CRO TD2 :

simple I/O, function, variable range and parameter passing
(1 séance sur machine), 13 juillet 2023

En complément de ce sujet se trouvent sur Moodle, les transparents présentant les notions vues dans ce TD ainsi qu'un **document complémentaire** (`scanf.pdf`) précisant le comportement de `scanf` sur les caractères utile en fin de TD. Pensez à les télécharger et les parcourir avant de commencer.

1 Entrées/Sorties formatées : `printf` et `scanf`

L'affichage à l'écran et la lecture au clavier se font par les fonctions `printf` et `scanf`. Ces fonctions sont relativement mal conçues et peuvent provoquer une grande perte de temps si on ne les utilise pas correctement.

1.1 `printf`

`printf`

Lorsque l'on tape `man 3 printf` la réponse (le début) indique :
`int printf(const char *format, ...);`
C'est le **prototype** de la fonction `printf`, qui indique le type de ses argument et de son résultat. Les trois petits points en deuxième argument indiquent *un nombre variable d'arguments*. Le langage C propose un mécanisme pour définir des fonctions avec un nombre d'arguments variable. Le principe des entrées/sorties formatées en C est le suivant :

— Le premier argument est une chaîne de caractères qui contient un certain nombre de *formats*, par exemple `%d` pour un entier, `%c` pour un caractère.

— Les arguments suivants sont les valeurs qui vont être interprétées comme étant du type spécifié par les formats, dans l'ordre de la chaîne de caractère.

Une documentation complète des formats est disponible ici :

— `https://fr.cppreference.com/w/c/io/fscanf`

— `https://fr.cppreference.com/w/c/io/fprintf`

Par exemple, comment afficher la valeur 65 (`1000001` en binaire) ? Cela dépend de commet nous voulons *l'interpréter* : comme un caractère, comme un entier en octal etc. Si bien que l'instruction suivante :

```
printf("Ascii[%d]=%c, en octal: %o en hexa: %x\n",65,65,65,65);
```

donnera comme affichage :

```
Ascii[65]=A, en octal: 101 en hexa: 41
```

QUESTION 1 ► si vous ne comprenez pas le résultat de ce printf, appelez l'enseignant

QUESTION 2 ► Écrivez un programme `affichage.c` qui affiche, les carrés des 30 premiers entiers. Pour faire une boucle on utilisera la syntaxe suivante :

```
for(int i=0; i<30; i++)
{
  /* corps de la boucle */
}
```

Affichez les d'abord à la suite, puis un par ligne, puis alignés à droite (consultez les transparents sur Moodle disponible pour ça si besoin). Faites un Makefile pour compiler votre programme.

1.2 scanf

scanf

La syntaxe de la fonction `scanf` est :

```
scanf("chaîne de contrôle ",&expr1, ..., &exprn);
```

Ici, la `"chaîne de contrôle"` ne contient que les formats. `&expr1`, `&expr2`, etc. désignent les *adresses* des variables qui vont stocker les données lues au clavier (nous expliquerons ce mécanisme plus tard), il faut donc toujours qu'il y ait l'opérateur adresse (`&`) devant. L'absence de l'opérateur adresse entraîne systématiquement une erreur de segmentation.
Les données à entrer au clavier doivent être séparées par des blancs ou des <RETURN> sauf s'il s'agit de caractères (<RETURN> est un caractère).

Exemple d'utilisation de `scanf` :

```
#include <stdio.h>
int main()
{
  int i;
  printf("entrez un entier sous forme hexadecimale i = ");
  scanf("%x",&i);
  printf("i = %d\n",i);

  return 0;
}
```

Si on entre au clavier la valeur 1a, le programme affiche i = 26. Attention à bien donner l'*adresse* (opérateur `&`) de la variable `i` à affecter (i.e. `&i`).

QUESTION 3 ► Modifier votre programme précédent pour qu'il lise au clavier le nombre de carré qu'il doit afficher à l'écran. N'oubliez pas d'afficher une invite pour que l'utilisateur sache ce qu'il doit rentrer au clavier.

2 Les fonctions en C

Définition de fonctions en C

On peut en C découper un programme en plusieurs fonctions. Seule la fonction `main` est obligatoire. Même si vous ne définissez pas de fonction, vous utilisez les fonctions des bibliothèques C standard (`scanf, printf` par exemple)
En C toutes les fonctions sont *déclarées* au même niveau, il n'y a pas de déclaration de fonction *à l'intérieur d'autres fonctions*.
La déclaration de fonction suit la syntaxe suivante, illustrée par la fonction factorielle à droite.

```
type nom (type-1 arg-1,...,type-n arg-n)
{[déclarations de variables locales ]
  liste d'instructions
}
```

La première ligne `int factorielle(int n)` s'appelle l'*en-tête* (ou *prototype*, ou encore *signature*) de la fonction.

```
int factorielle(int n)
{
  int  i, fact=1;

  for (i = 1; i<=n; i++)
     fact *= i;
  return fact;
}
```

L'appel de la fonction se fait par exemple, dans une autre fonction, de la manière suivante :
`x=factorielle(10);`

Le mécanisme de passage de paramètre en C (on dit : passage de paramètre par valeur) doit être bien compris :

- Une fonction communique des valeurs avec son environnement appelant à travers les paramètres et le résultat.
- dans `int factorielle(int n)`, `n` est le *paramètre formel* de la fonction. Il peut être utilisé dans le corps de la fonction comme une variable locale.
- dans `x=factorielle(10)`, `10` est le *paramètre effectif* utilisé lors de cet appel.
- En C, tout se passe lors de cette appel comme si on exécutait le corps de la fonction *avec la case mémoire pour* `n` *contenant une copie de la case mémoire contenant 10*. On dit que les paramètres sont passés *par valeur*.
- Lorsque l'appel est terminé, la case mémoire de `n` disparaît.
- Donc : on peut modifier la valeur du paramètre formel `n` dans le corps de la fonction mais cela ne modifiera pas la valeur du paramètre effectif (10).

QUESTION 4 ► Recopiez la fonction `factorielle` ci-dessus dans un fichier, créez, à la suite de cette fonction, une fonction `main` qui appelle factorielle(10) et affiche le résultat à l'écran. Ajouter une règle a votre Makefile pour compiler votre programme.

Portée des variable

Les variables manipulées dans un programme C n'ont pas toutes la même durée de vie. On distingue deux catégories de variables.

- Les variables permanentes (ou statiques)
- Les variables temporaires

Chaque variable déclarée a une *portée* (ou durée de vie) qui est la portion de code dans laquelle elle est connue. On peut déclarer des variables au début de chaque bloc (un bloc est le début d'une fonction ou d'une portion de code entre accolade). La *portée* (ou durée de vie) de ces variables est limitée au bloc.
Les blocs sont forcément imbriqués lexicalement, lors d'une utilisation d'une variable n, elle peut avoir été déclarée deux fois dans la suite des blocs englobant. L'utilisation correspond à celle de la première définition rencontrée lorsque l'on remonte dans les blocs (i.e. la dernière effectuée temporellement).

QUESTION 5 ► Copiez le programme suivant dans un fichier `portee.c`, rajoutez une ligne au Makefile et compilez le, est ce que vous comprenez bien ce qu'il se passe ?

```
#include <stdio.h>

static int n=0;

int main()
{
  int n=-1;

  printf("au début, n vaut %d\n",n);

  for (int i=0; i<10; i++)
    {
      int n=10;

      n++;
      printf("à l'itération %d, n vaut %d\n",i,n);
    }

  printf("après la boucle, n vaut %d\n",n);

  return 0;
}
```

Évidement, on évitera de nommer deux variables avec le même nom dans une même fonction..

☞ **Validez avec un enseignant.**

3 Retour sur les types de base et entrées/sorties

Maintenant que vous avez vu les fonctions d'E/S. prenez connaissance du tableau de format ci dessous :

Tous les formats (`printf, scanf`)

format	type d'objet pointé	représentation de la donnée saisie
%d	int	décimale signée
%hd	short int	décimale signée
%ld	long int	décimale signée
%u	unsigned int	décimale non signée
%hu	unsigned short int	décimale non signée
%lu	unsigned long int	décimale non signée
%o	int	octale
%ho	short int	octale
%lo	long int	octale
%x	int	hexadécimale
%hx	short int	hexadécimale
%lx	long int	hexadécimale
%p	void *	pointeur : adresse en hexadecimal
%f	float flottante	virgule fixe
%lf	double	flottante virgule fixe
%Lf	long double	flottante virgule fixe
%e	float flottante	notation exponentielle
%le	double	flottante notation exponentielle
%Le	long double	flottante notation exponentielle
%g	float flottante	virgule fixe ou notation exponentielle
%lg	double	flottante virgule fixe ou notation exponentielle
%Lg	long double	flottante virgule fixe ou notation exponentielle
%c	char	caractère
%s	char*	chaîne de caractères

QUESTION 6 ► Écrivez un programme qui lit un flottant au clavier et qui l'affiche à l'écran.

QUESTION 7 ► Ajoutez à votre programme les instructions pour lire un caractère au clavier et l'afficher à l'écran. Pourquoi n'y arrivez vous pas ?

stdin, stdout, Le buffer d'entrée de scanf

Par défaut, un programme C lit ses entrées sur un descripteur de ficher particulier qui s'appelle `stdin` et écrit ses sorties sur `stdout` (et ses erreurs sur `stderr`. On verra comment on peut rediriger les E/S vers d'autres fichiers, mais par défaut, `stdin` c'est le clavier et `stdout` c'est l'écran (plus précisément : la console ou on exécute le programme exécutable)
`scanf` lit ses entrées dans un *buffer* (ou tampon en Français) qui est rempli lorsque l'on tape un retour chariot après avec écrit ce que l'on voulait sur l'entrée standard `stdin`. Le problème c'est que le caractère 'retour chariot' est aussi mis dans le buffer, ce qui ne pose pas de problème quand on lit des flottant, mais devient très très contre-intuitif quand on lit des caractères.
La solution est donc **de ne jamais utiliser** `scanf` **pour lire des caractères**. Utiliser les fonctions `getc` et `fgets` pour cela.

QUESTION 8 ► Lisez le document "scanf.pdf" présent sur Moodle, et implémentez la méthode qui consiste à vider le tampon d'entrée pour résoudre le problème de lire un caractère après avoir lu un flottant.
Notez bien que les caractères sont des types entier, affichez votre caractère avec le format %d, qu'est ce qui s'affiche ?

Les chaînes de caractères en C

En C une chaîne de caractères n'est pas un type de base : c'est un tableau de caractères terminé par le caractère spécial `'\0'` (ou NULL : octet valant 0).
Pour l'instant, nous allons la déclarer comme un tableau de taille supérieure à la chaîne que nous voulons y rentrer.
Par exemple, la déclaration :
```
char chaine[80];
```
permettra de manipuler toutes les chaînes de moins de 79 caractères.

QUESTION 9 ► Ajoutez à votre programme les instructions pour lire une *chaîne de caractère* et la rentrer dans un tableau de caractère de taille 80 en utilisant `fgets`, puis l'afficher à l'écran.

4 Pour aller plus loin : traitement de trames

L'objectif du programme que vous devez développer est le suivant : on veut découper des trames d'un fichier de trames réseaux en différents paquets de 8 octets. Un fichier initial contient des trames réseaux. Chaque ligne du fichier correspond à une trame (fin de trame = `'\n'` i.e. passage à la ligne). Une trame est une suite d'octet (garantie sans `'\0'`), on la représente par la suite de caractère en code ascii. Une trame valide doit avoir le format suivant :

— La trame contient un nombre variable d'octets. La valeur de ces octets est quelconque mais ne peut être ni `'\n'` ni `'\0'`. Dans tous les cas, la taille maximum d'une trame est de 64 octets (`TAILLE_MAX_TRAME`=64).

— Le dernier octet de la trame contient un code détecteur d'erreurs : le *checksum*. Ce code détecteur d'erreur contient le nombre d'octet de la trame modulo 16, il est représenté par le caractère hexadécimal correspondant à ce nombre. Par exemple pour une trame contenant 21 octets (sans le dernier octet –checksum–) doit avoir un checksum de $5 = 21 \; mod \; 16$.

Votre programme doit transmettre chaque trame au fichier destination, sous forme d'une suite de paquets. chaque paquet a une taille de 8 octets, comprenant 7 octets de donnée et un octet de code détecteur d'erreur. Les paquets devront être séparés par des `'\n'` dans les fichiers destination (i.e. passage à la ligne). Un paquet ne contient des informations que d'une seule trame. Les paquets incomplets doivent être complétés par des '?' (octets de bourrage). Le code détecteur d'erreur (i.e. le dernier octet du paquet) est le nombre d'octets venant de la trame (c'est à dire sans compter les '?' rajoutés). Par ailleurs, vous devrez créer un fichier des trames rejetées (trames invalides, i.e. CRC non valide).

fichier de trames

```
x1
xy2
hnJQgN\X`SZpn`d
rK2
_O`[YqOg[gA
ORinDp[E8
gmRD3
0
q1
N[SgZb[BVooeOe`RcI2
jQLmRcp^
VoTicnYKpoHmaD
OhPWLJN\8
nFGYgroc8
SOL^MHe7
If2
C1
Mheaq`fEG[A
K1
]fCHrKmGr9
1D
0
ITniR\]WcE[B
r]lf`BfjoljqfnrF
IjmDaM`LdPQLDD
aZqO\dFX^pWQC
AR^jLbYYPdJ`CUcb0
J_`eQfKocaM[C
l1
FS]^bOgOQJrAjD
```

trames rejetées

```
hhnJQgN\X`SZpn`
gmRD
jQLmRcp
1
```

fichier de paquets

```
x??????1
xy?????2
rK?????2
_O`[YqO7
g[g????3
ORinDp[7
E??????1
q??????1
N[SgZb[7
BVooeOe7
`RcI???4
VoTicnY7
KpoHma?6
OhPWLJN7
\??????1
nFGYgro7
c??????1
SOL^MHe7
If?????2
C??????1
Mheaq`f7
EG[????3
K??????1
]fCHrKm7
Gr?????2
ITniR\]7
WcE[???4
r]lf`Bf7
joljqfn7
r??????1
IjmDaM`7
LdPQLD?6
aZqO\dF7
X^pWQ??5
AR^jLbY7
YPdJ`CU7
cb?????2
J_`eQfK7
ocaM[??5
l??????1
FS]^bOg7
OQJrAj?6
FS]^bOg7
OQJrAj?6
```

FIGURE 1 – Exemple de traitement de trames

Par exemple la trame : `OhPWLJN\8` est une trame valide (8 caractères), elle donnera lieu aux deux paquets suivants :

```
OhPWLJN7
\??????1
```

La figure 1 montre un exemple de fichier de trame reçu avec les fichiers générés par le programme.