     << setw(col2) << 1   << endl;
cout << setw(col1) << "cent deux :"
     << setw(col2) << 102 << endl;
cout << setw(col1) << "très froid :"
     << setw(col2) << -18 << endl;
cout << setw(col1) << "très très très froid :"
     << setw(col2) << -273.15 << endl;
```

```
            un :    1
     cent deux :  102
    très froid :  -18
très très très froid :-273.15
```

Manipulateur setw

Autre exemple :

```
for (int i(1); i < 5; ++i) {
  cout << i << " " << setw(i) << 8 << endl;
}
```

```
1 8
2  8
3   8
4    8
```

`setw` peut aussi s'utiliser **en lecture**, pour lire au plus un nombre donné de caractères.

```
string code;
cout << "Donnez moi un code de 5 caractères : ";
cin >> setw(5) >> code;
cout << "lu : " << code << endl;
```

```
Donnez moi un code de 5 caractères : abcdefgh
lu : abcde

Donnez moi un code de 5 caractères : ab
lu : ab
```

Manipulateurs (4)

Les flots d'**entrée** ont aussi le manipulateur suivant, qui peut s'avérer utile :

⑥ `ws` : saute tous les caractères blancs

Exemple :

```
cin >> ws;
getline(cin, phrase);
```

Les options des flots

Les **options** de flots sont configurées avec la fonction `setf()`, de la façon suivante :

$$flot\texttt{.setf(ios::}option\texttt{);}$$

Et pour annuler une option, faire :

$$flot\texttt{.unsetf(ios::}option\texttt{);}$$

Quelques exemples d'options :

① `showpoint` : affiche explicitement la partie fractionnaire des nombres réels ;

```
cout.setf(ios::showpoint);
cout << 16.0 << ", " << 16.1 << ", " << 16 << endl;
```

```
16.0000, 16.1000, 16
```


Jean-Cédric Chappelier
& Jamila Sam

Options de flot (2)

② `fixed/scientific` : affichage fixe ou scientifique (en puissance de 10) des nombres

```
cout.setf(ios::fixed);
cout << sqrt(200) << endl;
```

```
14.14214
```

```
cout.setf(ios::scientific);
cout << sqrt(200) << endl;
```

```
1.41421e+01
```


Jean-Cédric Chappelier
& Jamila Sam

Options de flot (3)

③ `left` : affichage à gauche

```
cout << setw(15) << "abcde" << "x" << endl;
cout << setw(15) << "ab"    << "x" << endl;
cout.setf(ios::left);
cout << setw(15) << "abcde" << "x" << endl;
cout << setw(15) << "ab"    << "x" << endl;
```

```
          abcdex
             abx
abcde          x
ab             x
```

Pour revenir à droite :
`flot.setf(ios::right);`
ou
`flot.unsetf(ios::left);`

Jean-Cédric Chappelier
& Jamila Sam

Manipulateur ou option ?

`out << hex`	`out.setf(ios::hex, ios::basefield)`
`out << oct`	`out.setf(ios::oct, ios::basefield)`
`out << dec`	`out.setf(ios::dec, ios::basefield)`
`out << setprecision(p)`	`out.precision(p)`
`out << setw(w)`	`out.width(p)`
`out << setfill(c)`	`out.fill(c)`
`in >> ws`	—
`out << showbase`	`out.setf(ios::showbase)`
`out << showpoint`	`out.setf(ios::showpoint)`
`out << fixed`	`out.setf(ios::fixed, ios::floatfield)`
`out << scientific`	`out.setf(ios::scientific, ios::floatfield)`
`out << left`	`out.setf(ios::left, ios::adjustfield)`
`out << right`	`out.setf(ios::right, ios::adjustfield)`

Jean-Cédric Chappelier
& Jamila Sam

Les entrées/sorties

Clavier / Terminal : `cin` / `cout` et `cerr`

Fichier de définitions : `#include <iostream>`

Utilisation :

écriture : `cout << expr1 << expr2 << ... ;`

lecture : `cin >> var1 >> var2 >> ... ;`

Saut à la ligne : `endl`

Lecture d'une ligne entière : `getline(cin, string);`

.....

Formatage :

Manipulateurs	
`#include <iomanip>`	
`cout << manip << expr << ...`	
`dec`, `oct`, `hex`	changement de base
`setprecision(int)`	nombre de chiffres à afficher
`setw(int)`	largeur de colonne
	nombre de caractères à lire
`setfill(char)`	caractère utilisé dans l'alignement
`cin >> ws`	saute les blancs

Options	
`setf(ios::option)`	
`unsetf(ios::option)`	
`ios::left`	alignement à gauche
`ios::showbase`	afficher la base
`ios::showpoint`	afficher toujours la virgule
`ios::fixed`	notation fixe
`ios::scientific`	notation scientifique

Les entrées/sorties (2)

Fichiers : `#include <fstream>`

Flot d'**entrée** (similaire à `cin`) : `ifstream`
Flot de **sortie** (similaire à `cout`) : `ofstream`

Création : *type_flot nom_de_flot*;

Lien (ouverture) : *flot*.open("*fichier*");


ouverture en binaire :

pour lecture : ifstream *flot*("*fichier*", ios::in|ios::binary);
pour écriture : ofstream *flot*("*fichier*", ios::out|ios::binary);

Utilisation : comme `cin` et `cout` :

flot << *expression* << ... ;
flot >> *variable_lue* >> ...;

Test d'échec de ouverture/lecture/écriture sur le flot : *flot*.fail()

Fermeture du fichier : *flot*.close()

Test de fin de fichier : *flot*.eof()

Jean-Cédric Chappelier
& Jamila Sam
EPFL

Ce que j'ai appris aujourd'hui

- à faire communiquer mon programme avec le « monde extérieur » pour
  - afficher des résultat ;
  - saisir des données ;
  - sauvegarder et relire des données ;
- à créer et lire des fichiers ;
- à formater mes résultats.

☞ Je peux maintenant écrire des programmes pouvant **interagir** avec l'utilisateur, mais aussi **manipuler des fichiers** et donc donner de la **persistance** aux données créées.