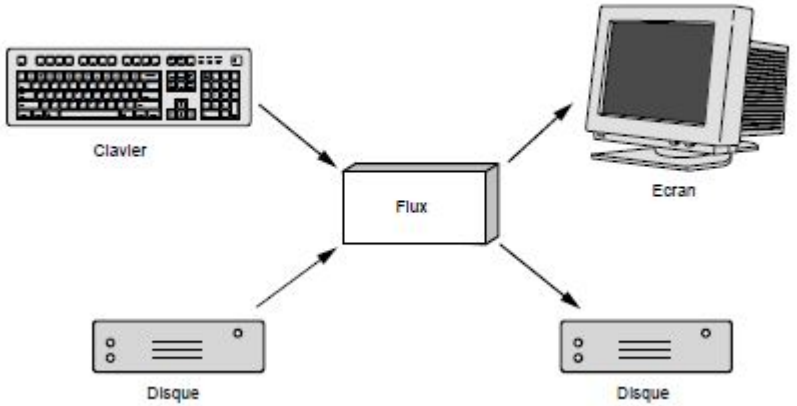
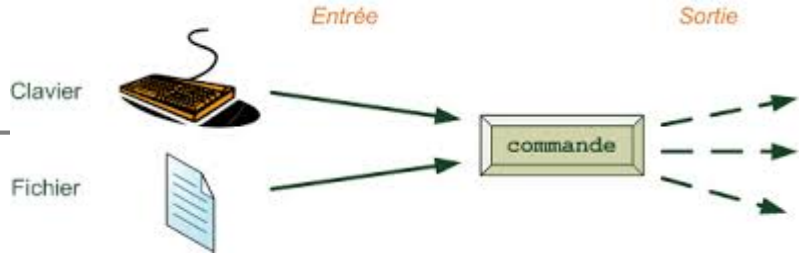
Programmation orientée Objet et C++ pour Eai

Flux & Fichiers



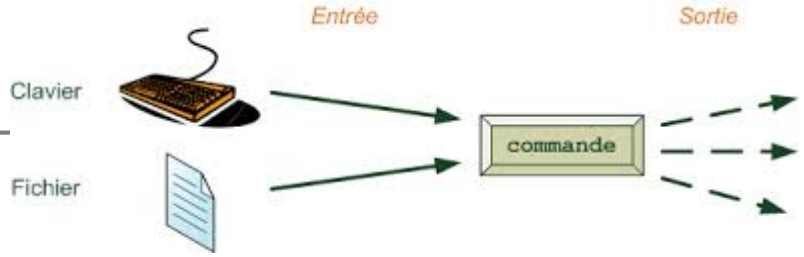
Flux – Définition
(stream in English)



- Entrées/Sorties (E/S) de fichier et de flux (flot) fait référence au transfert de données vers ou depuis un support de stockage
- Il s'agit de canaux permettant d'envoyer (**flux de sortie**) ou de recevoir (**flux d'entrée**) des données
- Ceci **n'existe qu'en C++ avec les classes iostream**
- Les fonctions standard du C (printf/scanf, read/write, etc) existent toujours dans <cstdio>, **mais il faut préférer les nouvelles fonctionnalités C++**

heig-vd

Flux - Prédéfinis

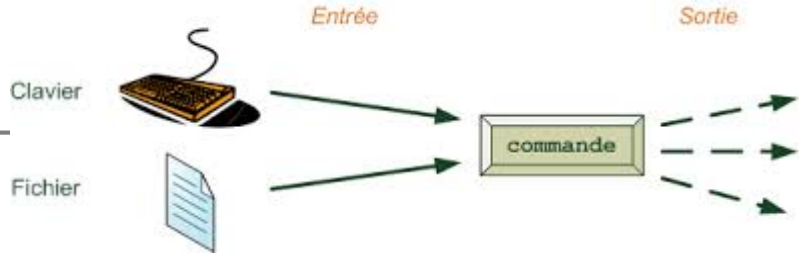


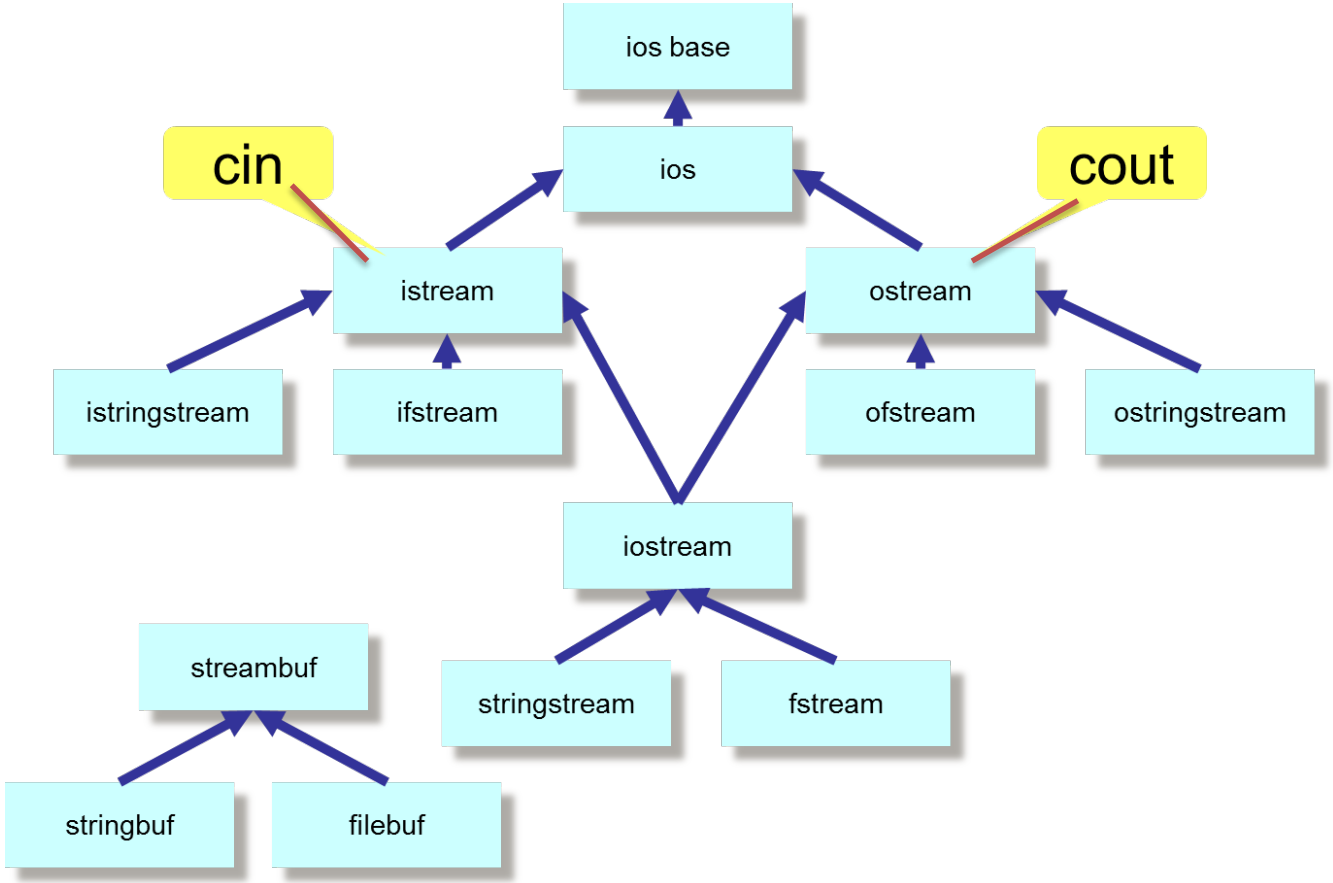
- Depuis le début nous utilisons deux flux "**cin**" et "**cout**" pour le **dialogue homme/machine**

- En fait "**cin**" et "**cout**" sont 2 flux prédéfinis (équivalent à "**stdin**" et "**stdout**" en C)

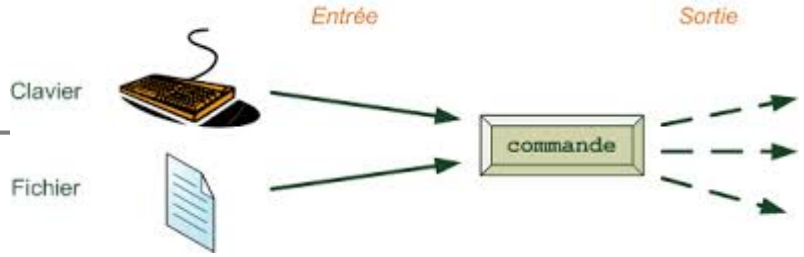
- Il existe encore **2 autres flux prédéfinis**
  - **cerr** sortie des erreurs (stderr en C)
  - **clog** idem mais bufferisé

Classes liées aux flux





Flux - Prédéfinis

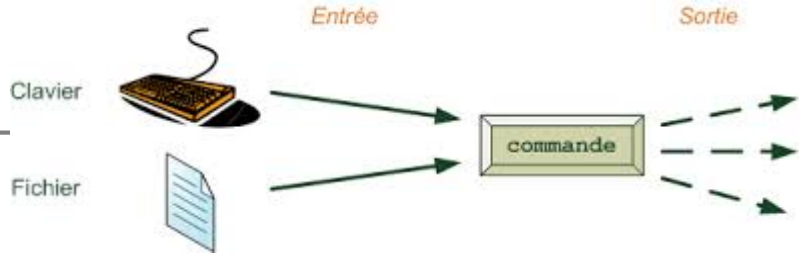


- Nous avons utilisé les opérateurs
  - « **<<** » d'**ostream** pour les écritures ("op. d'insertion")
  - « **>>** » d'**istream** pour les lectures ("op. d'extraction")

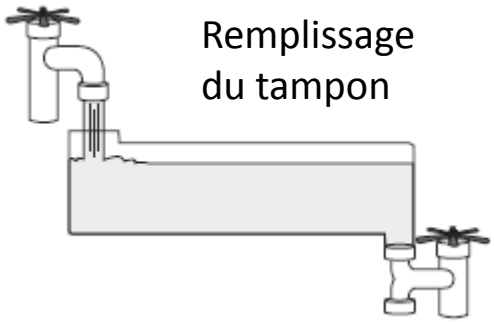
- qui sont **surchargés pour tous les types de base** et que nous **pouvons surcharger pour nos types**

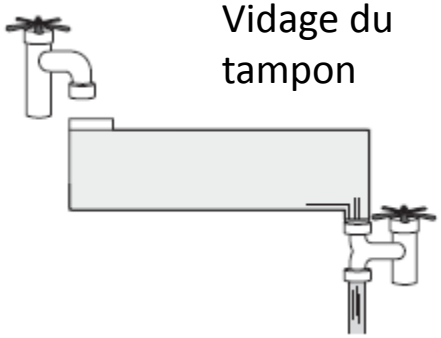
- Cependant, il **ne permettent pas de résoudre tous les problèmes**
  (ex : lecture d'une chaîne comprenant des espaces, etc …)

Tampons des flux



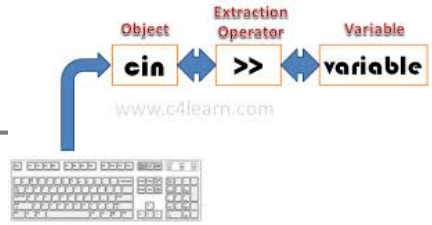
- Les flux utilisent des **tampons**
- Les données sont bien écrites dans le flux, mais elles ne sont pas immédiatement transférées
- Elles sont placées dans un tampon qui se remplit à fur et à mesure et qui est transféré en une seule fois lorsqu'il est plein





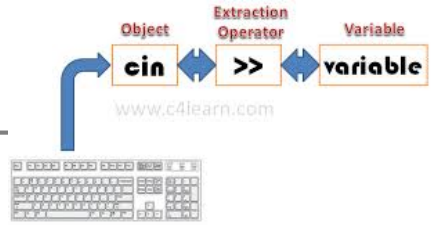
heig-vd

Quelques méthodes de cin



- cin possède un certain nombre d'autres fonctions membres permettant un contrôle plus "fin" des entrées
- Ces méthodes permettent de:
  - lire un caractère unique
  - lire des chaînes
  - ignorer une entrée
  - examiner le caractère suivant du tampon
  - replacer les données dans le tampon

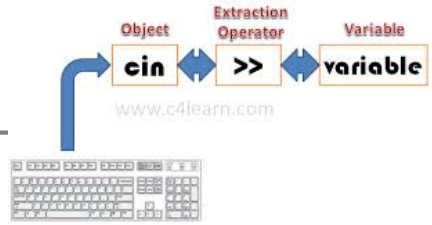
cin – get() 1/2



- **int get();** (sans paramètre)
  - renvoie la valeur du caractère obtenu ou EOF (*End Of File*, fin de fichier) lorsque la fin du fichier est atteinte

```
int main(void)
{
   char ch;
   // Lit les caractères jusqu'au moment où ctrl+z est saisi
   while ( (ch = cin.get()) != EOF)
   {
      cout << "ch : " << ch << endl;
   }
   cout << "\nTerminé !\n";
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

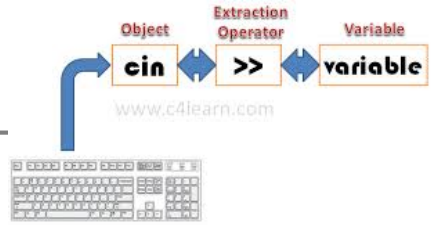
cin – get() 2/2



- **istream & get(char &ch);** (avec paramètre)
  - permet de lire un seul caractère et retourne le flux auquel on l'applique

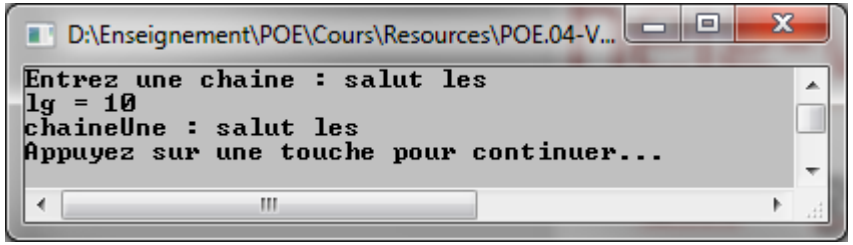
```
int main(void)
{
   char a, b, c;
   cout << "Entrez trois lettres : ";
   cin.get(a).get(b).get(c);
   cout << "a : " << a << "\nb : ";
   cout << b << "\nc : " << c << endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

cin – getline()

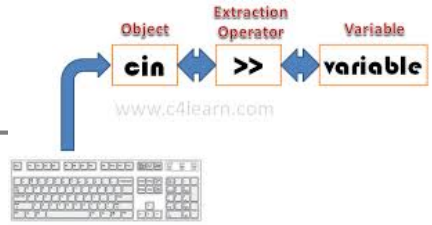


- **istream & getline(char * chaine, int lgMax, char delimiteur = '\n');**
  - permet de lire une chaîne de caractères (rajoute le '\0' de fin de chaîne) et s'interrompt lorsqu'une des deux conditions suivantes est satisfaite:
    - Le caractère "délimiteur a été trouvé
    - Taille -1 caractères ont été lus

```
int main(void)
{
   int lg;
   char chaineUne[256];
   cout << "Entrez une chaine : ";
   cin.getline(chaineUne, 250);
   lg = cin.gcount(); // obtient le nbr de carac. lus (y compris \0)
   cout << "lg = " << lg << endl;
   cout << "chaineUne : " << chaineUne << endl;
   return EXIT_SUCCESS;
}
```



cin – peek() et putback()



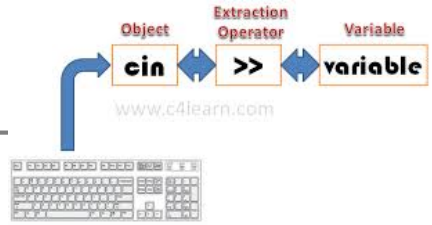
- **int peek();**
  - examine mais n'extrait pas le caractère suivant du flux d'entrée

```
c = cin.peek()  // to peek = jeter un coup d'oeil
```

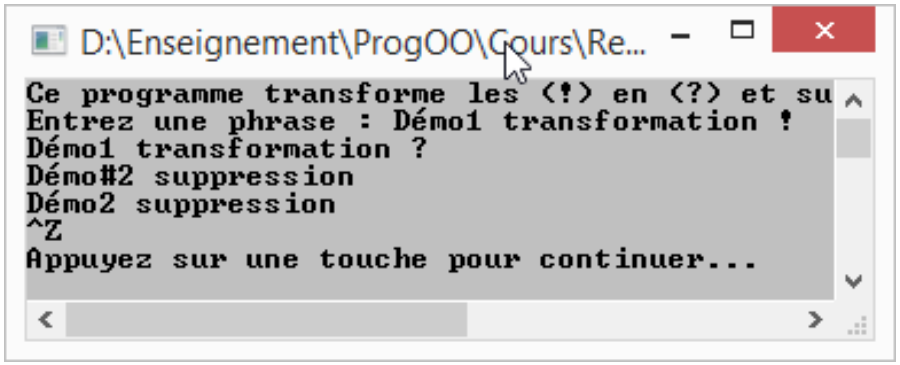
- **istream & putback(char ch);**
  - remet un caractère dans le flux d'entrée

```
cin.get(c).putback('a');
```

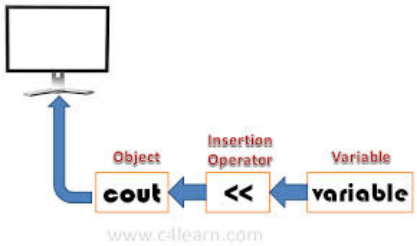
cin – peek() et putback()




```
int main(void)
{
   char ch;
   cout << "Programme transformant les (!) en (?) et supprime les (#)" << endl;
   cout << "Entrez une phrase : ";
   while (cin.get(ch))
   {
      if (ch == '!')
         cin.putback('?');
      else
         cout << ch;
      while (cin.peek() == '#')
         cin.ignore(1, '#');
   }
   return EXIT_SUCCESS;
}
```



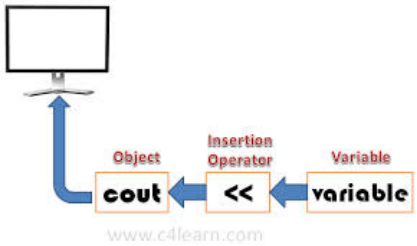
Quelques méthodes de cout



- Nous avons déjà vu au chapitre 2, un certain nombre de manipulateurs de flux de sortie. Voici quelques méthodes

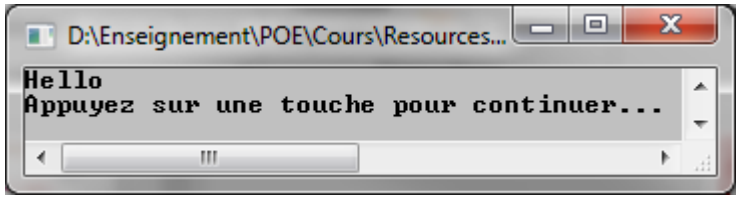
- Ces méthodes permettent de:
  - Écrire un caractère unique
  - Ecrire des chaînes

cout – put()

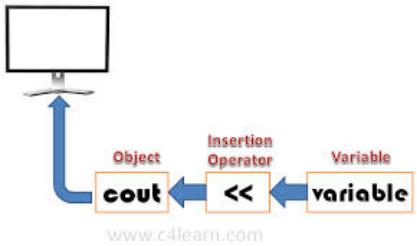


- **ostream & put(char ch);**
  - Ecrit un caractère unique vers le flux de sortie

```
int main(void)
{
    cout.put('H').put('e').put('l').put('l').put('o').put('\n');
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

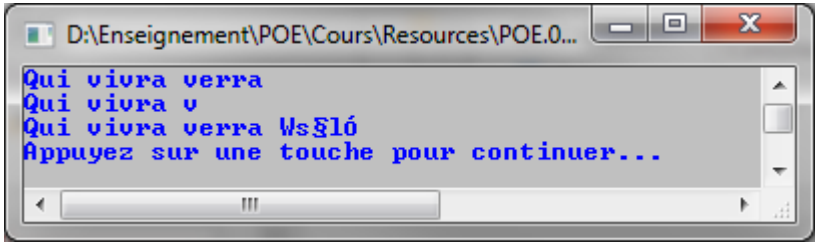


cout – write()

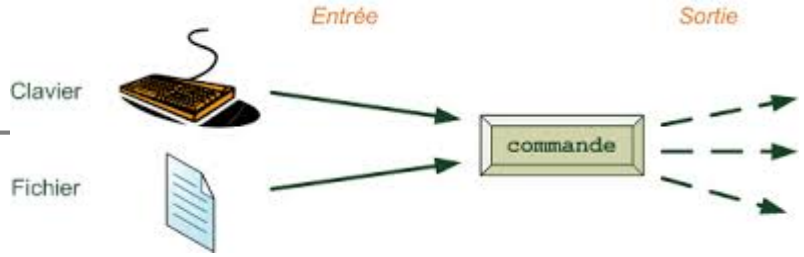


- **ostream & write(const char *texte, int taille);**
- fonctionne exactement comme l'opérateur <<, sauf qu'elle accepte un paramètre indiquant le nombre maximal de caractères à écrire

```
int main(void)
{
    char Une[] = "Qui vivra verra";
    int complet = strlen(Une);
    int tropCourt = complet - 4;
    int tropLong = complet + 6;

    cout.write(Une, complet) << endl;
    cout.write(Une, tropCourt) << endl;
    cout.write(Une, tropLong) << endl;
}
```

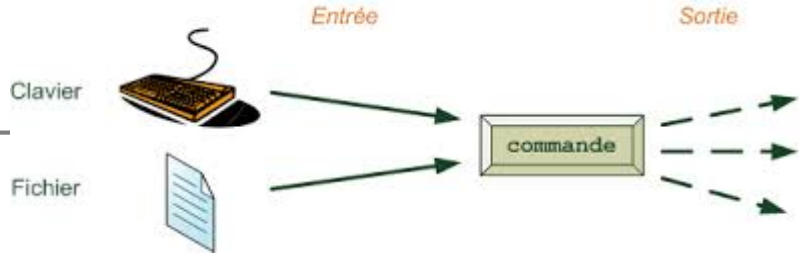


Flux – traitement des erreurs



- Les objets iostream gèrent des **indicateurs** qui précisent l'état de votre entrée et de votre sortie. Vous pouvez contrôler ces indicateurs à l'aide des fonctions booléennes eof(), bad(), fail() et good()
  - **eof()** renvoie la valeur true si l'objet iostream a rencontré EOF (Fin de fichier)
  - **bad()** renvoie true si vous tentez une opération non valide
  - **fail()** renvoie true dès que bad() renvoie true ou qu'une opération échoue
  - **good()** renvoie true lorsque les trois autres opérations renvoient false

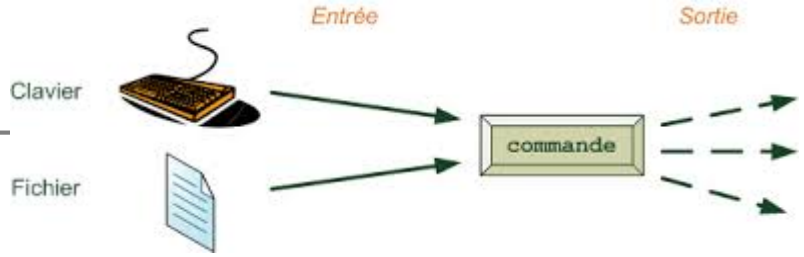
Flux – traitement des erreurs



- Les opérateurs " **()** " et " **!** " ont été redéfinis dans la classe "**ios**", pour faciliter la détection d'erreurs
  - **()** renvoie true s'il **n'y a PAS d'erreur** dans le traitement du flux
  - **!** renvoie true **s'il y a une erreur** dans le traitement du flux

heig-vd

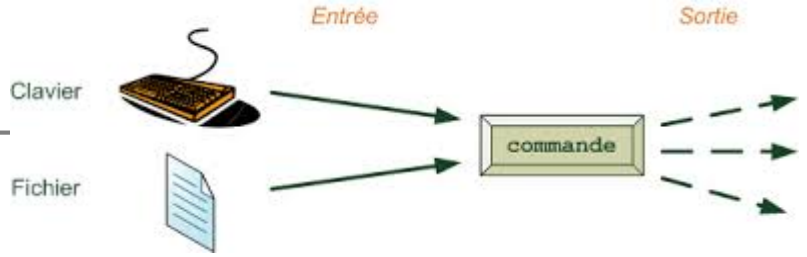
Flux – traitement des erreurs



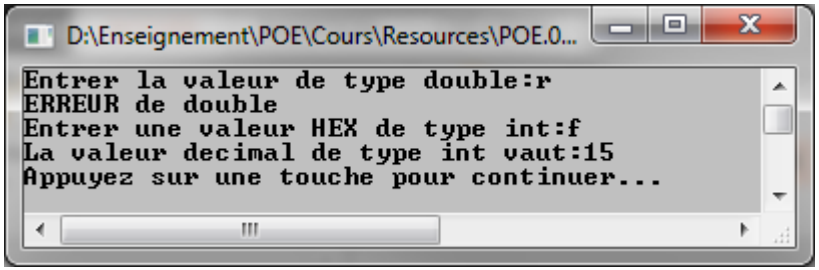
- Pour éviter les erreurs dans iostream, il faut:

1. Détecter l'existence de l'erreur
2. Vider le buffer
3. Effacer l'indication de l'erreur

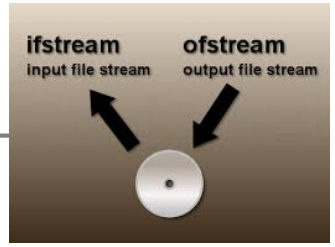
Flux – traitement des erreurs




```
int main(void)
{
   double d = 5.5;
   int i = 10;

   cout << "Entrer la valeur de type double:";
   cin >> d;
   if (!cin) // test si erreur d'entrée
   {
      cout << "ERREUR de double" << endl;
      cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
      cin.clear ();  // efface le bit d'erreur
   }
   else
      cout << "La valeur de type double vaut:" << d << endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

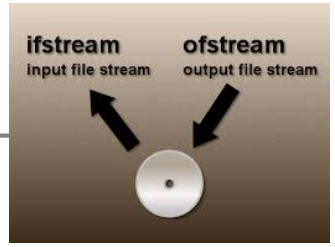


Gestion de fichiers



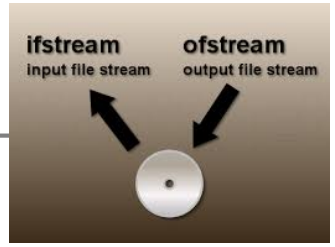
- **Les flux offrent un moyen uniforme** de traitement des données provenant du **clavier ou du disque dur**, ou allant vers **l'écran ou le disque dur**
- Tout ce qui a été vu dans ce chapitre va pouvoir être utilisé pour les fichiers
- Pour les fichiers, il faut inclure "**fstream**"

Gestion de fichiers – objet fichier



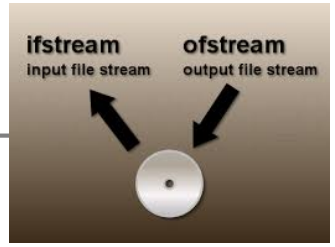
- "**fstream**" permet de **déclarer des objets de type « fichier »**
- Il y a 3 types de "stream":
  - **ifstream** in; // fichier en lecture seulement
  - **ofstream** out; // fichier en écriture seulement
  - **fstream** io; // fichier en lecture/écriture

Gestion de fichiers - open



- Après la déclaration, il faut ouvrir les fichier avec la méthode "open"
  - void ifstream::open(const char *filename, ios::openmode mode = ios::in);
  - void ofstream::open(const char *filename, ios::openmode mode = ios::out | ios::trunc);
  - void fstream::open(const char *filename, ios::openmode mode = ios::in | ios::out);
- Et choisir le mode
  - ios::in Ouverture en lecture (seulement avec ifstream)
  - ios:: out Ouverture en écriture (seulement avec ofstream)
  - ios::trunc Ecrase un fichier de même nom en écriture
  - ios::app Ouverture écriture séquentielle à la fin pour ajout (append)
  - ios::ate Ouverture écriture à accès direct à la fin pour ajout (append)
  - ios::binary Ouverture de fichier binaire

Gestion de fichiers – eof et close



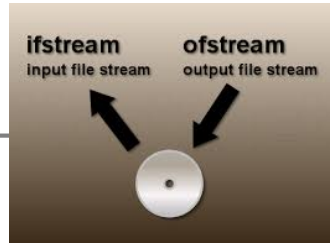
- Il faut tester si le fichier a été ouvert correctement.
  Si open () échoue, le flux sera à faux lorsqu'il est utilisé dans une expression booléenne

```
if(!mystream) {
   cout << "Le fichier ne peut pas être ouvert.\n";
   // traitement de l'erreur
}
```

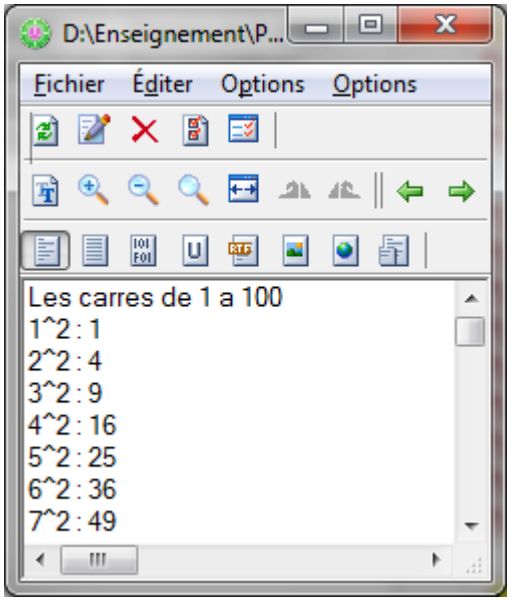
- La méthode "eof()" permet de savoir si l'on a atteint la fin d'un fichier
- Et bien sûr à la fin, il faut fermer le fichier
  avec la méthode "close()"

heig-vd

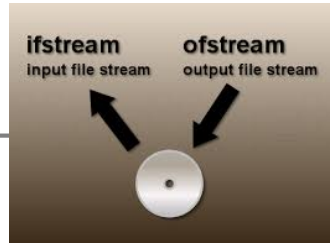
Gestion de fichiers texte - écriture




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

int main(void)
{
   // déclaration d'un objet flux pour le fichier
   ofstream fichier;
   int i;
   // ouverture du fichier
   fichier.open("carres.txt");
   // écriture dans le fichier
   fichier << "Les carres de 1 a 100" << endl;
   for (i = 1 ; i <= 100; i++)
      fichier << i << "^2 : " << i * i << endl;
   // fermeture du fichier
   fichier.close();
   system("PAUSE");
   return EXIT_SUCCESS;
}
```



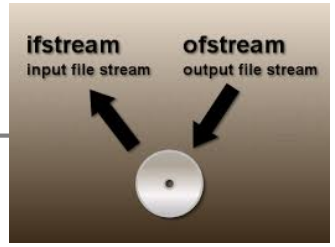
Gestion de fichiers texte – lecture (1/2)




```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main(void)
{
   // déclaration d'un objet flux pour le fichier
   ifstream fichier;
   string ligne;
   int nombre_ligne = 0;
   // ouverture du fichier
   fichier.open("carres.txt");
   if (!fichier) // test de l'ouverture du fichier
   {
      cout << "ERREUR: fichier pas ouvert corectement\n";
      system("PAUSE");
      return EXIT_FAILURE;
   }
```

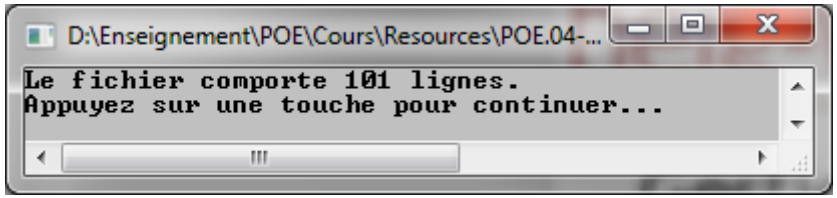
- Voir suite page suivante

Gestion de fichiers texte – lecture (2/2)

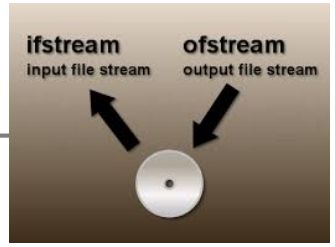


- Suite…

```
while (!fichier.eof())
{
   // lecture du fichier, vérification du succès
   if (getline(fichier, ligne))
         nombre_ligne ++;
}
 // fermeture du fichier
fichier.close();
cout << "Le fichier comporte " << nombre_ligne << " lignes." << endl;
system("PAUSE");
return EXIT_SUCCESS;
}
```

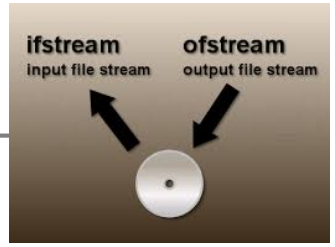


Gestion de fichiers binaire



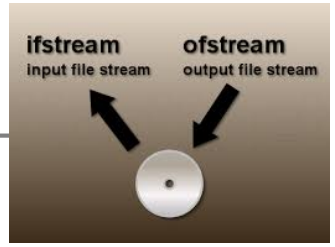
- Les méthodes "put()" et "get()" permettent d'écrire/lire des fichiers binaires:
  - istream &get(char &*ch*); // lit 1 caractère
  - ostream &put(char *ch*); // écrit 1 caractère
- Mais les méthodes à utiliser pour écrire et lire des blocs de données binaires sont: "read()" et "write()"
  - istream &read(char *buf, streamsize num);
  - ostream &write(const char *buf, streamsize num);

Gestion de fichiers binaire R/W (1/2)

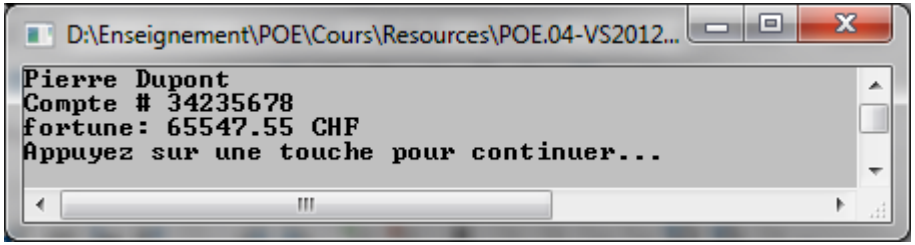



```
struct status {
   char name[80];
   double fortune;
   unsigned long num_compte;
};

int main(void)
{
   status acc;

   strcpy(acc.name, "Pierre Dupont");
   acc.fortune = 65547.55;
   acc.num_compte = 34235678;
   //écriture des données binaires (et ouverture à la déclaration)
   ofstream outbal("fortune", ios::out | ios::binary);
   if(!outbal) {
      cout << "ERREUR: impossible d'ouvrir le fichier en lecture\n";
      return 1;
   }
   outbal.write((char *) &acc, sizeof(status));
   outbal.close();
```

Gestion de fichiers binaire R/W (1/2)

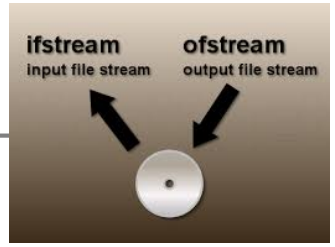



```
    // lecture des données (et ouverture à la déclaration)
    ifstream inbal("fortune", ios::in | ios::binary);
    if(!inbal) {
       cout << "ERREUR: impossible d'ouvrir le fichier en ecriture\n";
       return 1;
    }
    inbal.read((char *) &acc, sizeof(status));
    cout << acc.name << endl;
    cout << "Compte # " << acc.num_compte;
    cout.precision(2);
    cout.setf(ios::fixed);  // set format spécifique (hex, left, fixed, etc.)
    cout << endl << "fortune: " << acc.fortune << " CHF\n";
    inbal.close();
    system("PAUSE");
    return EXIT_SUCCESS;
}
```



heig-vd

Passer un fichier en paramètre
Par référence




```
int ModifyFile(ifstream &InputFile, ofstream &OutputFile)
{
    char StringFromFile[7];

    if (InputFile)
    {
        InputFile >> StringFromFile;
        InputFile.close();
    }

    if (OutputFile)
    {
        OutputFile << "Texte de sortie";
        OutputFile.close();
    }
    return 0;
}
```

```
int main(void)
{
    string FilePath;

    cout << "Entrer un nom de fichier: ";
    cin >> FilePath;

    ifstream InputFile(FilePath.c_str());
    ofstream OutputFile(FilePath.c_str());

    ModifyFile(InputFile, OutputFile);
    return 0;
}
```

heig-vd

Vos questions


Exercices
Flux et fichiers