Chapitre 7 : Programmation en C **spé ISN – TS**

I. Présentation générale

1. Le langage C

Le langage C est un langage de bas niveau dans le sens où il permet l'accès à des données que manipulent les ordinateurs (bits, octets, adresses) et qui ne sont pas souvent disponibles à partir de langages évolués tels que Fortran, Pascal ou ADA. Le langage C a été conçu pour l'écriture de systèmes d'exploitation (plus de 90% du noyau du système UNIX est écrit en langage C).

2. Fichiers sources et fichiers exécutables

Les fichiers écrits en langage C (appelés *fichiers sources*) peuvent être créés avec n'importe quel éditeur de texte (comme Notepad++), et sont sauvegardés avec l'extension .c (*exemple.c*).

Les instructions contenues dans ces fichiers ne sont pas directement compréhensibles par la machine. Il faut donc les traduire en langage machine : c'est le rôle du compilateur, qui crée à partir du fichier *.c* un fichier binaire exécutable. Cet **exécutable** sera ensuite exécuté par la machine autant de fois que nécessaire.

Nous utiliserons le logiciel Code::Blocks pour éditer, compiler et exécuter les programmes.

3. Code minimal

Le code minimal pour un fichier source est le suivant :

```
#include <stdio.h>              // chargement de bibliothèques de fonctions

int main() {                    // fonction principale main
                                /* ... les instructions sont à écrire
                                à cet endroit */
    return 0 ;
}
```

stdio.h bibliothèque (ensemble d'instructions) permettant de gérer les échanges entre l'ordinateur et l'utilisateur

main() fonction principale, qui ne prend aucun argument en entrée

int main() la fonction main doit rendre un entier en dernière instruction

return 0 ; la fonction renvoie la valeur 0.

//... pour écrire des commentaires sur une ligne

/*...*/ pour écrire des commentaires sur plusieurs lignes

4. Types de variables

Un certain nombre de mots-clés sont prédéfinis dans le compilateur. Ceux-ci ne peuvent donc pas être utilisés par le programmeur comme noms de variables. Parmi ces mots-clés prédéfinis, on trouve les définitions des types de variables :

int	type entier → on peut préciser sa taille avec short ou long → on peut préciser s'il est signé (signed / unsigned) par défaut, un int est un signed int
char	chaîne de caractères en code ASCII
float	type réel
double	type réel, de précision plus importante que float
long double	type réel, de précision plus importante que double

II. Interaction avec l'utilisateur

1. Fonction printf

La fonction printf est une fonction permettant de faire afficher un message (en **sortie**) lors de l'exécution du programme. Par exemple :

```
printf("Bonjour") ;                 // affichera le message Bonjour
printf("%d divisé par %d vaut %f", 7, 2, 3.5) ;
                                    // affichera le message 7 divisé par 2 vaut 3.5
```

L'argument de la fonction printf est une chaîne de caractères, et peut contenir des séquences de contrôle permettant d'inclure des variables dans la sortie.

Dans le deuxième exemple, le premier %d est remplacé par 7, le deuxième %d par 2 et le %f par 3,5.

Les séquences de contrôle commencent par le caractère %, suivi d'une lettre indiquant le type de variable qui viendra se placer à cet endroit. Exemples :

d ou i	entier signé au format décimal
u	entier non signé au format décimal
o	entier signé au format octal
x	entier signé au format hexadécimal
f	réel
c	caractère
s	chaîne de caractères

2. Fonction scanf

La fonction scanf est une fonction permettant de lire des informations à partir de l'entrée standard (*i.e.* le clavier). Elle utilise les mêmes formats de variables que printf mais le nom de la variable doit être précédé du caractère & (esperluette).

Exemple :

```
int ma_variable                          // Déclaration de la variable ma_variable
printf("Entrez un nombre entier\n") ;
      // L'ajout de \n permet de faire un saut de ligne après la consigne
scanf("%d", &ma_variable) ;
      // Initialisation de ma_variable avec la valeur entrée au clavier
```

III. Premiers programmes

1. Erreurs lors de la compilation ou de l'exécution

Certaines erreurs de programmation rendent la compilation du fichier source impossible. Pour les éviter, voici quelques règles d'écriture :

- chaque instruction doit se terminer par un point virgule ;
- les instructions sont toujours groupées dans des blocs entre accolades.

La compilation détecte aussi les erreurs de parenthèses, et les incompatibilités de typage de variables.
Par ailleurs, de bonnes habitudes prises dès le départ permettent de rendre le code plus lisible, donc plus facile à déboguer :

- penser à toujours bien indenter le code ;
- donner des noms explicites aux variables et fonctions ;
- commenter dès que nécessaire ;
- ne pas insérer d'espace dans les noms de variables ou de fonctions (il serait interprété par le compilateur). Si besoin, utiliser une nomenclature comme *nomComplexe* ou *nom_complexe* ;
- ne pas hésiter à aérer le code : laisser par exemple de l'espace entre les parties entrée / traitement / sortie.

Voici les erreurs de compilation les plus fréquentes :

- error: expected ';' before... : il manque un point-virgule quelque part ;
- error: expected declaration or statement at end of input : il manque une accolade ;
- error: 'nom_variable' undeclared (first use in this function) : vous utilisez une variable qui n'a pas été préalablement déclarée. Soit il manque une déclaration, soit l'orthographe utilisé est différent entre la déclaration et l'utilisation.
- a.c:(.text+0x19): undefined reference to 'nom_fonction' : La fonction n'est pas reconnue par le compilateur. Soit elle n'a pas été définie, soit l'orthographe utilisé est différent entre la déclaration et l'utilisation.
- fatal error: nom fichier.h: Aucun fichier ou dossier de ce type : Problème d'orthographe du fichier d'en-tête à inclure.

D'autres types d'erreurs ne sont pas visibles à la compilation, mais peuvent apparaître au moment de l'exécution.
Exemple :

```
while (i = 0) {
      ...
}
```

Ce code peut provoquer une boucle infinie puisque l'instruction passée en paramètre de la boucle while n'est pas une comparaison mais une affectation.

2. Structure de boucles

```
for (i = 0 ; i < n ; i = i + 1) {
      ... ;
}
```

```
while (i = 0) {
      ... ;
}
```

```
if (i == 0) {
      ... ;
}
else if (i == 1) {
      ... ;
}
else {
      ... ;
}
```

```
switch (n) {
      case 1 : ... ; break ;
      case 2 : ... ; break ;
      case 100 : ... ; break ;
      default : ... ; break ;
}
```

3. Exercices

1. Écrire un programme permettant d'afficher *Hello World !* à l'écran.
2. Écrire un programme demandant un nombre entier à l'utilisateur et affichant le double de ce nombre en sortie.
3. Écrire un programme demandant deux nombres entiers à l'utilisateur puis affichant la moyenne de ces deux nombres.
Tester pour {4 ; 8} et pour {3 ; 6}. Que constate-t-on ?
Modifier le programme pour que la moyenne soit toujours exacte.
4. Écrire un programme calculant la surface d'un triangle dont on aura demandé la hauteur et la base à l'utilisateur.
5. Écrire un programme permettant de permuter les valeurs contenues dans deux variables *a* et *b*.
Modifier le programme pour qu'il réalise la permutation circulaire de *a*, *b* et *c*.
6. Écrire un programme permettant de calculer la somme $\sum_{i=1}^{n} i^2$.
7. Écrire un programme permettant de préciser, selon la moyenne obtenue au baccalauréat (nombre réel demandé à l'utilisateur), s'il est recalé, au rattrapage, ou admis.
8. Écrire un programme permettant de déterminer le plus petit *n* tel que 2^n est supérieur à un seuil précisé par l'utilisateur.

Chapitre 7 : Programmation en C **spé ISN – TS**

3. Exercices

1. Écrire un programme permettant d'afficher *Hello World !* à l'écran.
2. Écrire un programme demandant un nombre entier à l'utilisateur et affichant le double de ce nombre en sortie.
3. Écrire un programme demandant deux nombres entiers à l'utilisateur puis affichant la moyenne de ces deux nombres.
Tester pour {4 ; 8} et pour {3 ; 6}. Que constate-t-on ?
Modifier le programme pour que la moyenne soit toujours exacte.
4. Écrire un programme calculant la surface d'un triangle dont on aura demandé la hauteur et la base à l'utilisateur.
5. Écrire un programme permettant de permuter les valeurs contenues dans deux variables *a* et *b*.
Modifier le programme pour qu'il réalise la permutation circulaire de *a*, *b* et *c*.
6. Écrire un programme permettant de calculer la somme $\sum_{i=1}^{n} i^2$.
7. Écrire un programme permettant de préciser, selon la moyenne obtenue au baccalauréat (nombre réel demandé à l'utilisateur), s'il est recalé, au rattrapage, ou admis.
8. Écrire un programme permettant de déterminer le plus petit *n* tel que 2^n est supérieur à un seuil précisé par l'utilisateur.

Chapitre 7 : Programmation en C Page 3 / 3