ENTREES/SORTIES

Les fonctions d'entrée/sortie ne sont pas des fonctions du langage C++. Ces fonctions sont implantées en C++ et rangées dans la librairie standard de C++. Les prototypes des fonctions se trouvent dans les fichiers *iostream.h* ou *fstream.h*.Les fonctions d'entrée/sortie de C sont aussi dans la librairie standard, leurs prototypes se trouvant dans le fichier *conio.h*.

ENTREES/SORTIES C++

flux

Au niveau le plus bas, un fichier est considéré comme une suite de caractères. Cette suite de caractères est appelée **flux** ou **flot**

Au niveau utilisateur, le fichier est constitué d'une suite de données de types éventuellement différents.

Les opérations d'entrée/sortie sont réalisées par les classes *istream* et *ostream*, munies des opérateurs d'**extraction** >> et d'**injection** ou **insertion** <<.

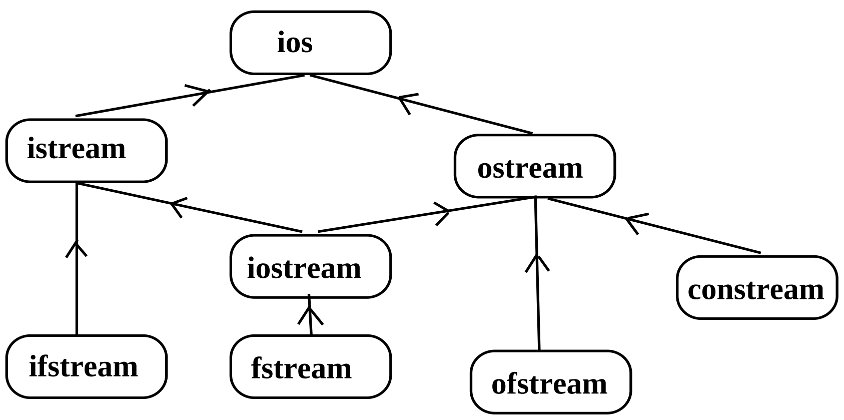
Quatre objets permettent au programmeur de manipuler des entrées/sorties au clavier/écran :

1. **cin** : objet de la classe **istream**, entrée standard au clavier.
2. **cout** : objet de la classe **ostream**, sortie standard à l'écran.
3. **cerr** : objet de la classe **ostream**,ortie standard des erreurs, associé à l'écran
4. **clog** : sortie standard des erreurs, tamponnée.

Trois classes permettent de manipuler les fichiers

1. **ifstream** : fichier en entrée.
2. **ofstream** : fichier en sortie.
3. **fstream** : fichiers en entrée/sortie.

La figure suivante représente une version extrêmement simplifiée de la hiérarchie des classes permettant la gestion des flux :



entrée

définition, ouverture, fermeture, fermeture de fichier

Un fichier peut être ouvert en même temps qu'il est défini.

ifstream x	définit un flux *x* non connecté à un fichier. Ce flux ne pourra être que lu. Il faudra procéder à une ouverture avant de lire le fichier.
ifstream x("nomFichier")	définit un flux *x* et le connecte au fichier de nom *nomFichier* du répertoire courant.
ifstream x("nomFichier", ios::in)	réalise les mêmes opérations que précédemment
x.open("nomFichier", ios::in)	connecte le flux *x* déclaré précédemment au fichier *nomFichier* du répertoire courant. *ios::in* est facultatif.
x.close()	déconnecte le flux *x* du fichier auquel il a été connecté par une ouverture.
x.fail()	a la valeur vrai (1) si l'opération précédente a échoué : echec à l'ouverture, ou à la lecture. *x.fail()* a la même valeur que *!x*.
x.good()	a la valeur vrai (1) si l'opération précédente a réussi .

lecture

Soit *x* un flux défini et ouvert en lecture :

x.eof()	a la valeur vrai si la fin de fichier est atteinte et faux sinon. La fin de fichier est atteinte quand le caractère de fin de fichier est lu.
x.get(c), c=x.get()	lit le caractère suivant dans le flux *x* et l'affecte à *c*. Le caractère suivant peut être la marque de fin de fichier, et dans ce cas *c* vaut -1 ou *c* vaut *EOF*.
x.peek(c)	lit le caractère suivant dans le flux, et le laisse dans le flux. Le caractère lu vaut *EOF* si c'est la fin de fichier.
x.seekg(n, pos= ios::beg)	*n* est un entier long : on se positionne sur le $n^{ième}$ caractère du fichier. Le premier caractère est le caractère de rang 0. Le second paramètre est *ios::beg* par défaut, il peut être égal à *ios::cur* et on se positionne *n* caractères après (ou avant si n est négatif) le caractère courant, ou *ios::end* et on se positionne *n* caractères après (ou avant si n est négatif) le dernier caractère.
x.tellg()	a pour valeur le rang de la position courante dans le flux.
x.getline(char * a, int b, char c='\n')	lit une suite de caractères et les range dans la chaîne *a*. La lecture n'échoue que si aucun caractère n'a pu être lu. Un zéro terminal est ajouté en fin de chaîne de caractères *a*. L'arrêt de la lecture se produit sur l'une des trois conditions suivantes : ♦ rencontre du délimiteur (le $3^{ème}$ paramètre), ce caractère n'est pas rangé dans la chaîne, mais il est lu. ♦ après la lecture de **b-1** caractères ♦ rencontre de la fin du fichier
x.get(char * a, int b, char c='\n')	est identique à getline mais le délimiteur n'est **pas lu.**
x.read(char * a, int b)	lit *b* caractères dans le fichier *x* et les affecte à la chaîne *a*. Si le caractère *EOF* est rencontré, l'opération lit autant de caractères que possible, et échoue : *x.fail()* est donc vrai. Dans ce cas **x.gcount()** a pour valeur le

nombre de caractères lus. *read* n'ajoute pas de zéro terminal à la chaîne *a*.

x.ignore(int n=1, int delim=EOF) ignore *n* caractères, ou s'arrête dès la rencontre du délimiteur.

exemples :

1. affichage d'un fichier à l'écran en utilisant *read* :

```
#include <fstream.h>

void main ()
{      const int long_ligne = 15;
       ifstream f;
       char *ligne;
       char c;
       ligne = new char [long_ligne+1]; // +1 pour le 0 terminal
       f.open("fichier0.cpp",ios::in);
       if (f.fail()) cout<<"erreur à l\'ouverture";

       do
       {      f.read(ligne,long_ligne);
              ligne[f.gcount()]=0;  // caracatère de code 0
              cout<<ligne;
       }while (!f.eof());
       delete ligne;
}
```

2. affichage d'un fichier à l'écran en utilisant *get* :

```
#include <fstream.h>

void main ()
{      const int long_ligne = 15;
       ifstream f;
       char *ligne;
       char c;
       ligne = new char [long_ligne+1]; // +1 pour le 0 terminal
       f.open("fichier0.cpp",ios::in);
       if (f.fail()) cout<<"erreur à l\'ouverture";

       do
       {      f.get(ligne,long_ligne,EOF);
              cout<<ligne;
       }while (!f.eof());
       delete ligne;
}
```

3. copie d'un fichier à l'écran en utilisant le *get* d'un caractère :

```
#include <iostream.h>
#include <fstream.h>

void main()
{      char c;
       ifstream x("essaif.cpp");
       if (!x.fail())
              while( (c=x.get())!=EOF ) cout<<c;
}
```

ou

```
#include <iostream.h>
#include <fstream.h>

void main()
{      char c;
```

```
        ifstream x("essaif.cpp");
        if (!x.fail()){
                c=x.get() ;
                while( !x.eof()){
                        cout<<c;
                        c=x.get() ;
                }
        }
}
```

opérateur >>

L'opérateur >> peut également servir à faire des lectures dans un fichier, et réalise des lectures en fonction du type des variables qui sont le second opérande de l'expression.

Attention : la priorité de l'opérateur >> et << est supérieure à la priorité de l'opérateur > par exemple, et donc *cout<<13>12*; affiche *13* et a pour valeur *cout>12*.

1. **int, long, unsigned** ignore les espaces, convertit une valeur entière tant que c'est possible, et s'arrête sur le premier caractère ne pouvant pas entrer dans la constitution d'un entier. *fail()* devient vrai en fin de fichier ou si on ne peut décoder une valeur entière.
2. **float, double, long double** même chose que précédemment
3. **chaîne de caractères** ignore les espaces, puis lit tous les caractères jusqu'à l'espace (ou retour) suivant. Il faut faire attention à ne pas déborder de l'espace réservé :

```
char chaîne[taille];

entree>>chaine; // risques de débordement
entree.width(taille-1);entree>>chaine; // pas de risques
entree>>setw(taille-1)>>chaine;     // arrêt à taille-1 caractères
```

La première tentative de lecture après avoir atteint la fin de fichier, retourne une chaîne de caractères de longueur 0.

4. **char** lit le premier caractère suivant qui est différent de espace. La fonction *fail()* devient vraie à la première tentative de lecture après avoir atteint la fin de fichier.

Le caractère **^Z** est le caractère de **fin de fichier au clavier**, et doit être rencontré comme premier caractère de la donnée à lire.

sortie

définition, ouverture, fermeture d'un fichier

Un fichier peut être ouvert en même temps qu'il est défini.

ofstream x	définit un flux *x* non connecté à un fichier. Ce flux ne pourra être que écrit. Il faudra procéder à une ouverture avant d'écrire dans le fichier.
ofstream x("nomFichier")	définit un flux *x* et le connecte au fichier de nom *nomFichier* du répertoire courant.
ofstream x("nomFichier", ios::out)	réalise les mêmes opérations que précédemment.
x.open("nomFichier", ios::out)	connecte le flux *x* déclaré précédemment au fichier *nomFichier* du répertoire courant. *ios::out* est facultatif.
x.close()	déconnecte le flux *x* du fichier auquel il a été connecté par une ouverture.
x.fail()	a la valeur vrai (1) si l'opération précédente a échoué : echec à l'ouverture, ou à la lecture. *x.fail()* a la même valeur que *!x*.

écriture

x.put(c)	range un caractère à la suite dans le flux *x*.
x.write(char * a, int b)	injecte *b* caractères qui se trouvent dans la suite de caractères adressés par *a* dans le flux *x*.
x.seekp(n, pos=ios::beg)	idem que pour les fichiers en entrée.
x.tellp()	idem que pour les fichiers en entrée.

exemples :

1. Recopie d'un fichier dans un autre.

```
#include <fstream.h>

void main ()
{      ifstream f; ofstream s;
       const int long_ligne=15;char *ligne;
       char c;
       ligne = new char [long_ligne+1];
       f.open("toto.txt",ios::in);
       if (f.fail()) cout<<"erreur à l\'ouverture";
       s.open("toto",ios::out);
       if (s.fail()) cout<<"erreur à l\'ouverture";

       do
       {      f.read(ligne,long_ligne);
              s.write(ligne,f.gcount());
       }while (!f.eof());
       delete ligne;
}
```

opérateur <<, manipulateur

L'opérateur << sert à injecter des valeurs dans un fichier, ou à l'écran. On peut utiliser les manipulateurs pour formatter ces sorties. Les manipulateurs sont définis dans *iomanip.h*.

- **dec, hex, oct** les prochaines sorties d'entier se feront en décimal, en hexadécimal, ou en octal.
- **setbase(n)** n=8, 10 ou 16 : les prochaines sorties d'entier se feront dans la base indiquée;
- **setw(n)** définit la largeur du champ en sortie : si le champ est trop petit, il est élargi pour effectuer la sortie.
- **endl** insère une fin de ligne.
- **setfill(c)** définit le caractère *c* comme caractère de remplissage, à la place des blancs.
- **setprecision(n)** définit la précision en en sortie à *n* chiffres.
- **setiosflags(long b)** met les flags à *b*;
- **resetiosflags(long b)** met à 0 les flags qui sont à 1 dans *b*.

Les flags sont les suivants :

skipws	saute les blancs.
left, rigth	alignemlent à gauche ou à droite.
dec, oct, hex	base
showbase	montre la base avecun 0 devant en octal et 0x en hexadécimal
showpoint	montre le point pour les flottants
showpos	montre le + pour les nombres positifs.

uppercase chiffres en majuscule pour les sorties héxadécimales.
scientific notation scientifique pour les réels.
fixed point décimal fixe pour les réels.

Appeler la fonction membre **cout.setf(ios::x)** permet de faire la même chose que précédemment.

exemple :

```
#include <iostream.h>
#include <iomanip.h>

void main()
{
  cout<<hex<<setw(5)<<setfill('*')<<12<<13;
}
```

Affiche le résultat *****cd*

```
#include <iostream.h>
#include <iomanip.h>

void main()
{ cout.setf(ios::left);                       | ou cout<<setiosflags(ios::left)
  cout<<hex<<setw(5)<<setfill('*')<<12<<13;   |          << .....
}
```

affiche la sortie **c****d**

On peut afficher un booléen :

```
cout<<true ;             // affiche 1
cout<<boolalpha<<true ; // affiche true
```

entrée/sortie

Un objet de la classe *fstream* peut être ouvert en entrée, ou en sortie ou en entrée sortie. Le choix du mode d'ouverture se fait à l'aide du deuxième paramètre de la définition ou de l'ouverture. Ce deuxième paramètre paut être :

- **ios::in** le fichier est ouvert en lecture/ecriture, positionné au début. Les écritures se font par dessus l'existant.
- **ios::out** le fichier est ouvert en écriture, positionné au début. Les écritures se font par dessus l'existant.
- **ios::ate** le fichier est ouvert en lecture/écriture, positionné à la fin , et évidemment pas effacé.
- **ios::app** le fichier est ouvert en lecture/écriture, positionné à la fin , et évidemment pas effacé.
- **ios::trunc** le fichier est ouvert en lecture/écriture et tronqué le fichier s'il existe
- **ios::binary** ouverture du fichier en binaire (voir plus loin).

Les méthodes de *fstream* sont l'union des méthodes de *ifstream* et *ofstream*.

remarque :

les constantes *ios::in*, *ios::out*, ... ont des valeurs qui sont des puissances de 2 : 1, 2, 4, 8, ... L'expression *ios::in|ios::app* réalise le *ou bit à bit* de 1 et 8 qui vaut 9 et est donc équivalent à 1+8, dans la mesure où les nombres sont des puissances de 2, mais le ou bit à bit est plus rapide à évaluer que le plus.

fichiers binaires

Les fichiers étudiés jusqu'à présent ne peuvent contenir que des chaînes de caractères. Si on veut écrire une séquence de longs entiers dans un fichier : 12345 1254360 30465, on aura besoin de 19 caractères. En revanche la représentation en binaire de ces trois entiers longs ne nécessitera que 3× 4=12 caractères. Pour ranger des valeurs entier long dans un fichier sans qu'elles soient décodées en chaîne de caractères, il faut considérer les quatre caractères de l'entier long comme quatre caractères et non comme un entier long. Il est donc nécessaire de surcharger les opérateurs << et >> pour traiter les opérations d'entrée sortie dans un fichier binaire :

```
#include <fstream.h>
#include <conio.h>

//       surcharge de << et >>

fstream& operator<<(fstream &f, long &n)
{  f.write((char*)&n, sizeof n                    );return f; }

fstream& operator>>(fstream &f, long &n)
{  f.read((char*)&n, sizeof n);return f; }

//       programme de test

void main ()
{ fstream f;
  char c; long n;
  clrscr();
       f.open("fichier3",ios::out|ios::binary);
         if (f.fail()) {cout<<"erreur à l\'ouverture";cin.get();}

       // écriture de 0, 1, 2 et 3 en binaire dans un fichier
       for (n=0;n<=3;++n)
       {  f<<n; // ou plus simplement f.write((char*)&n, sizeof n)
       };
       f.close();

       // lecture d'un fichier binaire de longs entiers
       // et affichage à l'écran
       f.open("fichier3",ios::in|ios::binary);
       do
       {  f>>n; // ou plus simplement f.read((char*)&n, sizeof n)
          if(f.fail()) break;
          cout<< n<<" ";
       } while(1);
       f.close();
}
```

Un fichier de caractères est terminé par le caractère 0. Il y a échec à la lecture du caractère de code 0. Un fichier binaire est également terminé par un caractère de code 0. Un fichier binaire doit à tout prix être spécifié binaire : en effet s'il n'est pas spécifié binaire, le système de lecture saute les caractères espaces (de code 32_d) ou les caractères retour (de code 13_d), et il y a décalage dans la lecture des données. Si le fichier est spécifié binaire il n'y a plus d'interprétation.

Au lieu d'utiliser un transtypage pour la surcharge des opérateurs << et >> on peut utiliser une union :

```
union chaine_ou_long{ char c[4]; long i;} ;
fstream& operator<<(fstream &f, long &n)
{ chaine_ou_long x; x.i=n;
  f.write(x.c, sizeof x.c);return f; }
```

```
fstream& operator>>(fstream &f, long &n)
{ chaine_ou_long x;
  f.read(x.c, sizeof x.c);n=x.i;
  return f; }
```

Si on veut pouvoir continuer à utiliser le type *fstream* normalement avec ses deux opérateurs << et >>, le mieux est de définir un nouveau type *fbin*, pour lequel on surdéfinira les opérateurs << et >>, et *fstream* garde les siens.

```
class fbin : public fstream
{
  public:
   fbin& fbin::operator<<( long &n)
    {write((char*)&n,sizeof(n));return *this;}
   fbin& fbin::operator>>( long &n)
    {read((char*)&n,sizeof(n));return *this; }
};

void main ()
{ fbin f;
  char c; long n;
  clrscr();
       f.open("fichier3",ios::out|ios::binary);
         if (f.fail()) {cout<<"erreur à l\'ouverture";cin.get();}

       // écriture de 0, 1, 2 et 3 en binaire dans un fichier
       for (n=0;n<=3;++n) {  f<<n;};
       f.close();

       // lecture d'un fichier binaire de longs entiers
       // et affichage à l'écran
       f.open("fichier3",ios::in|ios::binary);
       do
       {  f>>n;
          if(f.fail()) break;
          cout<< n<<" ";
       } while(1);
       f.close();
}
```

exercices

1. Ecrire un programme C++ qui compte le nombre d’occurrences de chaque voyelle, ainsi que le nombre total de caractères contenus dans un fichier.

2. Ecrire un programme C++ qui concatène 3 fichiers f_1, f_2, f_3 dans un fichier f_r.

3. Ecrire un programme qui copie un fichier dans un autre en supprimant les fins de ligne.

4. Ecrire un programme qui recopie à l'envers un fichier dans un autre.

```
#include <fstream.h>
#include <conio.h>
void main ()
{      fstream f1, f2;
       char c; long i, n;
       f1.open("essai.dat",ios::out);
       f1.write("a\nbb\nccc\n", 9);
       f1.close();
       f1.open("essai.dat",ios::in);
       f2.open("essai1.dat",ios::out);
       f1.seekg(-1,ios::end);// positionné sur le dernier caractère
       n=f1.tellg();
       for(i=n; i>=0; --i){f1.seekg(i,ios::beg);f1.get(c);f2.put(c);}
       f1.close();
       f2.close();
```

```
}
```

5. Soit la classe :

```
class etudiant
{ private :  char * nom;
                   int numero;
  public :   etudiant(); etudiant( etudiant & ); etudiant(char * n, int n);
                   ~etudiant();
                   etudiant& operator = (etudiant & );
                   friend fstream & operator <<(fstream & f, etudiant & e);
                   friend fstream & operator >>(fstream & f, etudiant & e);
};
```

- Définir les opérateurs << et >>.
- Nous désirons pouvoir accéder au i$^{\text{ème}}$ élément d'un fichier, à l'aide de l'opérateur [ ]. Par exemple, si f est un fichier contenant des étudiants, nous désirons pouvoir écrire f[i] pour obtenir une référence à une copie du i$^{\text{ème}}$ élément du fichier. Ecrire la définition de l'opérateur [ ] comme fonction membre de la classe etudiant.

Les fichiers en C

définition

Il faut inclure le fichier *stdio.h*

```
FILE * f ;
```

ouverture/fermeture

```
f=fopen(nomfichier, mode)
```

le retour est NULL si l'ouverture ne s'est pas faite.
le nom du fichier est un nom avec éventuellement le chemin.
le mode est un chaîne de caractères :

" r " lecture seule
" w " création en écriture, le fichier est écrasé s'il existe.
" a" écriture à la fin, création si le fichier n'existe pas.
"r+ " lecture/écriture (mise à jour). Le fichier n'est pas écrasé s'il existe.
" w+ " lecture/écriture (mise à jour). Le fichier est écrasé s'il existe.
" a+ " lecture/écriture (mise à jour). ouverture en fin de fichier.

On peut ajouter **t** ou **b** à la fin du mode pour spécifier que c'est du texte ou du binaire.

remarque : quand un fichier est ouvert pour la mise à jour, on peut faire des lectures ou des écritures sur le même fichier, cependant une lecture ne peut pas être faite après une ecriture sans avoir exécuté auparavant un *fseek* ou un *rewind*. De même une écriture ne peut pas suivre une entrée à moins d'un *fseek* ou d'une lecture qui a rencontré la marque de fin de fichier.

```
fclose(f)
```

lecture/écriture

unsigned int fread (adresse, taille_bloc, nb_bloc, fichier)	lit dans le fichier *nb_blocs* de taille *taille_bloc* et les range à l'adresse *adresse*. Retourne 0 si ça s'est mal passé et sinon le nombre de blocs lus.
unsigned int fwrite(adresse, taille_bloc, nb_bloc, fichier)	écriture . Retourne le nombres de blocs écrits.
bool feof(fichier)	vrai si la fin de fichier a été atteinte et faux sinon.
int fseek(FILE *stream, long pos, int a_partir_de);	positionne le curseur de fichier à *pos*. a*_partir_de* vaut 0 pour début, 1 pour courant et 2 pour fin. Retourne 0 si ça a marché et non 0 sinon.
void rewind(FILE *stream);	se repositionne en début de fichier
long int ftell(FILE *stream);	renvoie la position courante dans le fichier.
int fscanf(fichier, format, arguments), **int fprintf(fichier, format, arguments)**	lit le fichier ou écrit dans le fichier en respectant le format (voir *printf*). La lecture renvoie EOF si la fin du fichier est atteinte. *fprintf* renvoie le nombre d'octets écrits ou EOF s'il y a eut une erreur.
int fgetc(fichier)	retourne le caractère lu ou EOF
int fputc(caractère, fichier)	retourne le caractère écrit ou EOF si echec

char *fgets (caractères, n, fichier) lit n-1 caractère au maximum, renvoie un pointeur sur la chaîne lue, s’arrête sur fin de ligne qui est rangé dans la chaîne de caractères, ou sur fin de fichier, le zéro terminal est ajouté.

int fputs(caractères, fichier) écriture d’une chaîne de caractères. Retourne EOF si echec.

exemple :

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>

void main()
{
      FILE * f;
      char c;
      f=fopen(”  ... ”,  ”r”);         // affichage d’un fichier à l’écran
      while(!feof(f))
      {
            fread(&c,1,1,f);
            if(c==10) {  putch(‘\n’)         // il faut écrire le LF;
                         putch(‘\r’);}       // et le CR
            else putch(c);
      }
      fclose(f);

      int i=123;
      fopen(” ... ”, ” w”);
      fwrite(&i, 2, 1, f);      //écriture en binaire de l’entier 123 sur 2 octets.
}
```

Remarque : on peut obtenir un « handle » du fichier avec la fonction *fileno(FILE *)*, et définir une variable de type *fstream f(”handle de fichier”),* puis utiliser les fonction de *fstream* ; mais le curseur dans le fichier *FILE* * n’avance pas ...

LA LIBRAIRIE STDIO.H

- **int printf(const char * chaine {, arg1, ...})** écrit une chaîne formattée à l'écran : à chaque argument est appliqué une *spécification de format* contenue dans la chaîne. Il doit y avoir le même nombre de spécifications de format que d'arguments. Un format est constitué de la façon suivante :

  %[indicateur] [largeur] [.précision] [F|N|h|l] type

  [indicateur] **-** justifié à gauche, sinon à droite
  + présence du signe + si nécessaire
  ' ' absence du signe +
  #o ajoute des 0 à gauche à un argument différent de 0.
  #x (X) ajoute 0x ou 0X.
  #e (E) le résultat contient toujours un point.
  [largeur] **n** caractères sont imprimés.
  0n des zéros à gauche.
  * la largeur est founie par l'argument correspondant dans la liste.
  [.précision] **.0** précision par défaut.
  .n n chiffres après le **.**
  .* la précision est donnée par un argument de la liste.
  [F|N|h|l] modificateur : F pointeur far, N pointeur near, h short int et l long pour un int et long double pour un double
  type **d, i** entier signé
  o octal
  u entier non signé
  x, X héxadécimal
  f, e, g flottant **f=--.--, e=--.--e--, g** : l'un ou l'autre suivant les valeurs.
  E, G flottant avec **E**.

  exemple :

```
#include <stdio.h>
void main()
{
    printf("%-5d%5d\n", 12, 13);
    printf("%#x %#o %d\n", 12, 12, 12);
    printf("%f %.1f %e\n", 12.0, 12.0, 12.0);
    printf("%#10x %#20o %5d\n", 12, 12, 12);
    printf("%-+#10x %+#20o %5d\n", 12, 12, 12);
}
```

  Affiche :

```
12      13
0xc 014 12
12.000000 12.0 1.200000e+01
       0xc                  014    12
0xc                         014    12
```

- **scanf(char * format, liste d'arguments adresses)** Les arguments adresses sont des adresses de variables (&...)
  Le format a le même sens que pour printf excepté :
  1. l'indicateur * indique que la valeur lue n'est pas prise en compte : sa valeur est perdue.

2. la largeur indique le nombre maximum de caractères lus : on peut en lire moins si onn rencontre un séparateur (espace, tabulation ou linefeed).
3. les caractères ordinaires dans le format exigent la présence du même caractère dans le dispositif de lecture.
4.

sprintf

int sprintf(char *buffer, const char *format[, argument, ...]);

Fabrique le buffer qui contient la chaîne formattée comme pour un affichage. Si ça marche renvoie le nombre d’octets fabriqués (sans le zéro terminal), et retourne EOF s’il se produit une erreur.

LA LIBRAIRIE CONIO.H

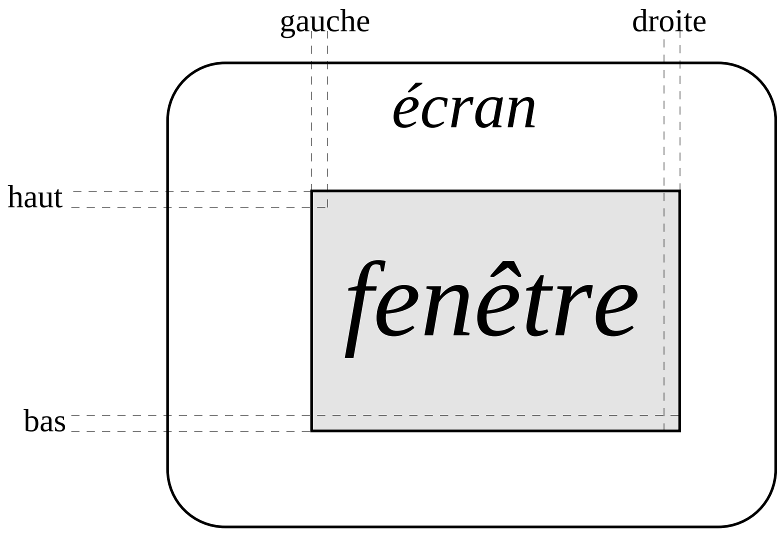
La librairie *conio.h* fournit des primitives de lecture/écriture de texte au clavier/écran dans l'environnement windows.

fenêtre

Un écran peut être découpé en **fenêtre**s, constituées de lignes elles mêmes constituées de caractères. Une fenêtre est une zone rectangulaire de l'écran. Quand un programme écrit à l'écran l'affichage se fait dans la fenêtre active; le reste de l'écran n'est pas affecté par les affichages.

L'ouverture d'une fenêtre se fait à l'aide de la fonction :

void window (int gauche, int haut, int droit, int bas) qui réalise l'ouverture de la fenêtre suivante :



Si les coordonnées ne sont pas valables, *window* ne fait rien. Les coordonnées de la fenêtre sont des coordonnées absolues par rapport à l'écran.

manipulation texte ou écran

- **void clrscr()** efface le contenu de la fenêtre courante.
- **void clreol()** efface le contenu de la ligne à partir du curseur, jusqu'à la fin de la ligne.
- **void delline()** détruit la ligne où est placé le curseur.
- **void insline()** insère une nouvelle ligne à l'endroit où est placé le curseur.
- **void gotoxy(x, y)** le curseur va se placer sur le caractère de coordonnées (x, y) dans la fenêtre courante.
- **int wherex()** a pour valeur la coordonnée en x du curseur.
- **int wherey()** a pour valeur la coordonnée en y du curseur.

écriture de texte

- **int cputs (const char * chaine**) écrit dansla fenêtre courante la chaîne *chaine* (à zéro terminal). Le caractère *\n* est un caractère de passage à la ligne sans retour en début de ligne. le caractère *\r* retourne en début de ligne. *cputs* a pour valeur le dernier caractère affiché.
- **int putch (int c)** écrit le caractère c sur l'écran, dans la fenêtre courante.

lecture

- **int getch()** a pour valeur un caractère lu au clavier sans écho à l'écran.
- **int getche()** a pour valeur un caractère lu au clavier avec écho à l'écran.
- **int kbhit()** a pour valeur vrai si une touche du clavier a été pressée.
- **char * cgets(char * chaine)** *cgets* lit des caractères au clavier jusqu'à l'occurrence de CR+LF, ou jusqu'au nombre maximal de caractères, qui doit se trouver dans *chaine[0]*.
  1. Le couple CR+LF est remplacé par 0 dans la chaîne.
  2. C*haine[1]* contient le nombre de caractères effectivement lus.
  3. *cgets* retourne un pointeur sur *chaine[2]*.

manipulation de blocs de texte

- **int movetext(int g, int h, int d, int b, int dg, int dh)** fait une recopie du rectangle (g, h, d, b) en (dg, dh). Les coordonnées peuvent se chevaucher. *movetext* renvoie 0 si les coordonnées sont en dehors de l'écran.
- **int gettext(int g, int h, int d, int b, void * dest)** sauvegarde dans *dest* le texte contenu à l'écran dans le rectangle spécifié. retourne 1 *gettext* si ça s'est bien passé. La taille du tampon doit être égale à C×L×2 caractères.
- **int puttext(int g, int h, int d, int b, void * source)** sauvegarde dans *dest* le texte contenu à l'écran dans le rectangle spécifié. *puttext* retourne 1 si ça s'est bien passé.

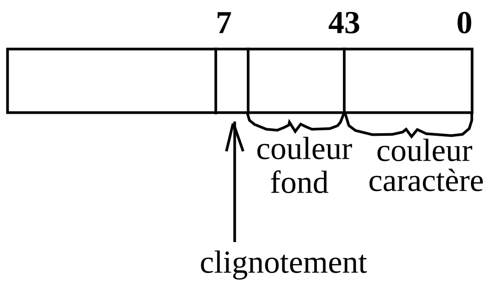
contrôle des attributs d'affichage

Les couleurs sont définies comme un type énumération :

enum couleur{BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LIGHTGRAY, DARKGRAY, LIGHTBLUE, LIGHTGREEN, LIGHTCYAN, LIGHTRED, LIGHTMAGENTA, YELLOW, WHITE};

La couleur de fond d'un caractère est comprise entre 0 et 7, et le caractère BLINK fait clignoter le caractère.

- **void textcolor (int couleur)** sélectionne la couleur d'affichage des caractères.
- **void textbackground (int couleur)** sélectionne la couleur du fond.
- **void textattr(int attribut)** un seul appel permet de modifier la couleur du caractère, et la couleur du fond. Les informations sont codées sur l'octet de poids faible de l'attribut :



*textattr(GREEN+16*RED+BLIK)* : caractères verts sur fond rouge et clignotants

- **void highvideo()** caractères en surbrillance.

- **void lowvideo()** caractères en sous-brillance.
- **void normvideo()** caractère normaux.

les aspects du curseur et autres fonctions

- **void _setcursortype(int curseur)** change l'aspect du curseur en fonction de la variable curseur :
  _NOCURSOR pas de curseur.
  _SOLIDCURSOR le curseur est un bloc.
  _NORMALCURSOR curseur normal.
- **void textmode(int mode)** modifie le mode texte en fonction de la variable mode :
  LASTMODE mode précédent.
  BW40 noir et blanc 40 colonnes.
  BW80 noir et blanc 80 colonnes.
  C40 16 couleurs et 40 colonnes.
  C80 16 couleurs et 80 colonnes.
  MONO monochrome 80 colonnes.
  C4350 EGA(80×43) VGA(80×50)
- **directvideo** variable initialisée à 1 dans ce cas Borlandc se charge des écritures : le matériel doit être 100% compatible ; si la variable est à 0 les écritures se font via le BIOS, et c'est un peu plus lent.
- **void gettextinfo(struct textinfo * r)** range dans la structure pointée par *r* les informations concernant le mode texte :

```
struct text_info{
             unsigned char
                    winleft,
                    wintop,             //description de la fenêtre
                    winright,
                    winbottom,
                    attribute,          //attribut du texte
                    normattr,           // attribut normal
                    curmode,            // BW40, BW80, ...
                    screenheight,// hauteur
                    screenwidth, // et largeur de l'écran
                    curx,               // coordonnées
                    cury;               // du curseur
}
```