Entrée-sortie simple

Entrée-sortie

L'objectif des instructions d'entrée-sortie est d'assurer la communication du programme avec les unités périphériques. Ces opérations se réalisent à l'aide des fichiers fig.3.1). On dit que le programme lit le données depuis un fichier et écrit les résultats dans un autre fichier Le plus souvent quand les données sont préparées manuellement et les résultats sont destinées pour être lus, ils doivent être présentés en forme textuelle et les fichiers correspondants doivent être du type texte. Le fichier-texte contient une suite de caractères qui peuvent être des lettres, des chiffres, des signes de ponctuation et des caractères non imprimables comme la nouvelle ligne, le signe de tabulation et c. Mais pas toutes les données dans le programme sont caractères. C'est pourquoi on a besoin de conversion entre la présentation externe des données (texte) et la présentation interne.

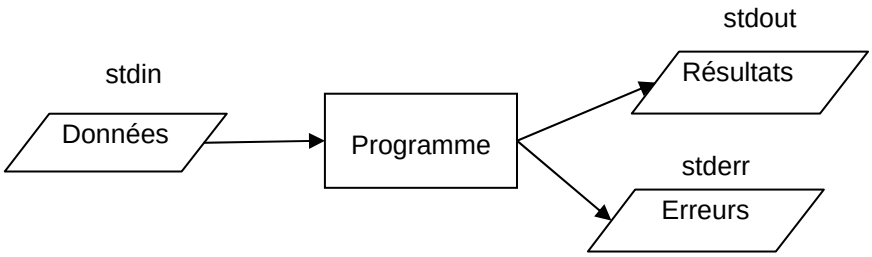

Fig.3.1 Entrée-sortie

Avant d'être utilisé chaque fichier doit être ouvert rt après l'usage il doit être fermé. L'ouverture prépare le fichier pour lecture ou écriture En C il y a trois fichiers qui sont ouverts avec le début du programme. Ce sont les fichiers standards stdin (le clavier), stdout (l'écran) et stderr (pour afficher les erreurs). Le clavier et l'écran permettent que l'utilisateur puisse tapez les données et voir les résultats.

Pour utiliser les instructions d'entrée-sortie on doit inclure la bibliothèque standard d'entrée-sortie stdio.h et son fichier en-tête

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

Ecrire sur l'écran

Ecriture des caractères

Fonction putchar

Utilisation

La fonction putchar admet un paramètre de type int

```
putchar (param )
```

Sémantique des paramètres

- param est une expression de type entier, présentant le caractère à afficher.

Valeur rendue

La fonction putchar retourne la valeur de son argument. En cas d'erreur elle rend la valeur EOF.

Exemples

```
putchar( 'A' );
putchar('\n');
putchar('\'');
```

Fonction puts

Utilisation

La fonction puts admet un paramètre de type char* (string).

```
puts (param )
```

Sémantique des paramètres

- param est une expression de type char*, présentant la chaîne de caractères à afficher.

Valeur rendue

La fonction puts retourne une valeur ≥ 0 en cas de succès et EOF dans cas d'erreur.

Exemples

```
puts( "Zorro est arrivé");
```

Ecriture formatée : printf

Utilisation

La fonction fprintf admet un nombre variable de paramètres. Son utilisation est la suivante :

```
printf (format , param1 , param2 , ... , paramn )
```

Sémantique des paramètres

- format est une chaîne de caractères qui spécifie ce qui doit être écrit.
- $param_i$ est une expression délivrant une valeur à écrire.

Valeur rendue

La fonction printf retourne le nombre de caractères écrits, ou une valeur négative si il y a eu une erreur d'entrée-sortie.

Présentation

La chaîne format contient des caractères ordinaires (c'est à dire diférents du caractère %) qui doivent être copiés tels quels, et des spécifications (introduites par le caractère %), décrivant la manière dont doivent être écrits les paramètres $param_1$, $param_2$,... $param_n$.

Si il y a moins de $param_i$ que n'en réclame le format, le comportement n'est pas défini. Si il y a davantage de $param_i$ que n'en nécessite le format, les $param_i$ en excès sont évalués, mais leur valeur est ignorée.

Les spécifications

Une spécification se compose des éléments suivants :

1. 1 Un certain nombre (éventuellement zéro) d'indicateurs pouvant être les caractères suivants :
   - **–** $param_i$ sera cadré à gauche dans son champ d'impression.
   - **+** si $param_i$ est un nombre signé, il sera imprimé préecédé du signe + ou –.
   - **espace** si $param_i$ est un nombre signé et si son premier caractère n'est pas un signe, on imprimera un blanc devant $param_i$. Si on a à la fois l'indicateur + et l'indicateur blanc, ce dernier sera ignoré.
   - # cet indicateur demande l'impression de $param_i$ sous une forme non standard. Pour le format o, cet indicateur force la précision à être augmentée de manière à ce que $param_i$ soit imprimé en commençant par un zéro. Pour les formats *x* et X, cet indicateur a pour but de faire précéder $param_i$ respectivement de 0x ou 0X, sauf si $param_i$ est nul. Pour les formats e, E, f, g et G, cet indicateur force $param_i$ à être imprimé avec un point décimal, même si il n'y a aucun chiffre après. Pour les formats g et G, cet indicateur empêche les zéros de la fin d'être enlevés. Pour les autres formats, le comportement n'est pas défini.
   - **0** pour les formats d, i, o, u, x, X, e, E, f, g, et G cet indicateur a pour effet de compléter l'écriture de $param_i$ avec des 0 en tête, de manière à ce qu'il remplisse tout son champ d'impression. Si il y a à la fois les deux indicateurs 0 et –, l'indicateur 0 sera ignoré. Pour les formats d, i, o, u, x et X, si il y a une indication de précision, l'indicateur 0 est ignoré. Pour les autres formats, le comportement n'est pas défini.

2. Un nombre entier décimal (optionnel) indiquant la taille minimum du champ d'impression, exprimée en caractères. Si $param_i$ s'écrit sur un nombre de caractères inférieur à cette taille, $param_i$ est complété à droite ou à gauche (selon que l'on aura utilisé ou pas l'indicateur -), avec des blancs ou des 0, comme il a été expliqué plus haut.

3. Une indication (optionnelle) de précision, qui donne :

   - le nombre minimum de chiffres pour les formats d, i, o, u, x et X.
   - le nombre de chiffres après le point décimal pour les formats e, E et f.
   - le nombre maximum de chiffres significatifs pour les formats g et G
   - le nombre maximum de caractères pour le format s.
     - Cette indication de précision prend la forme d'un point (.) suivi d'un nombre décimal optionnel. Si ce nombre est omis, la précision est prise égale à 0.
     - Remarque
     - Le nombre entier décimal indiquant la taille maximum du champ d'impression et/ou le nombre entier décimal indiquant la précision, peuvent être remplacées par le caractère *. Si le caractère * a été utilisé une seule fois dans le format, la valeur
     - correspondante (taille du champ ou précision) sera prise égale à $param_{i-1}$. Si le caractère * a été utilisé deux fois, la taille du champ d'impression sera égale à $param_{i-2}$, et la précision sera égale à $param_{i-1}$. Si la taille maximum du champ d'impression a une valeur négative, cela a la sémantique de l'indicateur – suivi de la valeur (positive) de la taille du champ d'impression. Si la précision est négative, cela a la sémantique d'une précision absente.

4. un caractère optionnel, qui peut être :

   - **h** – s'appliquant aux formats d, i, o, u, x ou X : $param_i$ sera interprété comme un short int ou unsigned short int.
   - **h** – s'appliquant au format n : $param_i$ sera interprété comme un pointeur vers un short int.
   - **l –** s'appliquant aux formats d, i, o, u, x ou X : $param_i$ sera interprété comme un long int ou un unsigned long int.
   - **l –** s'appliquant au format n : $param_i$ sera interprété comme un pointeur vers un long int.
   - **L –** s'appliquant aux formats e, E, f, g ou G : $param_i$ sera interprété comme un long double.

     (a) Si un h, l ou L s'applique à un autre format que ceux indiqués ci-dessus, le comportement est indéterminé.

5. un caractère qui peut prendre les valeurs suivantes :

   - **d, i –** $param_i$ sera interprété comme un int, et écrit sous forme décimale signée.
   - **u –** $param_i$ sera interprété comme un unsigned int, et écrit sous forme décimale non signée.
   - **o –** $param_i$ sera interprété comme un int, et écrit sous forme octale non signée.
   - **x, X –** $param_i$ sera interprété comme un int, et écrit sous forme hexadécimale non signée. La notation hexadécimale utilisera les lettres abcdef dans le cas du format x, et les lettres ABCDEF dans le cas du format X.
     - Dans les cas qui précédent, la précision indique le nombre minimum de chiffres avec lesquels écrire $param_i$. Si $param_i$ s'écrit avec moins de chiffres, il sera complété avec des zéros. La précision par défaut est 1. Une valeur nulle demandée avec une précision nulle, ne sera pas imprimée.
   - **c** $param_i$ sera interprété comme un unsigned char.
   - **s** $param_i$ sera interprété comme l'adresse d'un tableau de caractères (terminé ou non par un null). Les caractères du tableau seront imprimés jusqu'au premier des deux évènements possibles :
     - impression de précision caractères de $param_i$.
     - rencontre de null dans $param_i$.

       Dans le cas où $param_i$ n'est pas terminé par un null, le format d'impression doit comporter une indication de précision.
   - **p –** $param_i$ sera interprété comme un pointeur vers void. Le pointeur sera imprimé sous une forme dépendante de l'implémentation.
   - **n –** $param_i$ sera interprété comme un pointeur vers un int auquel sera affecté le nombre de caractères écrits jusqu'alors par cette invocation de printf.
   - **e,E –** $param_i$ sera interprété comme un double et écrit sous la forme :

- ***[-] pe.pf e signe exposant***
- dans laquelle **pe** et **pf** sont respectivement partie entière et partie fractionnaire de la mantisse. La partie entiére est exprimée avec un seul chiffre, la partie fractionnaire est exprimée avec un nombre de chiffres égal à la précision. La précision est prise égale à 6 par défaut. Si la précision est 0, le point décimal n'apparaît pas. L'exposant contient toujours au moins deux chiffres. Si $param_i$ est nul, l'exposant sera nul. Dans le cas du format **E**, la lettre E est imprimée à la place de e.

- **f –** $param_i$ sera interprété comme un double et écrit sous la forme :
  - ***[-] pe.pf***
  - dans laquelle **pe** et **pf** sont respectivement partie entiére et partie fractionnaire de la mantisse. La partie fractionnaire est exprimée avec un nombre de chiffres égal à la précision. La précision est prise égale à 6 par défaut. Si la précision est 0, le point décimal n'apparaît pas.
- **g,G –** $param_i$ sera interprété comme un double et écrit sous le format f, ou le format e, selon sa valeur. Si $param_i$ a un exposant inférieur à -4 ou plus grand ou égal à la précision, il sera imprimé sous le format **e**, sinon il sera imprimé sous le format **f**. D'éventuels zéros à la fin de la partie fractionnaire sont enlevés. Le point décimal n'apparaît que si il est suivi d'au moins un chiffre. Dans ce qui précède, l'utilisation du format **G** implique l'utilisation du format **E** à la place du format **e**.
- **% –** Un caractère % est écrit.

Exemples

Le programme :

```
void main (void ) {
  int i = 23674;
  int j = -23674;
  long k = (1L << 31);
  double x = 1e-8 + 1000;
  char c = 'A' ;
  char *chaine = "chaine de caracteres" ;  printf ("impression de i : \n");
  printf ("|%d| \t |%u| \t |%o| \t |%x|" ,i,i,i,i) ;
  printf ("\n|%10d| \t |%10.6d| \t |%-10.3o| \t |%.5x|" ,i,i,i,i) ;
  printf ("\nimpression de j : \n");
  printf("%d \t %u \t %o \t %x",j,j,j,j);
  printf ("\nimpression de k: \n") ;
  printf ("%ld \t %lu \t %lo \t %lx", k, k, k,k) ;
  printf ("\nimpression de x: \n") ;
  printf ("|%f| \t |%e| \t |%g|",x, x ,x ) ;
  printf ("\n|%.2f| \t |%.2e|",x,x);
  printf ("\n|%+10.2f| \t |%+-10.2e|",x,x);
  printf ("\n|%.20f| \t |%.20e|" ,x,x) ;
  printf ("\nimpression de c : \n") ;
  printf ("%c \t %d",c, c ) ;
  printf ("\nimpression de chaine : \n");
  printf ("|%s| \t |%.10s|" , chaine, chaine) ;
  printf ("\n|%20.10s| \t |%-20.10s|" , chaine, chaine) ;
  printf ("\n");;
}
```

va afficher les résultats suivants (fig. 3.1)

```
impression de i :
|23674|          |23674|         |56172|         |5c7a|
|     23674|     |    023674|    |56172     |    |05c7a|
impression de j :
-23674   41862   121606          a386
impression de k:
-2147483648      2147483648      20000000000     80000000
impression de x:
|1000.000000|    |1.000000e+03|          |1000|
|1000.00|        |1.00e+03|
|  +1000.00|     |+1.00e+03 |
|1000.000000010000010000000|      |1.000000000010000010000e+03|
impression de c :
A        65
impression de chaine :
|chaine de caracteres|   |chaine de |
|          chaine de |   |chaine de           |
```

Fig.3.1 Affichage de l'exemple pour sortie

Lire le clavier

Tous ce qui est tapé au clavier est enregistré dans une zone de mémoire tampon et visualisé sur l'écran. Mais que la touche <Enter> n'a pas été enfoncée la ligne reste en cours de frappe et susceptible de corrections. Les instructions de lecture utilisent seulement les caractères nécessaires. Le premier caractère non déjà utilisé sera le premier pris par l'instruction suivante.

La touche <Enter> met au tampon une marque de fin de ligne, qui est équivalent à une espace pour les instructions de lecture.

La touche <Ctrl-Z> met au tampon une marque de fin d'entrée (fin du fichier). Quand on essaie de lire un caractère Toute tentative pour lire ensuite est une erreur.

Lecture de caractères

fonction *getchar*

Utilisation

```
getchar ()
```

Description

La fonction getchar lit un caractère du stdin.

Valeur rendue

Si la lecture se fait sans erreur et sans rencontre de la fin de fichier, getchar rend le

caractère lu. Si il y a erreur d'entrée-sortie, ou rencontre de fin de fichier, getchar rend la

valeur EOF. Pour cette raison, le type de la valeur rendue est int et non pas char.

Exemple

```
int C;
C = getchar();
```

Lecture formatée - fonction *scanf*

Utilisation

La fonction *scanf* admet un nombre variable de paramètres. Son utilisation est la suivante :

scanf *(format , param$_1$, param$_2$, ... , param$_n$)*

Sémantique des paramètres

- *format* est une chaîne de caractères qui spécifie la forme de l'entrée admissible.
- les $param_i$ sont des pointeurs. Ils pointent des variables dans lesquelles *scanf* dépose les valeurs lues du stdin (clavier), après les avoir converties en binaire.

Valeur rendue

Si au moins un $param_i$ s'est vu affecter une valeur, *scanf* retourne le nombre de $param_i$ affectés. Si il y a eu rencontre de fin de fichier ou erreur d'entrée-sortie avant toute affectation à un $param_i$, *scanf* retourne **EOF**.

Description

scanf lit une suite de caractères du fichier stdin (le clavier) en vérifiant que cette suite est conforme à la description qui en est faite dans format. Cette vérification s'accompagne d'un effet de bord qui consiste à affecter des valeurs aux variables pointées par les différents $param_i$.

Quelques définitions

flot d'entrée – il s'agit de la suite de caractères lus du fichier stdin.

caractères blancs – il s'agit des six caractères suivants : espace, tab, line feed, new line, vertical tab et form feed.

modèle – un modèle est la description d'un ensemble de chaînes de caractères. Exemple : %d est le modèle des chaînes formées de chiffres décimaux, éventuellement signées.

conforme – on dira qu'une chaîne est conforme à un modèle quand elle appartient à l'ensemble des chaînes décrites par un modèle. Exemple : 123 est conforme au modèle **%d**.

directive – une directive peut être :

- une suite de caractères blancs qui est une spécification d'un nombre quelconque de caractères blancs. Exemple : un espace est un modèle pour un nombre quelconque d'espaces, ou d'un nombre quelconque d'espaces et de tab mélangés, ou d'un nombre quelconque d'espaces, de tab et de line-feed mélangés etc.
- une suite de caractères ordinaires (c'est à dire qui ne sont ni des caractères blancs, ni le caractère %) qui est un modèle pour elle-même. Exemple : la chaîne ***hello*** est une spécification de la seule chaîne ***hello***.
- des séquences d'échappement introduites par le caractère **%**. Ces séquences jouent un double rôle : elle sont à la fois un modèle des chaînes acceptables dans le flot d'entrée, et elles sont également des ordres de conversion de la chaîne lue et d'affectation du résultat à une variable pointée par le $param_i$ correspondant. Exemple : la directive %d est un modèle des nombres décimaux et un ordre de conversion de la chaîne lue en valeur binaire et d'affectation à l'entier pointé par le $param_i$ correspondant.

Les séquences d'échappement (spécifications de conversion)

1. Obligatoirement, le caractère **%**.
2. Facultativement, le caractère ***** qui indique une suppression d'affectation : lire une donnée comme indiqué et l'oublier (c'est-à-dire ne pas le ranger dans une variable).
3. Facultativement, un nombre donnant la largeur maximum du champ (utile, par exemple, pour la lecture de nombres collés les uns aux autres).
4. Facultativement, une lettre qui apporte des informations complémentaires sur la nature de l'argument correspondant :
   - **h –** devant d, i, o, u, x : l'argument est l'adresse d'un short (et non pas l'adresse d'un int) ;
   - **l –** devant d, i, o, u, x : l'argument est l'adresse d'un long (et non pas l'adresse d'un int). Devant e, f, g : l'argument est l'adresse d'un double (et non pas d'un float) ;
   - **L –** devant e, f, g : l'argument est l'adresse d'un long double (et non pas d'un float).
5. Obligatoirement, un caractère de conversion parmi
   - **d** – Argument : int *. Donnée : nombre entier en notation décimale.
   - **i** – Argument : int *. Donnée : entier en notation décimale, octale (précédé de 0) ou hexadécimale (précédé de 0x ou 0X).
   - **o** – Argument : int *. Donnée : entier en octal (précédé ou non de 0).
   - **u** – Argument : unsigned int *. Donnée : entier en notation décimale.
   - **x** – Argument : int *. Donnée : entier en notation hexadécimale (précédé ou non de 0x ou 0X).

- **c** – Argument : char *. Donnée : autant de caractères que la largeur du champ l'indique (par défaut : 1). Ces caractères sont copiés dans l'espace dont l'adresse est donnée en argument. Les blancs et tabulations ne sont pas pris pour des séparateurs (ils sont copiés comme les autres caractères) et il n'y a pas de '\0' automatiquement ajouté à la fin de la chaîne.
- **s** – Argument : char *. Donnée : chaîne de caractères terminée par un caractère d'espacement (blanc, tabulation, fin de ligne) qui n'en fait pas partie. Un '\0' sera automatiquement ajouté après les caractères lus.
- **e** – Argument : float *. Donnée : nombre flottant, c'est-à-dire dans l'ordre : signe facultatif, suite de chiffres, point décimal et suite de chiffres facultatifs, exposant (e ou E, signe facultatif et suite de chiffres) facultatif.
- **f** – Comme e.
- **g** – Comme e.
- **n** – Argument : int *. Aucune lecture. Effet : l'argument correspondant doit être l'adresse d'une variable entière ; celle-ci reçoit pour valeur le nombre de caractères effectivement lus jusqu'à ce point par le présent appel de la fonction scanf.
- **[*caractère...caractère*]** – Argument : char *. Donnée : la plus longue suite faite de caractères appartenant à l'ensemblea indiqué entre crochets. Un caractère au moins sera lu. Un '\0' est ajouté à la fin de la chaîne lue.
- **[^*caractère...caractère*]** – Argument : char *. Donnée : comme ci-dessus mais les caractères permis sont maintenant ceux qui n'appartiennent pas à l'ensemble indiqué.
- **% –** Pas d'argument. Le caractère % doit se présenter sur le flot d'entrée.

Algorithme de *scanf*

La chaîne format doit se composer d'un ensemble de directives. Il doit y avoir autant de $param_i$ que de directives demandant l'affectation d'une valeur. Si il n'y a pas suffisamment de $param_i$ pour le format, le comportement n'est pas défini. Si il y a davantage de $param_i$ que demandé par le format, les $param_i$ en excès sont _evalués mais ils sont inutilisées.

La fonction *scanf* exécute dans l'ordre chaque directive du format. Si une directive échoue, la fonction *scanf* retourne à l'appelant.

- L'exécution d'une directive formée de caractères blancs, consiste à consommer dans le flot d'entrée la plus longue séquence possible de caractères blancs. Même si cette séquence est de taille nulle, la directive a réussi.
- L'exécution d'une directive formée de caractères ordinaires, consiste à consommer dans le flot d'entrée une séquence identique à la directive. Au premier caractère différent, la directive a échoué et ce caractère reste non lu.
- L'exécution d'une directive formée d'une séquence d'échappement, consiste à :
  - consommer dans le flot d'entrée la plus longue séquence possible de caractères blancs. Cette séquence peut être de taille nulle. Cette action ne s'applique pas aux formats **c, n,** ou **[caractères].**
  - consommer dans le flot d'entrée la plus longue séquence possible de caractères qui soit conforme au modèle. Si cette séquence est de taille nulle, la directive a échoué.
  - si la directive ne contient pas le caractère *****, convertir la chaîne lue et l'affecter à l'objet pointé par le $param_i$ correspondant. Si cet objet n'est pas de la taille ou du type convenable pour la recevoir, le comportement n'est pas défini.
- Remarques sur la gestion des espaces blancs
- La gestion des espaces blancs est assez pénible. Il y a deux méthodes de consommation des espaces blancs du flot de données :
  - le *format* contient une directive formée de caractères blancs. Si une telle directive ne peut consommer aucun caractère blanc, c'est un cas d'échec de la directive, mais pas un échec de l'instruction.
  - le *format* contient une séquence d'échappement (autre que **c, n,** ou **[caractères]**) dont l'exécution commence par consommer d'éventuels caractères blancs dans le flot de données. Si il n'y a pas de caractères blancs à consommer, ce n'est pas une condition d'échec de la directive.

Exemples

1. Si existe la déclaration

```
int JOUR,MOIS, ANNEE;
```

Instruction	Données valides	Données invalides
`scanf("%i %i %i", &JOUR,&MOIS,&ANNEE);`	`12 4 1980` `12  004  1980`	`12/4/1980`
`scanf("%i/%i/%i", &JOUR,&MOIS &ANNEE);`	`12/4/1980` `12/04/01980`	`12 4 1980` `12 /4 /1980`

Les résultats de l'instruction :

```
RECU = scanf("%i %i %i", &JOUR, &MOIS, &ANNEE);
```

seront

Données	RECU	JOUR	MOIS	ANNEE
12 4 1980	3	12	4	1980
12/4/1980	1	12	-	-
12.4 1980	1	12	-	-
12 4 19.80	3	12	4	19

2. Si existe la déclaration

```
int x,y;
```

Instruction	**Données valides**	**Données invalides**
scanf("(%d,%d)",&x, &y);	(22,33) (22, 33)	(22 , 33)
scanf(" (%d , %d)",&x, &y);	(22,33) (22, 33) (22 , 33)	(22 ; 33)

Instruction	données	z	x	y
z=scanf(“%2d%3d”,&x,&y);	34565	2	34	565
	2456	2	24	56
	234567	2	23	456
z=scanf(“%2d%d”, &x,&y);	234567	2	23	4567
z=scanf(“%2d %d”, &x,&y);	234567	2	23	4567