Entées-Sorties (suite)

Manipulation externes de fichiers

- Suppression.
- Renommage.
- Création de fichier temporaire.
- Stdio fournit des fonctions pour faciliter la programmation.

Suppression de fichiers

- On peut supprimer un fichier (si on a les droits).
- Fonction `remove`
- Signature:
```
int remove(const char * nom_fichier) ;
```
- Renvoie 0 si Ok, -1 si erreur.
- Erreur consultable avec `perror`

Renommage de fichier

- On peut renommer un fichier
- Fonction `rename`.
- Signature:
```
int rename(const char * old, const char * new) ;
```
- Renvoie 0 si Ok, -1 sinon.

Création de fichier temporaire

- On peut créer un fichier temporaire pour stocker des résultats.
- On a possibilité de le supprimer ensuite.
- On a deux possibilités:
  - Le créer explicitement (`fopen`)et le supprimer ensuite (`remove`)
  - Le créer implicitement, il est détruit quand on ferme le flux.

Création de fichier temporaire : `tmpname`

- Pour la première solution on doit créer un fichier.
- On doit s'assurer que le nom de fichier qu'on va utiliser est unique.
- Fonction `tmpname`.
- Signature :
  ```
  char * tmpname(char * str) ;
  ```
- Renvoie un pointeur sur la chaîne de caractères. `NULL` si problème .
- Si `str` est `NULL`, la fonction renvoie un pointeur vers un buffer interne qui sera libéré au prochaine appel.

Création de fichier temporaire : `tmpfile`

- Fonction `tmpfile`
- Crée un `FILE *`, le met sur le système de fichiers. Le supprime une fois qu'on fait `fclose`
- Signature :
  `FILE * tmpfile(void) ;`
- Renvoie le pointeur sur la structure de fichier. `NULL` si erreur.

Fonctions `sscanf, sprintf,s*`

`*scanf`

- Les fonctions `*scanf` sont utiles car permettent de convertir des chaînes de caractères en des valeurs de types entiers, flottants…
- Deux fonctions vues :
  - `scanf` pour le terminal.
  - `fscanf` pour la lecture depuis un fichier.

Lecture depuis une chaîne de caractères

- La fonction sscanf permet cela
- La signature de la fonction est similaire à frpintf
- Signature :
  `int sscanf(const char * str, ''format'', …) ;`
- Renvoie le nombre d'affectations correctement effectuée.
- N'avance pas dans la chaîne comme `scanf` ou `fscanf`.

Fonction `sprintf`

- La fonction `sprintf` est l'analogue de `printf` et `fprintf` pour les chaînes de caractères.
- Permet de faire des écritures formatées dans un chaîne de caractères.
- Signature :
  ```
  int sprintf(char *str,const char *format,…);
  ```
- La chaîne dans laquelle écrire doit être assez grande.
- Renvoie le nombre d'octets écrits (sans compter `'\0'`)
- Un nombre négatif si problème.

Fonctions Variadiques

Fonctions variadiques

- Une fonction variadique est une fonction avec un nombre non borné de paramètres.
- Exemple le plus connu : `printf`
  - Seul le premier paramètre est obligatoire.
- Le langage C est un des rares à proposer cette fonctionnalité
- L'utilisation de telles fonction est **dangereuse**.

L’ellipse

- Une fonction variadique doit comporter au moins un vrai paramètres.
- On déclare ensuite le nombre de paramètres variables avec l’ellipse : ‘...’
- Ce doit être le dernier *« paramètre »* de la fonction.

L’ellipse

- Exemple :

  On veut faire une fonction max qui prend un nombre arbitraire de paramètres entier:

- Signature :

```
int max(int nombre, … ) ;
```

- Exemple d’utilisation

```
int maximum= max(4,1,2,256,9) ;
```

Librairie `stdarg`

- Il faut ensuite parvenir à récupérer la liste des entiers passés en paramètres.
- Cela est possible grâce à `stdarg.h` qui fournit un ensemble de macro qui vont permettre la manipulation et le parcours de la liste.
- Il n’y a pas de vérification de types à proprement parler.

`stdarg` : type de données

- La liste des variables est stockées dans une `va_list` (va pour *value argument*).
- Plusieurs macros sont fournies pour manipuler cette liste :
  - `va_start` pour démarrer le parcours de la liste.
  - `va_arg` pour récupérer une valeur de la liste.
  - `va_end` pour fermer la procédure.

va_start

- `va_start(va_list list, last)`
- Le premier paramètre est la liste.
- Le second paramètre est le dernier vrai paramètre dans la déclaration de la fonction.
- Dans l'exemple `max` ce serait `nombre`.
- Elle permet l'initialisation de la liste et ajoute du code C avec des accolades.

va_arg

- Une fois la liste initialisée on peut extraire les éléments les uns après les autres.
- `type va_arg(va_list list, type)`
- Premier paramètre : la liste.
- Second paramètre : le type attendu (p. ex `int`).
- Exemple :

```
int k=va_arg(list, int) ;
```

`va_arg`

- Chaque appel à `va_arg` avance dans la liste.
- On doit extraire tous les éléments de la liste.
- On doit donc connaître le nombre de paramètres attendu → d'où l'intérêt d'avoir au moins un vrai paramètre.
- Pour `printf`, ce sont les ‘ %’ qui déterminent le nombre attendu.

`va_close`

- Une fois la liste parcourue intégralement. On doit fermer les accolades créées par `va_start`.
- `va_close(va_list)`
- L’ouverture, le traitement et la fermeture doit tout être fait depuis la fonction.
  - On ne peut pas déléguer le traitement.

Exemple max

```
int max(int nombre,...){
int resultat, i ; int m=0 ;
va_ list ap ;
va_start(ap,nombre) ;
for(i=0;i<nombre;i++){
  resultat=va_arg(ap,int) ;
  if(resultat>m){m=resultat;}
}
va_end(ap) ;
return m ;
}
```

Fonctions utiles de `stdlib.h`

Fonction `exit`

- La fonction permet de quitter le programme dès qu'elle est invoquée.
- Signature :
  `void exit(int status) ;`
- La fonction est appelée pour une sortie « normale »

  → ferme tous les fichiers ouverts.
- Permet de « renvoyer » un entier
- La valeur est consultable depuis le terminal avec la variable `$?`

Exemple

Code :

```
void f(){
//
exit(1) ;
//
}

void main(int argc, char
** argv){
//...
f() ;
//...
}
```

Bash

```
> ./exe
> echo $?
```

`abort`

- Même rôle que exit mais en cas **d’erreur grave**.
- Ferme les flux ouverts.
- Envoie un signal particulier à l’appelant.
- Provoque un « `Core dump` »
  - Fichier binaire qui permet le débogage.

system

- La fonction `system` permet d'invoquer une commande du système.

```
int r=system(''ls -l'') ;
```

- Appelle la commande `ls` avec l'option `-l`
  le résultat est affiché dans le terminal.
- Renvoie O si Ok, -1 si erreur, diff de 0 si problème.