Un livre de Wikilivres.

Exercices en langage C

Préparation

La **préparation** consiste à installer un environnement de programmation suivant votre système d'exploitation.

En ligne de commande

Linux

Pour les distributions compatibles Debian (comme Ubuntu) :

```
# apt-get install build-essential
...
$ cc exox.c -o exox
```

Mac

Les systèmes d'exploitation Mac OS X fournissent en standard les compilateurs GCC. tapez la commande suivante pour vérifier :

gcc --version

donne par exemple :

```
i686-apple-darwin8-gcc-4.0.1 (GCC) 4.0.1 (Apple Computer, Inc. build 5367)
```

Si gcc n'est pas présent, installez-le depuis les disques systèmes fournis avec votre machine *Mac OS X Install Disc 1* dossier *Xcode Tools* ou téléchargez la dernière version depuis le site Apple[1] et choisissez *Tools Downloads* et télécharger (1Gb) puis installer la version adaptée à votre système (10.4 ou 10.5).

Microsoft Windows

Le compilateur C est fourni dans la suite libre GNU Compiler Collection portée sous Windows dans le paquet MinGW (Minimalist GNU for Windows) [2]. Vous pouvez alors utiliser le compilateur gcc dans une fenêtre MS-DOS à condition d'avoir ajouté le chemin du répertoire bin de votre installation MinGW dans la variable d'environnement utilisateur nommée `Path`.

Vous pouvez aussi utiliser un environnement de développement avec le confort du shell Linux : MSys [2]. Vous pouvez utiliser le compilateur gcc dans une fenêtre ouverte grâce au script msys.bat situé dans un sous répertoire bin d'installation de Msys. MSys créé aussi un raccourci sur votre bureau.

```
$ gcc hello.c -o hello
$ ./hello
```

Avec un environnement de développement intégré

Eclipse

Eclipse est un environnement de développement générique et extensible (site officiel http://www.eclipse.org). Initialement prévu pour développer en Java, il peut maintenant également gérer des projets développés avec d'autres langages de programmation tels que le C et le C++ grâce à l'ensemble de plugins CDT.

Le compilateur C/C++ n'est pas intégré au plugin. Si aucun compilateur C/C++ n'est installé, il faut en installer un.

Installation Eclipse

Vous pouvez télécharger directement une version d'Eclipse adaptée spécifiquement au développement C/C++ depuis la page de téléchargement du site Eclipse [3].

Vous pouvez aussi télécharger la plateforme Eclipse puis le plug-in CDT :

- Lancez Eclipse, puis dans le menu déroulant :`Help>Software Updates>Find and Install...`
- Cochez *Search for new features to install*, bouton *Next*. Bouton *New Remote Site...*, entrez l'adresse de téléchargement :

```
Name: CDT
URL: Voir l'url adaptée à votre version d'Eclipse sur la page de téléchargement de CDT [4]
```

- Bouton *Finish*, choisissez un miroir proche de vous et continuez l'installation.

Remarque pour Windows

Sous Windows, MinGW et Msys [5] sont aussi nécessaires car ils fournissent le compilateur gcc et les outils comme GNU Make.

Pour un lien automatique avec Eclipse, mettre dans une variable d'environnement utilisateur nommée *Path* le chemin du répertoire qui contient les binaires de MinGW et Msys.

Première utilisation d'Eclipse

- Créez un nouveau projet (*File > New > Project*) de type *Standard Make C Project* nommé "Exercices". Ensuite depuis l'arborescence de projet, rajoutez un nouveaux fichier source (*New > Source File*.
- Après avoir modifié votre source, dans le menu déroulant *Project > Build all*.
- L'exécution se lance depuis le menu déroulant *Run*.

Références et liens

1. Site Apple Xcode Tool : [1] (http://developer.apple.com/tools/xcode).
2. Le site du projet libre MinGW et MSys : [2] (http://www.mingw.org).
3. Page de téléchargement d'éclipse : Eclipse Downloads (http://www.eclipse.org/downloads/) [archive]
4. Page de téléchargement d'Eclipse / CDT : CDT Downloads Page (http://www.eclipse.org /cdt/downloads.php) [archive]
5. Page d'accueil des projets MinGW et Msys : MinGW - Home (http://www.mingw.org/) [archive]

Notions de base

Ces exercices sur les **notions de base** abordent :

- définition de fonction
- la fonction `main`
- l'appel de fonction
- la bibliothèques standard, l'inclusion avec le préprocesseur
- la sortie formatée avec `printf`
- les paramètres de `main`

Exercice 1 : programme minimal *nop*

Écrire un programme qui ne fait rien (*nop* signifie *no operation*, soit « aucune opération »).

Pour le compiler et l'exécuter :

```
> gcc -Wall -o nop.exe nop.c
> ./nop.exe
>
```

Remarque qualité :

- L'option **-Wall** n'est pas obligatoire. Elle demande au compilateur de signaler toutes les fautes qu'il peut

détecter. Cela produit quelquefois des *warnings* ou avertissement que le programmeur, désireux de produire un code de qualité, s'efforcera de comprendre la signification et de corriger.
- Je donne l'extension .exe à l'exécutable produit pour l'identifier plus facilement. Sous Windows l'extension .exe est obligatoire et rajoutée par l'éditeur de lien si oubli.

Notions :

- définition de fonction
- fonction principale `main` servant de point d'entrée
- valeur retournée par une fonction

Solution

Un programme C commence par l'exécution d'une fonction `main`. Il existe deux prototypes standards pour la fonction `main` : `int main(void)` et `int main(int argc, char* argv[])`. Nous utilisons le plus simple car nous n'avons pas besoin de `argc` ni `argv`. Remarquons que les deux prototypes indiquent que `main` retourne un entier de type `int`. Nous retournons 0 (zéro), valeur conventionnelle pour indiquer que le programme est terminé sans erreur. Voici notre **définition** de la fonction `main` :

```c
/* Solution en retournant 0 */
int main(void)
{
  return 0;
}
```

Nous pouvons aussi utiliser pour le code retour la constante **EXIT_SUCCESS** de *stdlib.h* plus parlante que 0.

```c
#include <stdlib.h>

int main(void)
{
  return EXIT_SUCCESS;
}
```

Exercice 2 : programme *false*

Dans les faits, le programme précédent retourne toujours le code de succès 0 ; il correspond donc à la commande Unix *true*. La commande *true* retourne systématiquement le code de succès 0.

```
MacMini-TM:~ thierry$ true
MacMini-TM:~ thierry$ echo $?
0
```

Pour écrire *faux* (équivalent du programme UNIX *false*), cet exercice introduit une complication : l'appel d'une fonction contenue dans le même fichier source que la fonction principale main.

Écrire la fonction *main()* d'un programme *faux* qui appelle la fonction un suivante et retourne sa valeur à la sortie :

La fonction a appeler :

```c
int un(void)
{
  return 1;
}
```

Pour compiler et exécuter :

```
> gcc -Wall -o faux.exe faux.c
> ./faux.exe
> echo $?
1
```

Notions :

- déclaration de fonction
- appel de fonction
- prototype de fonction

Solution

On définit la fonction `un` avant la fonction `main`. Dans `main`, à l'aide de l'opérateur d'appel de fonction `()`, on procède à l'appel de fonction `un()`, dont on retourne immédiatement la valeur.

```c
int un(void)
{
  return 1;
}

int main(void)
{
  return un();
}
```

Remarques qualité :

- Lorsque on définit une fonction de la sorte, elle est par défaut **extern** (visible de l'extérieur). Dans un projet plus important, elle pourrait donc être appelée à la place d'une autre portant le même nom. Pour limiter sa visibilité en interne, il faudrait utiliser le mot clé **static**. `static int un(void)`.
- Que se passe-t-il si l'on définit la fonction `un` après `main` ? En compilant `main`, le compilateur va remarquer que l'on appelle une fonction `un` dont il ne sait rien. Cela ne l'empêche pas de compiler le programme ! (**sans l'option -Wall il ne signalera rien !**) Le compilateur fait comme si `un` avait été déclarée par : `extern int un()`. Il y a déclaration implicite de fonction, alors que le compilateur ne sait rien de l'implémentation réelle de la fonction ! Par chance, la déclaration explicite correspond au prototype `int un(void)` et le programme suivant fonctionne. Il vaut mieux éviter ces mécanismes hasardeux, en demandant au compilateur d'être strict et en utilisant l'option **-Wall**.

```c
/* Réponse dangereuse */
int main(void)
{
  return un();
}

int un(void)
{
```

```
  return 1;
}
```

- Ce qui donnera à la compilation

```
MacMini-TM:~/Documents/developpement/c thierry$ gcc -o faux.exe faux.c
MacMini-TM:~/Documents/developpement/c thierry$ gcc -Wall -o faux.exe faux.c
faux.c: In function 'main':
faux.c:4: warning: implicit declaration of function 'un'
```

Si l'on souhaite tout de même mettre la fonction *main* en premier, il aurait fallu avertir le compilateur en utilisant un prototype : la signature de la fonction, placé avant le *main*.

```
static int un(void);

int main(void)
{
  return un();
}

static int un(void)
{
  return 1;
}
```

Exercice 3 : programme *hello, world*

Écrire un programme qui affiche *hello, world*.

```
> gcc  -o hello.exe hello.c
> ./hello.exe
hello, world
>
```

Notions :

- inclusion d'en-tête standard
- passage de paramètres de fonction
- `printf`
- chaîne de caractères statique

Solution

On utilise le préprocesseur pour inclure `stdio.h` car le prototype de `printf` se trouve dans `stdio.h`. On passe la chaîne de caractères `"hello, world\n"` à l'appel de `printf`. On remarque que la chaîne est terminée par un caractère de nouvelle ligne (`\n`). Sans cela, une fois le programme terminé, l'invite de commande risque d'être affichée à la suite du mot *world* au lieu de se trouver au début d'une nouvelle ligne.

```
/* Bonne réponse */
#include <stdio.h>

int main(void)
```

```
{
  (void)printf("hello, world\n");
  return 0;
}
```

Remarques

- la fonction printf() retourne le nombre de caractère écrits dans le flux de sortie, ici j'ai décidé de ne pas utiliser ce code retour, je le signale en castant à void la fonction. L'outil qualité lint signale toute fonction retournant une valeur qui n'est pas affectée à une variable ou ignoré explicitement. Cela permet d'éviter l'erreur qui consiste à oublier de traiter un code retour de fonction.
- Que se passe-t-il si l'on omet le `#include <stdio.h>` ? Le compilateur va remarquer que l'on appelle une fonction `printf` dont il ne sait rien. Cela ne l'empêche pas de compiler le programme ! Le compilateur fait comme si `printf` avait été déclarée par : `extern int printf()`. Il y a déclaration implicite de fonction, alors que le compilateur ne sait rien de l'implémentation réelle de la fonction ! Si l'on omet d'inclure les prototypes des fonctions appelées, on risque donc d'appeler ces fonctions avec des **paramètres incompatibles sans avertissement du compilateur** :

```
/* Appel de printf aberrant risquant de compiler sans avertissement
   faute d'avoir inclu stdio.h. Un crash à l'exécution est probable. */
int main(void)
{
  printf(123);
  return 0;
}
```

L'option de gcc -Wall signale ce problème à la compilation par ce message :

```
hello.c:3: warning: implicit declaration of function 'printf'
```

Pour détecter le problème, c'est-à-dire que l'entier `123` (de type `int`) est passé au lieu d'une chaîne de caractères (de type `const char*`), le compilateur doit voir le prototype de `printf`, qui est : `int printf(const char*, ...)`. Ce qui nous intéresse avec le `#include <stdio.h>`, c'est que le préprocesseur génère la sortie suivante en entrée du compilateur :

```
/* Ce que le compilateur voit */
/* Contenu de stdio.h... */

/* Prototype de printf */
int printf(const char *, ...);

/* ... fin de stdio.h */

int main(void)
{
  printf("hello, world\n");
  return 0;
}
```

Notons que certains compilateurs ont une connaissance implicite des fonctions très communes comme `printf`. Ainsi, même sans voir le prototype de `printf`, certaines versions de GCC génèrent un avertissement, à moins d'utiliser la commande de compilation suivante : `gcc -fno-builtin -o hello.exe hello.c`. L'option **-Wall** active l'option --fno-builtin.

Exercice 4 : programme comptant les *arguments* en ligne de commande

Écrire un programme qui affiche le nombre de paramètres en ligne de commande (en anglais, les *arguments* sont les données passées au programme, ou à une fonction).

Compilation et exécutions attendues :

```
> gcc -Wall -o arg.exe arg.c
> ./arg.exe
0
> ./arg.exe foo bar
2
```

Notions :

- paramètres de `main`
- sortie formatée

Solution

Pour avoir accès aux arguments en ligne de commande, on utilise la seconde définition standard de `main`, qui a pour premier paramètre le nombre d'arguments `argc`, et pour second paramètre le vecteur d'arguments `argv`. Notons que sur Unix, le nom de l'exécutable lui-même (`./arg`) est un argument, donc `argc` vaut au moins 1. Le résultat recherché est `argc - 1`. Pour afficher un nombre entier, nous utilisons des caractères de conversion dans la chaîne de formatage passée à `printf`. Le caractère de conversion `d` suivant le caractère `%` indique que nous affichons un entier de type `int`.

```c
/* Bonne réponse */
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  (void)printf("%d\n", argc - 1);
  return EXIT_SUCCESS;
}
```

Variables et constantes

Entrée d'une valeur

Écrivez un programme age.c qui demande l'âge de l'utilisateur, puis qui l'affiche.

Pour lire l'age , vous utiliserez la fonction `scanf` déclarée dans stdio.h sous la forme `(void)scanf("%d", &ageLu);`.

Solution

Correction

```c
/*
Nom : age.c
Auteur : Thierry46
Role : Demande l'âge de l'utilisateur et l'affiche à l'écran.
Paramètres : non pris en compte.
Code retour : 0 (EXIT_SUCCESS)
Pour produire un exécutable avec le compilateur libre GCC :
   gcc -Wall -o age.exe age.c
Pour exécuter, tapez : ./age.exe

Version : 1.0 du 5/1/2008
Licence : GNU GPL
*/

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   /* Déclare la variable entière age qui contiendra la valeur lue */
   int ageLu = 99;

   /* Ecrit la question à l'écran */
   (void)puts("Quel est votre age :");

   /* Lit la réponse de l'utilisateur */
   (void)scanf("%d", &ageLu);

   /* Affiche à l'écran l'entier lu */
   (void)printf("Vous avez %d an(s) !\n", ageLu);
   (void)scanf("%d", &ageLu);

   return EXIT_SUCCESS;
}
```

Exécution

```
MacMini-TM:~/Documents/developpement/c thierry$ ./age.exe
Quel est votre age :
43
Vous avez 43 an(s) !
```

Avec valeur illégale (non entière) entrée :

```
MacMini-TM:~/Documents/developpement/c thierry$ ./age.exe
Quel est votre age :
?
Vous avez 99 an(s) !
```

Remarques

- L'initialisation de ageLu permet d'avoir une valeur par défaut (99) en cas de problème de lecture.
- Le code retour de scanf() est volontairement ignoré pour simplifier, ce qui est déconseillé dans un programme opérationnel.
- Il faudrait analyser le code retour et agir en conséquence.
- La fonction scanf() permet bien d'autres possibilités, voir sa page de man : *man -s3 scanf*.
- Voir les autres fonctions de cette famille : fscanf, sscanf...

Calculer si un nombre est premier

p est premier si et seulement si quel que soit 2<=i<=p-1 on a : p ne divise pas i.

Solution

```
#include<stdio.h>
#include<conio.h>
int main()
{int i,p;
         for(i=2;i<p;i++)
                     if(p%i==0) break;
                     if (i==p)
                     printf("p est premier");
    else printf("p n'est pas premier");
getch();
}
```

Types

Exercice 1 : longueurs en octet des types du langage C

Écrivez un programme taille.c qui affiche à l'écran la taille des différents types de données en octet du langage C sur votre architecture machine.

Vous utiliserez l'opérateur *sizeof(type)*.

Solution

Correction

```
/*
 Nom : taille.c
 Auteur : Thierry46
 Role : affiche à l'écran la taille en octet des différents types de données
 du langage C sur votre architecture.
 Paramètres : non pris en compte.
 Code retour : 0 (EXIT_SUCCESS)
 Pour produire un exécutable avec le compilateur libre GCC :
 gcc -Wall -pedantic -std=c99 -o taille.exe taille.c
 Pour exécuter, tapez : ./taille.exe

 Version : 1.2 du 10/3/2008
 Licence : GNU GPL
 */

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

int main(void)
```

```
{
   /* Affiche les tailles récupérées avec sizeof() */
   (void)printf("Taille des types en octet :\n"
      "\t- type char : %zu\n"
      "\t- type short : %zu\n"
      "\t- type int : %zu\n"
      "\t- type long : %zu\n"
      "\t- type long long : %zu\n"
      "\t- type float : %zu\n"
      "\t- type double : %zu\n"
      "\t- type long double : %zu\n",
      sizeof(char)*CHAR_BIT/8,
      sizeof(short)*CHAR_BIT/8, sizeof(int)*CHAR_BIT/8,
      sizeof(long)*CHAR_BIT/8, sizeof(long long)*CHAR_BIT/8,
      sizeof(float)*CHAR_BIT/8, sizeof(double)*CHAR_BIT/8,
      sizeof(long double)*CHAR_BIT/8);

   return EXIT_SUCCESS;
}
```

Résultats affichés sur ma machine :

```
MacMini-TM:~/Documents/developpement/c thierry$ ./taille.exe
Taille des types en octet :
        - type char : 1
        - type short : 2
        - type int : 4
        - type long : 4
        - type long long : 8
        - type float : 4
        - type double : 8
        - type long double : 16
```

Remarques :

- L'opérateur `sizeof` renvoie un résultat de type **size_t**. L'unité du résultat est de type char. Un char ne mesure pas forcément un octet ou 8 bits sur toutes les architectures. C'est pourquoi, pourdes raisons de portabilité, j'utilise la constante CHAR_BIT qui indique le nombre de bits qu'occupe un char.
- Une valeur de ce type size_t s'imprime avec le spécificateur de format **%zu** de `printf`.

Exercice 2 : caractéristiques numériques des types du langage C

Écrivez un programme limites.c qui affiche à l'écran les caractéristiques numériques des différents types du langage C.

Vous utiliserez les constantes symboliques des includes limits.h et float.h.

Solution

Correction

```
/*****************************************************
Nom ......... : limites.c
Role ........ : Imprime les caractéristiques numériques
pour les types du langage C
```

```c
/*
Auteur ..... : Thierry46
Version ..... : V1.1 du 10/3/2008
Licence ..... : GNU GPL

Compilation :
gcc -Wall -pedantic -std=c99 -o limites.exe limites.c
Pour exécuter, tapez : ./limites.exe
*****************************************************/
#include <stdio.h>
#include <stdlib.h>
// Pour les limites lies aux entiers et char
#include <limits.h>
// Pour les limites lies aux nombres reels
#include <float.h>

int main(int argc, char *argv[])
{
        // Impression des limites pour les nombres entiers
        (void)printf("CHAR_MIN = %d\n", CHAR_MIN);
        (void)printf("CHAR_MAX = %d\n", CHAR_MAX);
        (void)printf("INT_MIN = %d\n", INT_MIN);
        (void)printf("INT_MAX = %d\n", INT_MAX);
        (void)printf("LONG_MIN = %ld\n", LONG_MIN);
        (void)printf("LONG_MAX = %ld\n", LONG_MAX);

        // Impression des limites pour les réels
        (void)printf("FLT_EPSILON = %e\n", FLT_EPSILON);
        (void)printf("FLT_MIN = %e\n", FLT_MIN);
        (void)printf("FLT_MAX = %e\n", FLT_MAX);
        (void)printf("DBL_EPSILON = %le\n", DBL_EPSILON);
        (void)printf("DBL_MIN = %le\n", DBL_MIN);
        (void)printf("DBL_MAX = %le\n", DBL_MAX);

        return EXIT_SUCCESS;
} /* int main(... */
```

Résultats affichés sur ma machine :

```
MacMini-TM:~/Documents/developpement/c thierry$ ./limites.exe
CHAR_MIN = -128
CHAR_MAX = 127
INT_MIN = -2147483648
INT_MAX = 2147483647
LONG_MIN = -2147483648
LONG_MAX = 2147483647
FLT_EPSILON = 1.192093e-07
FLT_MIN = 1.175494e-38
FLT_MAX = 3.402823e+38
DBL_EPSILON = 2.220446e-16
DBL_MIN = 2.225074e-308
DBL_MAX = 1.797693e+308
```

Les opérateurs

Pour les besoins de certains exercices, on rappelle le tableau de priorité des opérateurs du C :

Catégorie d'opérateurs	Opérateurs	Associativité

fonction, tableau, membre de structure, pointeur sur un membre de structure	( ) [ ] . ->	Gauche -> Droite
opérateurs unaires	- ++ -- ! ~ * & sizeof (type)	Droite ->Gauche
multiplication, division, modulo	* / %	Gauche -> Droite
addition, soustraction	+ -	Gauche -> Droite
décalage	<< >>	Gauche -> Droite
opérateurs relationnels	< <= > >=	Gauche -> Droite
opérateurs de comparaison	== !=	Gauche -> Droite
et binaire	&	Gauche -> Droite
ou exclusif binaire	^	Gauche -> Droite
ou binaire	|	Gauche -> Droite
et logique	&&	Gauche -> Droite
ou logique	||	Gauche -> Droite
opérateur conditionnel	? :	Droite -> Gauche
opérateurs d'affectation	= += -= *= /= %= &= ^= |= <<= >>=	Droite -> Gauche
opérateur virgule	,	Gauche -> Droite

La priorité des opérateurs va décroissante lorsqu'on se déplace du haut du tableau vers le bas du tableau. Quand les opérateurs ont même priorité, c'est la colonne de droite sur la distributivité qui est utilisée.

Analyse de programmes

Nous allons présenter dans cette section un ensemble d'exercices destinés à se familiariser avec les opérateurs du langage C.

Exercice 1

- Quelle est la valeur de i après la suite d'instructions :

```
int i=10;
i = i-(i--);
```

Solution

La série d'instructions donnée est équivalente à :

```
int i=10;
i = i-(i); // on retire la post décrémentation pour la mettre après
i--;
```

Donc i vaut -1

- Quelle est la valeur de i après la suite d'instructions :

```
int i=10;
i = i-(--i);
```

Solution

La série d'instructions donnée est équivalente à :

```
int i=10;
i--; // i passe à 9
i = i-(i); // on retire la pré-décrémentation pour la mettre avant
```

Donc i vaut 0

Exercice 2 : la priorité des opérateurs

Enlever les parenthèses des expressions suivantes lorsqu'elles peuvent être retirées.

```
a=(25*12)+b;
if ((a>4) &&(b==18)) { }
((a>=6)&&(b<18))||(c!=18)
c=(a=(b+10));
```

Évaluer ces expressions pour a=6, b=18 et c=24. On supposera que les valeurs données le sont pour chacune des lignes : il n'y a pas d'exécution séquentielle comme dans un programme.

Petit programme de calcul

Écrivez un programme calcul.c qui calcule la distance entre deux points d'un plan :

- Lit les coordonnées de deux points : X_1 (x_1, y_1) et X_2 (x_2, y_2).

- Affiche les données lues
- Calcule la distance d entre les deux points X_1 et X_2, avec la formule :

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Affiche le résultat à l'écran.

Solution

```c
/*
Nom : calcul.c
Auteur : Thierry46
Role : calcule la distance entre deux points d'un plan.
Paramètres : non pris en compte.
Code retour : 0 (EXIT_SUCCESS)
Pour produire un exécutable avec le compilateur libre GCC :
   gcc -lm -Wall -o calcul.exe calcul.c
Pour exécuter, tapez : ./calcul.exe

Remarque : Utilise des fonctions requises par la norme C99
Version : 1.0 du 6/1/2008
Licence : GNU GPL
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[])
{
        // Declaration et initialisation des variables
        float x1 = 0.0;
        float y1 = 0.0;
        float x2 = 0.0;
        float y2 = 0.0;
        float d = 0.0;
        float dx, dy;

        // Lit les coordonnées de deux points : X1 (x1, y1) et X2
        (void)puts("Entrez les coordonnees du point X1 (deux nomb
        (void)scanf("%f %f", &x1, &y1);
        (void)puts("Entrez les coordonnees du point X2 (deux nomb
        (void)scanf("%f %f", &x2, &y2);

        // Affiche les nombres saisis
        (void)printf("\nX1(%g, %g) et X2(%g, %g)\n", x1, y1, x2,

        // Calcule la distance d entre les deux points X1 et X2
        dx = x2 - x1;
        dy = y2 - y1;
        d = sqrtf(dx*dx + dy*dy);

        // Affiche le résultat à l'écran
        (void)printf("d(X1, X2) = %g.\n", d);

        return EXIT_SUCCESS;
}
```

Structures de contrôle

Structures conditionnelles

Analyse de programme : choix multiple

Ces exercices d'analyses doivent être réalisés sans compilateurs, à la main à l'aide d'un crayon et d'une feuille de papier.

Soit le programme suivant :

```c
#include <stdio.h>
#include <stdlib.h>
int main(void) {
  unsigned char i=7;
  i=i/2; //"/": division entiere...
  switch(i) {
    case 1 : (void)printf("Premier\n");break;
    case 2 : (void)printf("Deuxième\n");break;
    case 3 : (void)printf("Troisième\n");break;
    default : (void)printf("Non classe\n");
  }
  return EXIT_SUCCESS;
}
```

Qu'est ce qui sera affiché à l'écran lors de l'exécution de ce programme ?

Même question pour le programme :

```c
#include <stdio.h>
#include <stdlib.h>
int main(void) {
  int i=18;
  i=i-(--i);
  switch(i) {
    case 1 : (void)printf("Premier\n");
    case 2 : (void)printf("Deuxième\n");
    case 3 : (void)printf("Troisième\n");
    default : (void)printf("Non classe\n");
  }
  return EXIT_SUCCESS;
}
```

Solution

- pour le premier programme :

Comme i vaut 7 au début du programme, on pourrait penser que i/2 donne 3.5 qui est arrondi à 4 (puisqu'affecté à une variable entière sur un octet). Mais il n'en est rien : 8/3 = 2.66 ne donne pas plus 3 mais seulement 2... parceque 8/3 donne un quotient de 2 avec 2 comme reste et que l'on s'intéresse à ce quotient justement ! C'est cela la division entière. Don quand on arrive au switch, i vaut 3 et donc

```
Troisième
```

s'affiche à l'écran.

- Pour le deuxième programme, c'est

```
Non classe
```

qui s'affiche car :

```
int i=18;
  i=i-(--i);
```

est la même chose que :

```
int i=18;
  i--; // i passe à 17
  i=i-(i); // 17-17 donne 0
```

Remarquons que

```
int i=18;
  i=i-(i--);
```

donnerait exactement le même résultat (dans le switch).

La suppression des break a des conséquences importantes. Par exemple

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
  int i=2;
  switch(i) {
    case 1 : (void)printf("Premier\n");
    case 2 : (void)printf("Deuxième\n");
    case 3 : (void)printf("Troisième\n");
    default : (void)printf("Non classe\n");
  }
  return EXIT_SUCCESS;
}
```

afficherait :

```
Deuxième
Troisième
Non classe
```

Test simple : programme non complet

Écrire la partie de programme réalisant la structure de contrôle demandée. (On ne demande pas le programme en entier : pas les "include" pas le main()...)

Un test sur une valeur dans la variable i : si i vaut 19 on écrit "OK" autrement on écrit "Rejet"

Solution

Cet exercice n'est qu'un échauffement pour l'exercice suivant.

```
/******* langage C *********/
if (i==19)
  printf("OK");
else
  printf("Rejet");
```

L'erreur la plus fréquente est d'écrire :

```
/******* langage C *********/
if (i=19)
  printf("OK");
else
  printf("Rejet");
```

qui affichera toujours OK quelque soit la valeur de i ! Pouvez-vous comprendre pourquoi ? Parce que i=19 met 19 dans i qui n'est pas zéro et donc est vrai... et toujours vrai !!!

Tester votre âge

Écrire un programme testage.c contenant une fonction `main` qui :

- lit sur le clavier l'âge de l'utilisateur avec la fonction `scanf`;
- teste si la réponse est valide par analyse du code retour de `scanf` et teste si la valeur est comprise entre 0 et 130;
- affiche si l'utilisateur est majeur (>= 18 ans) ou mineur.

Solution

```
/*
Nom : testage.c
Auteur : Thierry46
But : Demande l'âge de l'utilisateur,
        teste si la réponse est valide et
        affiche si l'utilisateur est majeur (>= 18 ans).
Paramètres : non pris en compte.
Code retour : 0 (EXIT_SUCCESS)
Pour produire un exécutable avec le compilateur libre GCC :
   gcc -Wall -std=c99 -o testage.exe testage.c
Pour exécuter, tapez : ./testage.exe

Version : 1.0 du 18/1/2008
Licence : GNU GPL
*/

#include <stdio.h>
#include <stdlib.h>

/* Constantes symboliques pour changement plus facile :
        pas besoin de chercher toutes les occurrences de la const
        dans le corps de la fonction. */
#define AGE_MAJEUR 18
```

```c
#define AGE_MINI 0
#define AGE_MAXI 130

int main(void)
{
        // Declare les variables
        int ageLu = 0;
        int nbChampLu = 0;
        int codeRetour = EXIT_SUCCESS; // Valeur par defaut

        // Ecrit la question à l'écran
        (void)puts("Quel est votre age :");

        // Lit la réponse de l'utilisateur : un entier
        nbChampLu = scanf("%d", &ageLu);

        // Teste si scanf a bien converti le nombre entré
        if (nbChampLu != 1)
        {
                (void)fputs("\aErreur : Vous devez entrer un nomb
                codeRetour = EXIT_FAILURE;
        }
        else if (ageLu < AGE_MINI || ageLu>AGE_MAXI)
        {
                (void)fprintf(stderr,
                        "\aErreur : Vous devez entrer un nombre e
                        AGE_MINI, AGE_MAXI);
                codeRetour = EXIT_FAILURE;
        }
        else
        {
                // Le test se fait à l'appel de puts
                (void)puts((ageLu < AGE_MAJEUR) ? "Tu es mineur !
        }

   return codeRetour;
} // int main(...
```

Structures répétitives

Analyse de programme : boucle simple

Cet exercice d'analyse doit être réalisé sans compilateur, à la main à l'aide d'un crayon et d'une feuille de papier. Soit le programme suivant :

```c
#include <stdio.h>
#include <stdlib.h>
int main(void){
  int i;
  for (i=0;i<5;i++) {// "/": division entiere
    (void)printf("Module EC%d\n",(i+9)/(i+1));
  }
  return EXIT_SUCCESS;
}
```

Qu'affichera à l'écran l'exécution de ce programme ?

solution

La seule difficulté de ce programme est la division entière, d'ailleurs précisée dans le commentaire. C'est la division que l'on a appris à l'école primaire : 5 / 2 : quotient = 2 et reste =1. La division entière donne le quotient c'est à dire 2 (et non pas 2,5 !!!)

```
Module EC9
Module EC5
Module EC3
Module EC3
Module EC2
```

Analyse de programme : boucle simple et tableau

Cet exercice d'analyse doit être réalisé sans compilateur, à la main à l'aide d'un crayon et d'une feuille de papier.

Soit le programme suivant :

```c
#include <stdio.h>
#include <stdlib.h>
int main(void){
  unsigned char i;
  unsigned char tab[5];
//initialisation du tableau
  tab[0]=1;tab[1]=2;tab[2]=4;tab[3]=8;tab[4]=16;
  for (i=0;i<5;i++) {
    (void)printf("Le %d° elt est %d\n",i+1,tab[i]);
  }
  return EXIT_SUCCESS;
}
```

Qu'affichera à l'écran l'exécution de ce programme ?

solution

Tableau de 5 cases numérotées de 0 à 4. Le programme affiche tout simplement son contenu :

```
Le 1° elt est 1
Le 2° elt est 2
Le 3° elt est 4
Le 4° elt est 8
Le 5° elt est 16
```

Remarque : Il est possible de remplacer :

```c
//********** Langage C ***************
  unsigned char tab[5];
//initialisation du tableau
  tab[0]=1;tab[1]=2;tab[2]=4;tab[3]=8;tab[4]=16;
//**********Fin Langage C ***************
```

par

```c
//********** Langage C ***************
```

```
//déclaration du tableau avec initialisation
  unsigned char tab[5]={1,2,4,8,16};
//**********Fin Langage C ***************
```

Voir même par :

```
//********** Langage C ***************
//déclaration du tableau avec initialisation
  unsigned char tab[10]={1,2,4,8,16};
//**********Fin Langage C ***************
```

pour lequel on déclare un tableu de 10 case mais on en initialise que 5 !

Somme d'entiers (boucle simple)

Ecrire un programme *somme* demandant à l'utilisateur de taper 10 entiers et qui affiche leur somme. Le programme ne devra utiliser que 3 variables et ne devra pas utiliser de tableau.

Solution

```
/*
Programme : somme
Role : demander 10 entiers et afficher leur somme
Compilation :
   gcc -Wall -std=c99 -o somme.exe somme.c
Execution :
   ./somme.exe
Version 2.0 du 9/1/2008
*/
#include <stdio.h>
#include <stdlib.h>

#define NB_ENTIERS 10

int main(void)
{
    int valeurLue = 0;
    int somme = 0;

    // Lecture des valeurs
    (void)printf("Vous allez devoir entrer %d entiers\n", NB_ENTI
    for (int i = 0; i < NB_ENTIERS ; i++)
    {
        // Demande et lecture de l'entier
        (void)printf("Tapez l'entier de rang %d : ", i+1);
        (void)scanf("%d", &valeurLue);
        somme = somme + valeurLue;
    } // for (int i = 0...

    // Affichage du résultat
    (void)printf("la somme des %d entiers vaut : %d\n",
        NB_ENTIERS, somme);

    return EXIT_SUCCESS;
} // int main(...
```

Remarques qualité sur la correction proposée :

- Un cartouche d'entête indique des informations utiles dont les particularités

de la compilation (-std=c99), la version.
- Les messages guident l'utilisateur : il sait ou il en est dans sa saisie.
- Utilisation d'une constante symbolique *NB_ENTIERS* au lieu de valeurs littérales 10 : si le nombre d'entier à demander change, il n'y aura qu'une seule ligne à modifier au lieu de 3 dans notre cas.
- *for (int i = 0* : c'est une syntaxe autorisée en C99. Avec les anciennes normes C, La variable aurait dû être déclarée hors de la boucle. Cette nouvelle syntaxe permet d'avoir des variables d'itération locales à la boucle for.
- *(void)printf(...* : la fonction printf retourne le nombre de caractères écrit dans le flux de sortie. Comme j'ai décidé de ne pas utiliser ce code retour, je le signale en castant à void la fonction. L'outil qualité *lint* signale toute fonction retournant une valeur qui n'est pas affectée à une variable ou ignoré explicitement. Cela permet d'éviter l'erreur qui consiste à oublier de traiter un code retour de fonction.
- Le code retour de scanf est ignoré. Il renvoie le nombre de valeurs correctement récupérée. Dans un code opérationnel robuste, tous les codes retours doivent être traités et un traitement doit être prévu en cas d'erreur. Le type void n'existait pas en C K&R.

Analyse de programme (double boucle)

Ces exercices d'analyses doivent être réalisés sans compilateurs, à la main à l'aide d'un crayon et d'une feuille de papier.

- Soit le programme suivant :

```c
#include <stdio.h> // pour printf
#include <stdlib.h> // pour system
main() {
  int i,j;
    system("clear"); //efface ecran sous linux (system("cls"); sous Windows)
    for(i=0;i<5;i++){
      for(j=i;j<5;j++)
        (void)printf("**");
      (void)printf("\n");
    }
    return EXIT_SUCCESS;
}
```

Que sera-t-il affiché à l'écran lors d'une exécution de ce programme ?

- Même question pour le programme suivant :

```c
#include <stdio.h> // pour printf
#include <stdlib.h> // pour system
main() {
  int i,j;
    system("clear"); //efface ecran sous linux (system("cls"); sous Windows)
    for(i=0;i<5;i++){
      for(j=5-i;j<5;j++)
        (void)printf("++");
      (void)printf("\n");
    }
    return EXIT_SUCCESS;
}
```

solution

- Pour le premier programme :

```
**********
********
******
****
**
```

- Pour le deuxième programme :

```

++
++++
++++++
++++++++
```

A noter la ligne vide en tout début.

Triangle d'étoiles (double boucles)

Compléter la fonction afficherTriangle dans le programme ci-dessous : cette fonction devra afficher un triangle rempli d'étoiles (*) sur un nombre de lignes donné passé en paramètre, exemple :

```
*
**
***
****
*****
******
*******
********
```

- 1ère version : sans utiliser de tableau à l'aide de deux boucles *for* imbriquées.
- 2ème version : avec une seule boucle *for* et un tableau de chaîne de caractère où vous accumulerez des étoiles.

```c
/*
Nom : etoile.c
Compilation : gcc -Wall -std=c99 -o etoile.exe etoile.c
Exécution : ./etoile.exe
*/
#include <stdio.h>
#include <stdlib.h>

/*
Nom ... : afficherTriangle
Role .. : Afficher un triangle d'etoiles
Parametre :
        nbLignes : nombre de lignes du triangle
*/
static void afficherTriangle(const int nbLignes)
{
// Partie à compléter
```

```
} // static void afficherTriangle(...

// Fonction principale pour test
int main(void)
{
        int nbLignes = 0;
        int nbChampsLu = 0;
        int codeRetour = EXIT_SUCCESS;

        (void)fputs("Lignes ? ", stdout);
        nbChampsLu = scanf("%u", &nbLignes);
        if (nbChampsLu == 1 && nbLignes > 0)
        {
                afficherTriangle(nbLignes);
        }
        else
        {
                (void)fputs("Erreur : Vous devez entrer un entier strictement positif !\n
                        stderr);
                codeRetour = EXIT_FAILURE;
    }

    return codeRetour;
} // int main(...
```

Solution

1ère version : sans utiliser de tableau à l'aide de deux boucles *for* imbriquées.

```
/*
Nom ... : afficherTriangle
Role .. : Afficher un triangle d'etoiles
Parametre :
        nbLignes : nombre de lignes du triangle
Version 1 du 9/1/2008
*/
static void afficherTriangle(const int nbLignes)
{
    for (int numLigne = 1; numLigne <= nbLignes; numLigne++)
    {
        for (int numColonne = 1; numColonne <= numLigne; numColon
        {
            (void)putchar('*');
        }
        (void)putchar('\n');
    }
} // static void afficherTriangle(...
```

Remarque :

- *for (int i = 0* : c'est une syntaxe introduite en C99. Avec les anciennes normes C, La variable aurait dû être déclarée hors de la boucle. Cette nouvelle syntaxe permet d'avoir des variables d'itération locales à la boucle for.
- *(void)putchar('*');* : la fonction putchar retourne le caractère écrit dans le flux de sortie ou EOF en cas d'erreur. Comme j'ai décidé de ne pas utiliser ce code retour, je le signale en castant à void la fonction. L'outil qualité *lint* signale toute fonction retournant une valeur qui n'est pas affectée à une variable ou ignoré explicitement. Cela permet d'éviter l'erreur qui consiste à oublier de traiter un code retour de fonction.

Dans un code opérationnel robuste, tous les codes retours doivent être traités et un traitement doit être prévu en cas d'erreur. Le type void n'existait pas en C K&R.

2ème version : avec une seule boucle *for* et un tableau de chaîne de caractère où vous accumulerez des étoiles.

```
/*
Nom ... : afficherTriangle
Role .. : Afficher un triangle d'etoiles
Parametre :
        nbLignes : nombre de lignes du triangle
Version 2 du 9/1/2008
*/
static void afficherTriangle(const int nbLignes)
{
        // Déclaration de la chaine de caractère (voir ma remarqu
        char chaineEtoiles[nbLignes+1];

        // Initialisation du tableau avec un terminateur de chain
        chaineEtoiles[0] = '\0';

    for (int numLigne = 1; numLigne <= nbLignes; numLigne++)
    {
        (void)strcat(chaineEtoiles, "*");
        (void)puts(chaineEtoiles);
    }
} // static void afficherTriangle(...
```

Remarque :

- Le tableau chaineEtoiles est de type VLA (Variable-Length Arrays) introduite par C99. Avec les versions de C précédente, il aurait fallu allouer dynamiquement le tableau au début du programme, puis le désalloué à la fin.

Tableaux statiques

Parcours d'un tableau

Écrire un programme nommé argv.c qui affiche :

- son nom de lancement (argv[0]);
- le nombre des ces arguments;
- la valeur de chaque argument reçu.

Rappels : La fonction main d'un programme C reçoit en argument :

- un entier **argc** indiquant le nombre d'élément du tableau argv;
- un tableau de chaines de caractère **argv** avec :
  - argv[0] : Nom d'appel du programme.
  - argv[i] : Valeur de l'argument de rang i.

Solution

Le fichier source en c99 :

```c
/*
Nom : argv.c
Auteur : Thierry46
Role : Affiche le nom de lancement du programme
        le nombre et les valeurs des paramètres passés passés à ce programme.
Paramètres :
        - argc : nombre de paramètres
        - argv : tableau de chaine de caractere contenant les parametre
Code retour : 0 (EXIT_SUCCESS)
Pour produire un exécutable avec le compilateur libre GCC :
   gcc -Wall -std=c99 -o argv.exe argv.c
Pour exécuter, tapez : ./argv.exe

Version : 1.0 du 16/1/2008
Licence : GNU GPL
*/

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
        // Affiche le nom de lancement du programme
        (void)printf("Nom d'appel du programme (argv[0]) : %s\n", argv[0]);

        // Affiche le nombre de parametre passe au programme
        (void)printf("Nombre de parametre passe au programme : %d\n", argc-1);

        // Affiche la valeur des parametres
        for (int i=1; i<argc; i++)
        {
                (void)printf("Parametre %d : %s\n", i, argv[i]);
        }

   return EXIT_SUCCESS;
} // int main(...
```

```
MacMini-TM:~/Documents/developpement/c thierry$ gcc -Wall -std=c99 -o argv.exe argv.c
MacMini-TM:~/Documents/developpement/c thierry$ ./argv.exe dudule 1
Nom d'appel du programme (argv[0]) : ./argv.exe
Nombre de parametre passe au programme : 2
Parametre 1 : dudule
Parametre 2 : 1
```

Position d'un élément dans un tableau

Écrire un programme *position.c* contenant une fonction principale *main* déterminant si un entier est contenu dans un tableau statique par l'appel à une fonction *position*.

La fonction main :

- définira et initialisera le tableau d'entier
- récupèrera dans son tableau d'argument argv le nombre à chercher.
- appellera la fonction *position*.

- affichera l'indice de l'élément dans le tableau ou un message indiquant que le nombre n'a pas été trouvé.

La fonction *position* :

- aura pour prototype : `static int position(int t[], int taille, int x)`.
- donnera l'indice d'un élément x dans le tableau t, ou -1 si x n'est pas trouvé.

Solution

Le fichier source en C99 :

```
/*
Nom : position.c
Role : Affiche le nom de lancement du programme
        le nombre et les valeurs des paramètres passés passés à ce programme.
Paramètres :
        - argc : nombre de paramètres
        - argv : tableau de chaine de caractere contenant les parametres
Code retour : 0 (EXIT_SUCCESS)
Pour produire un exécutable avec le compilateur libre GCC :
   gcc -Wall -std=c99 -o position.exe position.c
Pour exécuter, tapez : ./position.exe

Version : 1.0 du 16/1/2008
Licence : GNU GPL
*/
#include <stdio.h>
#include <stdlib.h>
static int position(int t[], int taille, int x);

int main(int argc, char *argv[])
{
        int tableau[] = { 2, 5, 45, 3, 9 };
        int taille;
        int x = 0;
        int cr = 0;
        int resultat;

        // Détermination de la taille du tableau
        taille = (int)(sizeof(tableau)/sizeof(tableau[0]));

        // Contrôle nombre de paramètre et affiche usage
        if (argc != 2)
        {
                (void)fprintf(stderr,
                        "%s : Cherche un entier x passe en parametre\n"
                        "usage : %s x\n",
                        argv[0], argv[0]);
                (void)fputs("Valeurs contenues dans le tableau : ", stderr);
                for (int i=0; i<taille; i++)
                {
                        (void)fprintf(stderr, "%d ", tableau[i]);
                }
                (void)fputs("\n",stderr);
                exit(EXIT_FAILURE);
        } // if (argc != 2)

        // Recupere le nombre passe en parametre et le convertit
        cr = sscanf(argv[1], "%d", &x);
        if (cr != 1)
        {
                (void)fprintf(stderr,
                        "Erreur : parametre \"%s\" invalide : doit etre un entier.\n",
```

```
                        argv[1]);
                exit(EXIT_FAILURE);
        } // if (cr != 1)

        // Appel de la fonction position et affichage du resultat.
        resultat = position(tableau, taille, x);
        if (resultat != -1)
        {
                (void)printf("La position de %d est : %d\n", x, resultat);
        }
        else
        {
                (void)printf("%d non trouve !\n", x);
        }

    return EXIT_SUCCESS;
}

/*
Fonction ..... : position
Rôle ......... : Retourne la position de l'entier x dans la tableau t
Paramètres .... :
        - t : tableau d'entiers passé en entier.
        - taille : nombre d'éléments du tableau
        - x : valeur a rechercher.
Retour : L'indice dans le tableau de l'élément x ou -1 si non trouve.
*/
static int position(int t[], int taille, int x)
{
        // On commence la recherche à partir de la fin du tableau
        int indice = taille - 1;
        while ((indice >= 0) && (t[indice] != x))
        {
                indice--;
        }
        return indice;
} // static int position(...
```

```
MacMini-TM:~/c thierry$ gcc -Wall -std=c99 -o position.exe position.c
MacMini-TM:~/c thierry$ ./position.exe
./position.exe : Cherche un entier x passe en parametre
usage : ./position.exe x
Valeurs contenues dans le tableau : 2 5 45 3 9
MacMini-TM:~/c thierry$ ./position.exe a
Erreur : parametre "a" invalide : doit etre un entier.
MacMini-TM:~/c thierry$ ./position.exe 5
La position de 5 est : 1
MacMini-TM:~/c thierry$ ./position.exe 99
99 non trouve !
```

Fonctions

Ces exercices concernent l'utilisation des fonctions de la bibliothèque standard du langage C.

Filtre qui passe le texte en majuscule

Problème à résoudre

Ecrivez un programme majuscule.c qui lit des données sur le flux stdin et écrits sur stdout après avoir transformé les caractères lus en majuscules. Vous utiliserez les fonctions `getchar`, `putchar` (stdio.h) et `toupper` (ctype.h).

Vous testerez votre programme en lui faisant convertir son propre fichier source majuscule.c.

majuscule.exe < majuscule.c

Solution proposée

Solution proposée

```c
/**
 * Programme . : majuscule
 * Role ..... : convertit les caractéres lus sur stdin en majuscule.
 * Parametres : Aucun
 * Compilation et EDL : gcc -Wall -o majuscule.exe majuscule.c
 *      Auteur ..... : Thierry46
 *      Version ..... : 1.0 du 4/2/2008
 *      Licence ..... : GNU GPL.
 */

#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>

int main(void)
{
        int c;
        while ((c=getchar()) != EOF)
        {
                (void)putchar(toupper(c));
        }

        return EXIT_SUCCESS;
}
```

Lire une ligne longue avec fgets

Problème à résoudre

La fonction `fgets` de la bibliothèque standard du langage C permet de lire une chaine de caractère de longueur limitée dans un flux.

Vous allez compléter une fonction *lire_ligne* répondant au spécifications suivantes :

- Retour d'une ligne lue dans un flux texte passé en paramètre.
- Vous éliminerez les caractères de saut de ligne lus.
- La longueur des lignes lues n'est pas limitée.
- Contrôle des paramètres et retour des codes d'erreurs systèmes, détection de la fin du fichier.
- Vous utiliserez au maximum les fonctions de la bibliothèque standard du langage C : allocation mémoire, chaines de caractères...
- Son prototype est donné par *lire_ligne.h*.
- Vous utiliserez le programme de *main_lire_ligne.c* pour lire_ligne.
- Vous devrez traiter le fichier *test_lire_ligne.txt* fourni.

- Les instructions de compilation et d'édition de lien sont dans les commentaires des fichiers fournis.

Éléments fournis

lire_ligne.c à compléter

```c
/**
 * Fonction .. : lire_ligne
 * Role ..... : tente de lire une ligne entiere depuis le flux.
 *      - Le caractere de saut de ligne '\n' final est enleve.
 *      - L'appellant doit se charger de la desallocation de la chaine retournee.
 *
 * Parametres :
 *      - pChaine : adresse de retour de la chaine
 *      - tailleBufferLecture : taille du buffer de lecture,
 *              dans la pratique on pourra choisir _POSIX_MAX_INPUT (ref limits.h).
 *      - flux : un pointeur sur le descripteur du fichier a lire.
 *
 *      - Valeur retournee :
 *              - Si OK : EXIT_SUCCESS.
 *              - Si Fin de fichier atteinte sans rencontrer \n : EOF,
 *                      - la chaine retournée contient le debut de la ligne longue.
 *                      - Si rien a lire, retour d'un pointeur NULL pour la chaine.
 *              - En cas de probleme : Le code d'erreur systeme utilisable par perror.
 *                      et retour d'un pointeur NULL pour la chaine.
 *
 * Auteur ..... : Votre nom
 * Version ..... : 1.0 du date
 * Licence ..... : GNU GPL.
    lire_ligne : Lecture d'une ligne longue dans un fichier texte
    Copyright (C) 2008  Thierry46

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 * Compilation . : gcc -pedantic -Wall -std=c99 -c lire_ligne.c
 * Fichiers lies : main_lire_ligne.c, lire_ligne.h
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <assert.h>
#include <errno.h>
#include "lire_ligne.h"

/*@null@*/ char *lire_ligne(size_t tailleBufferLecture,
        FILE * restrict flux, int *pCodeRetour)
{
// Début partie à compléter...
// Fin partie à compléter...
} // char *lire_ligne(...
```

lire_ligne.h

```
/*
 *  Nom .... : lire_ligne.h
 *  Role ... : Definitions et prototype de lire_ligne.h
 * Auteur ..... : Thierry46
 * Version ..... : 1.0 du 30/1/2008
 * Licence ..... : GNU GPL.
    lire_ligne : Lecture d'une ligne longue dans un fichier texte
    Copyright (C) 2008  Thierry46

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 * Fichier lie : lire_ligne.c
 */
#ifndef LIRE_LIGNE_H
#define LIRE_LIGNE_H

extern /*@null@*/ char *lire_ligne(size_t tailleBufferLecture,
        FILE * restrict flux, int *pCodeRetour);

#endif
```

main_lire_ligne.c

```
/*
 *  Nom ......... : main_lire_ligne.c
 *  Role ........ : Tester la fonction lire_ligne
 *  Parametre ... : Aucun
 *   Code retour . : EXIT_SUCESS (0)
 *   Auteur ..... : Thierry46
 *   Version ..... : 1.1 du 4/2/2008
 *   Licence ..... : GNU GPL.
    lire_ligne : Lecture d'une ligne longue dans un fichier texte
    Copyright (C) 2008  Thierry46

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 * Compilation . :
 *   Voir lire_ligne .c pour sa compilation
```

```c
 *   gcc -pedantic -Wall -std=c99 -o test_lire_ligne.exe main_lire_ligne.c lire_ligne.o
 * Exécution : ./test_lire_ligne.exe
 * Fichiers lies : lire_ligne.c, lire_ligne.h, test_lire_ligne.txt
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lire_ligne.h"

#define FIC_TEST "test_lire_ligne.txt"
// Valeur faible de TAILLE_BUFFER pour test débordements et lecture multiple.
#define TAILLE_BUFFER (size_t)10
#define TAILLE_MAX_MESSAGE 256

int main(void)
{
   FILE *hFicTest = NULL;
   char *chaine = NULL;
   int numLigne = 1;
   int retour = EXIT_SUCCESS;
   char message[TAILLE_MAX_MESSAGE];

   hFicTest = fopen(FIC_TEST, "r");
   if (hFicTest == NULL)
   {
      (void)printf("%s : le fichier %s n'a pas pu etre ouvert !\n",
         __func__, FIC_TEST);
      // Je laisse poursuivre pour tester assertion dans lire_ligne. //
   }

   do
   {
      chaine = lire_ligne(TAILLE_BUFFER, hFicTest, &retour);
      if (retour == EXIT_SUCCESS)
      {
         (void)strncpy(message, "Pas d'erreur", sizeof(message) - 1);
      }
      else if (retour == EOF)
      {
         (void)strncpy(message, "Fin de fichier", sizeof(message) - 1);
      }
      else
      {
         (void)strncpy(message, strerror(retour), sizeof(message) - 1);
      }
      message[sizeof(message) - 1] = '\0';

      (void)printf("Ligne %d : retour = %d (%s), chaine lue = \"%s\".\n",
         numLigne, retour, message,
         (chaine == NULL) ? "! Pointeur NULL !" : chaine);

      free(chaine);
      chaine = NULL;
   } while (retour == EXIT_SUCCESS);

   if (hFicTest != NULL)
   {
      (void)fclose(hFicTest);
   }

   return EXIT_SUCCESS;
} // int main(void)
```

test test_lire_ligne.txt

```
1
```

```
1234567880
ligne longue.

123456789012345678901234567890123456789012345678901234567890
fin
```

Solution proposée

Voir la solution

```c
/**
 * Fonction .. : lire_ligne
 * Role ..... : tente de lire une ligne entiere depuis le flux.
 * - Le caractere de saut de ligne '\n' final est enleve.
 * - L'appellant doit se charger de la desallocation de la chaine retournee.
 *
 * Parametres :
 * - pChaine : adresse de retour de la chaine
 * - tailleBufferLecture : taille du buffer de lecture,
 *   dans la pratique on pourra choisir _POSIX_MAX_INPUT (ref limits.h).
 * - flux : un pointeur sur le descripteur du fichier a lire.
 *
 * - Valeur retournee :
 *    - Si OK : EXIT_SUCCESS.
 *    - Si Fin de fichier atteinte sans rencontrer \n : EOF,
 *       - la chaine retournée contient le debut de la ligne longue.
 *       - Si rien a lire, retour d'un pointeur NULL pour la chaine.
 *       - En cas de probleme : Le code d'erreur systeme utilisable
 *         par perror et retour d'un pointeur NULL pour la chaine.
 *
 * Auteur ..... : Thierry46
 * Version ..... : 1.1 du 4/2/2008
 * Licence ..... : GNU GPL.
    lire_ligne : Lecture d'une ligne longue dans un fichier texte
    Copyright (C) 2008  Thierry46

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
 *
 * Compilation . : gcc -pedantic -Wall -std=c99 -c lire_ligne.c
 * Fichiers lies : main_lire_ligne.c, lire_ligne.h
 *
 * Remarques :
 * - Beaucoup de commentaires sont présents pour un usage didactique.
 * - Ce programme utilise une syntaxe specifique C99.
 * - Ce programme a ete controle avec l'outil de verification statique
 * - Splint v3.1.2 du 19 Jan 2008 : <http://www.splint.org/>
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
```

```c
#include <assert.h>
#include <errno.h>
#include "lire_ligne.h"

/*@null@*/ char *lire_ligne(size_t tailleBufferLecture,
        FILE * restrict flux, int *pCodeRetour)
{
   // Declarations
   char *chaineRetour = NULL;
   bool litEncore = true;
   bool rienLu = true;
   int codeRetour = EXIT_SUCCESS;

   // Controle rapide des parametres.
   // Si quelque chose ne va pas, arret.
   assert(tailleBufferLecture > 0);
   assert(flux != NULL);

   // On reinitialise errno
   errno = 0;

   // Boucle de lecture d'une ligne eventuellement longue
   do
   {
      size_t positionEcriture;
      char *nouvelleChaine;
      litEncore = false;

      // Demande d'une nouvelle zone memoire
      positionEcriture =
         (chaineRetour == NULL) ? 0 : strlen(chaineRetour);
      nouvelleChaine = realloc(chaineRetour,
         positionEcriture + tailleBufferLecture);
      if (nouvelleChaine == NULL) // Si probleme d'allocation
      {
         codeRetour = errno;
         free(chaineRetour);
         chaineRetour = NULL;
      }
      else // Si le systeme a accorde la memoire
      {
         chaineRetour = nouvelleChaine;
          if (fgets(chaineRetour+positionEcriture,
            (int)tailleBufferLecture, flux)
            != NULL)
         {
            // Recherche si un \n a ete lu en fin de chaine.
            char *positionNewLine = strrchr(chaineRetour, '\n');
            rienLu = false;
            if (positionNewLine != NULL)
            {
               // Suppression du caractere de fin de ligne \n,
               *positionNewLine = '\0';
            }
            else // fgets n'a pas pu lire la ligne complete.
            {
               litEncore = true;
            }
         } else if (ferror(flux) != 0)
         {
            codeRetour = errno;
            free(chaineRetour);
            chaineRetour = NULL;
         } else if (feof(flux) != 0)
         {
            codeRetour = EOF;
            if (rienLu)
            {
               free(chaineRetour);
```

```
                chaineRetour = NULL;
            }
        }
      }  // else if (nouvelleChaine == NULL)
   } while (litEncore);

   // Retour des resultats.
   *pCodeRetour = codeRetour;
   return chaineRetour;
} // char *lire_ligne(...
```

Remarques sur l'exercice

- Le test des paramètres dans la solution est expéditif.
- Pour obtenir des programmes robustes, le langage C oblige à une gestion pénible des erreurs. Avec le langage Java par exemple, les mécanismes d'exception facilitent la tâche du programmeur.
- L'utilisation de l'allocation dynamique de mémoire est risquée : fuite mémoire. Avec le langage Java par exemple, le ramasse miettes (Garbage collector) se charge de la libération de la mémoire.
- En Java des classes comme String et StringBuffer prennent en charge les chaines de caractères longues.

Test d'un générateur de nombre aléatoire

Problème à résoudre

La bibliothèque standard du langage C offre au programmeur plusieurs fonctions pour générer des nombres aléatoires. La plus connue est `rand()`.

Vous allez écrire un programme *verifrand.c* qui estime la répartition des nombres aléatoires générés : moyenne et dispersion. Nous allons traiter l'ensemble des nombres générés comme une série discrète regroupée.

1. Générez 1 million (NB_TIRAGE) notes (x_i) entre 0 et 20 (N) comprises (NB_NOTE = 21) à l'aide de la fonction `rand()`.
2. Répartissez-les dans le tableau effectif (n) où n_i représente l'effectif (nombre d'occurrences cumulées) de la note x_i.
3. calculez et affichez la moyenne arithmétique : $\overline{x} = \frac{\sum_{i=0}^{N}(n_i * x_i)}{\sum_{i=0}^{N} n_i} = \frac{\sum_{i=1}^{N}(n_i * i)}{NB\_TIRAGE}$
4. calculez et affichez l'écart moyen de la série : $Em = \frac{\sum_{i=0}^{N}(n_i * |x_i - \overline{x}|)}{\sum_{i=0}^{N} n_i} = \frac{\sum_{i=1}^{N}(n_i * |i - \overline{x}|)}{NB\_TIRAGE}$

Solution proposée

Voir la solution

```
/*
 Nom : verifrand.c
 Auteur : Thierry46
 Role : Verifier generateur de nombre aleatoire
 Licence : GNU GPL
```

```c
 Version : 1.0 du 17/2/2008
 Compilation : gcc -Wall -pedantic -std=c99 -o verifrand.exe verifrand.c
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//  NB_NOTE est appele N dans les formules
// 20 notes : de 0 à 20 inclus
#define NB_NOTE 21
#define NB_TIRAGE 1000000UL

int main(void)
{
   unsigned long effectif[NB_NOTE];// appele n dans les formules
   unsigned short note; // appelee i dans les formules, ici xi = i
   double moyenne; // appelee x barre dans les formules
   double ecartMoyen;
   unsigned long numTirage;
   unsigned short noteLaMoinsGeneree;
   unsigned short noteLaPlusGeneree;

   // Init a 0 du tableau des nombres d'occurence
   for (note=0; note<NB_NOTE; note++)
   {
      effectif[note] = 0;
   }

   // Generation des nombres aleatoires
   (void)printf("Generation et repartition %lu notes dans tableau effectif.\n",
      NB_TIRAGE);
   for (numTirage=0; numTirage<NB_TIRAGE; numTirage++)
   {
      effectif[rand()%NB_NOTE]++;
   }

   // Determination des notes les moins et les plus generees
   for (note=1, noteLaMoinsGeneree=0, noteLaPlusGeneree=0; note<NB_NOTE; note++)
   {
      if (effectif[note] < effectif[noteLaMoinsGeneree])
      {
         noteLaMoinsGeneree = note;
      }
      if (effectif[note] > effectif[noteLaPlusGeneree])
      {
         noteLaPlusGeneree = note;
      }
   }
   (void)printf("Note la moins generee : %hu (%lu fois)\n"
                "Note la plus generee : %hu (%lu fois)\n",
                noteLaMoinsGeneree, effectif[noteLaMoinsGeneree],
                noteLaPlusGeneree, effectif[noteLaPlusGeneree]);

   // Calcul de la moyenne de la série
   for (note=0, moyenne=0.0; note<NB_NOTE; note++)
   {
      moyenne += (double)effectif[note]*(double)note;
   }
   moyenne /= (double)NB_TIRAGE;
   (void)printf("La moyenne = %lf\n", moyenne);

   // Estimation de l'écart moyen de la série
   for (note=0, ecartMoyen=0; note<NB_NOTE; note++)
   {
      ecartMoyen += (double)effectif[note]*fabs((double)note - moyenne);
   }
   ecartMoyen /= (double)NB_TIRAGE;
   (void)printf("ecart moyen = %lf\n", ecartMoyen);
```

```
   return EXIT_SUCCESS;
} // int main(void)
```

Résultats obtenus :

```
MacMini-TM:~/Documents/c thierry$ ./verifrand.exe
Generation et repartition des 1000000 notes dans tableau effectif.
Note la moins generee : 14 (47263 fois)
Note la plus generee : 6 (48047 fois)
La moyenne = 9.991110
ecart moyen = 5.236995
```

Remarques sur l'exercice

- La série générée avec `rand()` sur ma machine est bien centrée sur la moyenne théorique 10/20 : les nombres de notes en dessous et au dessus de 10 sont quasi-identiques. L'écart moyen de la série est proche de 5 ce qui indique que les notes sont bien réparties.
- La fonction `rand()` est pseudo-aléatoire : deux exécutions du programme donnent exactement la même série de note.
- Vous pouvez tester d'autres fonctions comme `random()` ou `arc4random()`. Cette dernière, de qualité cryptographique, utilise l'algorithme nommé ARC4. Elle est aléatoire.

Chaines

Longueur d'une chaîne de caractères (pour manipuler)

Écrire une fonction C calculant la longueur en octets d'une chaîne de caractères, donnée en argument.

A titre d'exercice, pas utiliser la fonction strlen() du fichier d'include string.h.

Solution

La fonction doit recevoir une chaîne de caractères. Une chaîne de caractères en C est un tableau de caractères, finissant par le caractère nul. La fonction doit donc recevoir un pointeur vers le premier élément du tableau, de type `char *`, et chercher l'élément nul.

La fonction recevant un pointeur, il faut donc s'assurer qu'il est valide. On doit donc tester si le pointeur est nul ou non avant de regarder ce qui se trouve dans le tableau de caractères.

```
#include <stdlib.h>

int longueur(char *s)
{
    int n = -1;
    if (s != NULL)
    {
```

```
        n = 0;
        while (s[n] != '\0')
        {
            n++;
        }
    }
    return n;
}
```

Ici, on a choisi de retourner la valeur -1 si le pointeur est nul.

La boucle de recherche de l'élément nul peut être aussi écrite avec un `for`:

```
#include <stdlib.h>

int longueur(char *s)
{
    int n = -1;
    if (s != NULL)
    {
        for (n = 0; s[n] != '\0'; n++)
        {
            /* vide */
        }
    }
    return n;
}
```

Fichier source lg.c :

```
/*
Nom ....... : lg.c
Role ...... : Compte le nombre de caractères d'une chaine constan
Compilation : gcc -Wall -o lg.exe lg.c
Exécution . : ./lg.exe
*/
#include <stdio.h>
#include <stdlib.h>

int longueur(char *s)
{
        int n = -1;
        if (s != NULL)
        {
                n = 0;
                while (s[n] != '\0')
                {
                n++;
                }
        }
return n;
}

int main(void)
{
   int nb=0;
   char *C;
   C = "Chaine de caracteres constante";

   nb = longueur(C);

   (void)printf("Le nombre de caracteres est de : %d\n", nb);
   return EXIT_SUCCESS;
}
```

Résultats d'exécution :

```
MacMini-TM:~/Documents/developpement/c thierry$ ./lg.exe
Le nombre de caracteres est de : 30
```

Longueur d'une chaîne de caractères (par une fonction)

Écrire un programme *lgChaine.c* :

- qui lit des chaines de caractères tapées au clavier (flux stdin);
- qui calcule la longueur de chaque chaîne entrée et l'affiche ainsi que sa longueur;
- qui s'arrête si l'utilisateur ne frappe que la touche Entrée ou si le fichier est fini (Ctrl-D tapé par l'utilisateur).

Vous utiliserez :

- une des fonctions déclarée dans stdio.h.
- la fonction `strlen`, ainsi que d'autres si nécessaire, déclarée dans string.h.

Solution

```c
/*
Nom ..... : lgChaine.c
Auteur ... : Thierry46
Role ..... : Lit des chaines et affiche leur longueur.
Paramètres : non pris en compte.
Code retour : 0 (EXIT_SUCCESS)
Pour produire un executable avec le compilateur libre GCC :
   gcc -Wall -std=c99 -o lgChaine.exe lgChaine.c
Pour exécuter, tapez : ./lgChaine.exe

Version : 1.0 du 14/1/2008
Licence : GNU GPL
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>

int main(void)
{
        char reponse[_POSIX_MAX_INPUT];
        size_t lgReponse = 0;
        int codeRetour = EXIT_SUCCESS;
        char *posCr;
        char *chRetour;

        (void)puts("Entrez du texte puis la touche Entree,\n"
                "ou appuyez seulement sur la touche Entree pour a

        do
        {
                // Lecture de la chaine au clavier \n inclu
                chRetour = fgets(reponse, _POSIX_MAX_INPUT, stdin
                if (chRetour != NULL)
                {
                        // Suppression du \n final eventuel
                        posCr = strrchr(reponse, '\n');
```

Pointeurs

Soit un texte donné par une chaîne de caractères. Le but est de compter le nombre d'occurrences de chaque lettre sans distinction entre minuscules et majuscules.

Exercice 1

Question 1

Déclarer le texte comme un tableau statique initialisé par une chaîne de caractères constante, un tableau d'entiers statique pour compter les occurrences dont la taille est fixée par une constante et un pointeur pour parcourir le texte.

```
                        if (posCr != NULL)
                        {
                                *posCr = '\0';
                        }

                        // Calcule la longueur de la chaine
                        lgReponse = strlen(reponse);

                        if (lgReponse != 0)
                        {
                                (void)printf("Vous avez entre la
                                        "\t-\"%s\"\n"
                                        "\t- %zu caractere%s\n",
                                        reponse, lgReponse,
                                        (lgReponse>1) ? "s" : "")
                        }
                }
                else if (ferror(stdin))
                {
                        (void)fputs("Probleme de lecture.\n", std
                        codeRetour = EXIT_FAILURE;
                }
        } while ((chRetour != NULL) && (lgReponse != 0));

        return codeRetour;
} // int main(...
```

Remarques :

- L'utilisation de `fgets` est plus sûre que `gets` ou `scanf` : fgets permet de limiter le nombre de caractère (octet) lu et de ne pas déborder de la variable *reponse*. Les débordements mémoire sont une source d'erreur fréquente en C.
- Ce programme peut ne pas donner les résultats attendus si on redirige sur l'entrée standard un fichier comportant des terminateurs de ligne \r (Mac).
- La fonction `strlen` de string.h retourne une valeur de type size_t : type non signée donc toujours >=0, unsigned int ou unsigned long selon l'implémentation.

Solution

```
#include <stdio.h>

const int nb_lettres = 26;

int main(void)
{
  /* déclaration d'une chaîne <=> tableau de caractères. */
  char ch[]="ceci est une chaîne de test";
  printf("Chaîne en mémoire : %s\n",ch);

  /* déclaration d'un pointeur sur une chaîne de caracteres. */
  char *p = ch;

  /* déclaration d'un tableau de 26 cases correspondant
     aux lettres de l'alphabet, contenant les occurrences
     des lettres trouvées dans la chaîne.*/
  int occ[nb_lettres];
}
```

Question 2

Initialiser le vecteur d'entiers avec un parcours par indice.

Solution

```
/* initialisation du tableau des occurrences à 0. */
int i=0;
for (i=0; i<nb_lettres;++i)
  occ[i]=0;
```

Question 3

Compter les occurrences en utilisant la conversion entre le type char et le type int (la conversion d'un caractère donne son code dans le standard américain).

Solution

```
/* parcours de la chaîne.
   Caractère de fin d'une chaîne en C : '\0'. */
while (*p != '\0')
{
  if (*p >= 'a' && *p <= 'z')
  {
    ++occ[*p-'a'];
  }
  else if (*p >= 'A' && *p <= 'Z')
  {
    ++occ[*p-'A'];
  }
  /* incrémentation du pointeur <=> passage à la lettre suivante. */
  ++p;
}
```

Question 4

Afficher le résultat sur la sortie standard.

Solution

```
  for (i=0; i<nb_lettres; ++i)
  {
    printf("Nombre de %c : %i\n",
        'a'+i, /* le transtypage est automatique : 'a'+i renvoie un entier. */
        occ[i]);
  }

  return 0;
}
```

Exercice 2

Pointeurs et références

Donner et expliquer le résultat de l'exécution du programme suivant :

```c
#include <stdio.h>
#define taille_max 5

void parcours(int *tab)
{
  int *q=tab;
  do
  {
    printf("%d:%d\n", q-tab, *q-*tab);
  }
  while (++q-tab < taille_max);
}


void bizarre(int **copie, int *source)
{
  *copie=source;
}

int main(void)
{
  int chose[taille_max] = {1,3,2,4,5}, *truc;
  printf("chose : \n");
  parcours(chose);
  bizarre(&truc, chose);
  printf("truc : \n");
  parcours(truc);

  return 0;
}
```

Préprocesseur

Exercice 1

Écrire un fichier source hello.c. Les résultats du programme exécutable seront différents selon les options passées au préprocesseur sur la ligne de commande de compilation. Ce programme affichera :

- "Hello World", si la constante symbolique *WORLD* est définie.
- "Hello Fof", si *FOF* est définie.
- "Hello Nobody", si aucune de ces constantes n'est définie.

Lignes de Compilation avec gcc :

- gcc -D WORLD -o hello.exe hello.c
- gcc -D FOF -o hello.exe hello.c
- gcc -o hello.exe hello.c

Exécution : ./hello.exe

Solution

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

int main(void)
{
#if defined(WORLD)
   (void)puts("Hello World");
#elif defined(FOF)
   (void)puts("Hello Fof");
#else
   (void)puts("Hello Nobody\n");
#endif
   return EXIT_SUCCESS;
}
```