Introduction à la programmation C

Benoît Darties

Université de Bourgogne [1]
LE2I - Laboratoire Electronique Informatique Image

contact: benoit.darties@u-bourgogne.fr

1. Supports de cours dispensés à l'Université de Bourgogne. La totalité de ce document a été rédigée uniquement à partir des connaissances de son auteur, et en utilisant un matériel personnel. Réutilisation partielle ou complète de ce document soumise à approbation l'auteur

Ce que contient ce cours ...

Contenu du cours

- Bases du langage C : premiers programmes
- Rudiments de compilation
- Comment définir une structure à partir de types simples
- Notions de fonctions, différences avec les procédures

Table des matières

Un premier programme

Fichier programme "helloWorld.c"

```
1  #include <stdio.h>
2
3  int main() {
4      // affichage de ligne (ceci est un commentaire)
5      printf("Hello World!\n");
6      return 0;
7  }
```

- Inclusion de bibliothèques de fonctions, ici `stdio.h`
- Fonction principale : `int main() {...}`
- Afficher un message : fonction `printf()`
- Afficher un retour chariot : caractère spécial `\n`
- Retour de la valeur 0 : terminaison normale du programme
- Chaque instruction doit se terminer par un point-virgule
- ligne en commentaire en la précédant de //

Un premier programme

Fichier programme "helloWorld.c"

```
1  #include <stdio.h>
2
3  int main() {
4      // affichage de ligne (ceci est un commentaire)
5      printf("Hello World!\n");
6      return 0;
7  }
```

- Inclusion de bibliothèques de fonctions, ici `stdio.h`
- Fonction principale : `int main() {...}`
- Afficher un message : fonction `printf()`
- Afficher un retour chariot : caractère spécial `\n`
- Retour de la valeur 0 : terminaison normale du programme
- Chaque instruction doit se terminer par un point-virgule
- ligne en commentaire en la précédant de //

Un premier programme

Fichier programme "`helloWorld.c`"

```
1  #include <stdio.h>
2
3  int main() {
4      // affichage de ligne (ceci est un commentaire)
5      printf("Hello World!\n");
6      return 0;
7  }
```

- Inclusion de bibliothèques de fonctions, ici `stdio.h`
- Fonction principale : `int main() {...}`
- Afficher un message : fonction `printf()`
- Afficher un retour chariot : caractère spécial `\n`
- Retour de la valeur 0 : terminaison normale du programme
- Chaque instruction doit se terminer par un point-virgule
- ligne en commentaire en la précédant de //

Un premier programme

Fichier programme "`helloWorld.c`"

```
1  #include <stdio.h>
2
3  int main() {
4      // affichage de ligne (ceci est un commentaire)
5      printf("Hello World!\n");
6      return 0;
7  }
```

- Inclusion de bibliothèques de fonctions, ici `stdio.h`
- Fonction principale : `int main() {...}`
- Afficher un message : fonction `printf()`
- Afficher un retour chariot : caractère spécial `\n`
- Retour de la valeur 0 : terminaison normale du programme
- Chaque instruction doit se terminer par un point-virgule
- ligne en commentaire en la précédant de //

Un premier programme

Fichier programme "`helloWorld.c`"

```
1  #include <stdio.h>
2
3  int main() {
4      // affichage de ligne (ceci est un commentaire)
5      printf("Hello World!\n");
6      return 0;
7  }
```

- Inclusion de bibliothèques de fonctions, ici `stdio.h`
- Fonction principale : `int main() {...}`
- Afficher un message : fonction `printf()`
- Afficher un retour chariot : caractère spécial `\n`
- Retour de la valeur 0 : terminaison normale du programme
- Chaque instruction doit se terminer par un point-virgule
- ligne en commentaire en la précédant de //

Un premier programme

Fichier programme "`helloWorld.c`"

```
1  #include <stdio.h>
2
3  int main() {
4      // affichage de ligne (ceci est un commentaire)
5      printf("Hello World!\n");
6      return 0;
7  }
```

- Inclusion de bibliothèques de fonctions, ici `stdio.h`
- Fonction principale : `int main() {...}`
- Afficher un message : fonction `printf()`
- Afficher un retour chariot : caractère spécial `\n`
- Retour de la valeur 0 : terminaison normale du programme
- Chaque instruction doit se terminer par un point-virgule
- ligne en commentaire en la précédant de //

Un premier programme

Fichier programme "helloWorld.c"

```
1  #include <stdio.h>
2
3  int main() {
4      // affichage de ligne (ceci est un commentaire)
5      printf("Hello World!\n");
6      return 0;
7  }
```

- Inclusion de bibliothèques de fonctions, ici `stdio.h`
- Fonction principale : `int main() {...}`
- Afficher un message : fonction `printf()`
- Afficher un retour chariot : caractère spécial `\n`
- Retour de la valeur 0 : terminaison normale du programme
- Chaque instruction doit se terminer par un point-virgule
- ligne en commentaire en la précédant de //

Un premier programme

Fichier programme "`helloWorld.c`"

```
1  #include <stdio.h>
2
3  int main() {
4      // affichage de ligne (ceci est un commentaire)
5      printf("Hello World!\n");
6      return 0;
7  }
```

- Inclusion de bibliothèques de fonctions, ici `stdio.h`
- Fonction principale : `int main() {...}`
- Afficher un message : fonction `printf()`
- Afficher un retour chariot : caractère spécial `\n`
- Retour de la valeur 0 : terminaison normale du programme
- Chaque instruction doit se terminer par un point-virgule
- ligne en commentaire en la précédant de //

Un premier programme

Fichier `helloWorld.c` : code source

- Code écrit, interprétable et modifiable par un être humain
- N'est pas interprétable par un ordinateur
- Besoin d'être *compilé* pour être exécuté par un ordinateur

Opération de *compilation*

- Transformation d'un code source en un programme exécutable
- Ecriture en langage binaire compréhensible par un ordinateur
- Nécessité d'un compilateur C, par exemple `gcc` (référence)

Compilation du programme

Appel du compilateur `gcc` pour compilation

commande pour compilation :

```
gcc -o programmeHello helloWorld.c
```

- option `-o` *`filename`* : nom de l'exécutable créé
- Si pas de nom donné : `a.out` par défaut
- Programme binaire exécutable

Exécution du programme

Lancement de l'exécutable `programmeHello`

commande pour exécution :

```
./programmeHello
```

- `./` emplacement du fichier exécutable
- Droits d'exécution `+x` automatiquement ajoutés

Résultat de l'exécution :

```
Prompt> ./programmeHello
Hello World!
```

Apercu du fichier `programmeHello` (1/4)

```
00000000  cf fa ed fe 07 00 00 01  03 00 00 80 02 00 00 00  |................|
00000010  0f 00 00 00 00 05 00 00  85 00 20 00 00 00 00 00  |.......... .....|
00000020  19 00 00 00 48 00 00 00  5f 5f 50 41 47 45 5a 45  |....H...__PAGEZE|
00000030  52 4f 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |RO..............|
00000040  00 00 00 00 01 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  19 00 00 00 28 02 00 00  |............(...|
00000070  5f 5f 54 45 58 54 00 00  00 00 00 00 00 00 00 00  |__TEXT..........|
00000080  00 00 00 00 01 00 00 00  00 10 00 00 00 00 00 00  |................|
00000090  00 00 00 00 00 00 00 00  00 10 00 00 00 00 00 00  |................|
000000a0  07 00 00 00 05 00 00 00  06 00 00 00 00 00 00 00  |................|
000000b0  5f 5f 74 65 78 74 00 00  00 00 00 00 00 00 00 00  |__text..........|
000000c0  5f 5f 54 45 58 54 00 00  00 00 00 00 00 00 00 00  |__TEXT..........|
000000d0  40 0f 00 00 01 00 00 00  2a 00 00 00 00 00 00 00  |@.......*.......|
000000e0  40 0f 00 00 04 00 00 00  00 00 00 00 00 00 00 00  |@...............|
000000f0  00 04 00 80 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000100  5f 5f 73 74 75 62 73 00  00 00 00 00 00 00 00 00  |__stubs.........|
00000110  5f 5f 54 45 58 54 00 00  00 00 00 00 00 00 00 00  |__TEXT..........|
00000120  6a 0f 00 00 01 00 00 00  06 00 00 00 00 00 00 00  |j...............|
00000130  6a 0f 00 00 01 00 00 00  00 00 00 00 00 00 00 00  |j...............|
00000140  08 04 00 80 00 00 00 00  06 00 00 00 00 00 00 00  |................|
00000150  5f 5f 73 74 75 62 5f 68  65 6c 70 65 72 00 00 00  |__stub_helper...|
00000160  5f 5f 54 45 58 54 00 00  00 00 00 00 00 00 00 00  |__TEXT..........|
00000170  70 0f 00 00 01 00 00 00  1a 00 00 00 00 00 00 00  |p...............|
00000180  70 0f 00 00 02 00 00 00  00 00 00 00 00 00 00 00  |p...............|
00000190  00 04 00 80 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001a0  5f 5f 63 73 74 72 69 6e  67 00 00 00 00 00 00 00  |__cstring.......|
000001b0  5f 5f 54 45 58 54 00 00  00 00 00 00 00 00 00 00  |__TEXT..........|
000001c0  8a 0f 00 00 01 00 00 00  0e 00 00 00 00 00 00 00  |................|
000001d0  8a 0f 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Apercu du fichier a.out (2/4)

```
000001e0  02 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001f0  5f 5f 75 6e 77 69 6e 64  5f 69 6e 66 6f 00 00 00  |__unwind_info...|
00000200  5f 5f 54 45 58 54 00 00  00 00 00 00 00 00 00 00  |__TEXT..........|
00000210  98 0f 00 00 01 00 00 00  48 00 00 00 00 00 00 00  |........H.......|
00000220  98 0f 00 00 02 00 00 00  00 00 00 00 00 00 00 00  |................|
00000230  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000240  5f 5f 65 68 5f 66 72 61  6d 65 00 00 00 00 00 00  |__eh_frame......|
00000250  5f 5f 54 45 58 54 00 00  00 00 00 00 00 00 00 00  |__TEXT..........|
00000260  e0 0f 00 00 01 00 00 00  18 00 00 00 00 00 00 00  |................|
00000270  e0 0f 00 00 03 00 00 00  00 00 00 00 00 00 00 00  |................|
00000280  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000290  19 00 00 00 e8 00 00 00  5f 5f 44 41 54 41 00 00  |........__DATA..|
000002a0  00 00 00 00 00 00 00 00  00 10 00 00 01 00 00 00  |................|
000002b0  00 10 00 00 00 00 00 00  00 10 00 00 00 00 00 00  |................|
000002c0  00 10 00 00 00 00 00 00  07 00 00 00 03 00 00 00  |................|
000002d0  02 00 00 00 00 00 00 00  5f 5f 6e 6c 5f 73 79 6d  |........__nl_sym|
000002e0  62 6f 6c 5f 70 74 72 00  5f 5f 44 41 54 41 00 00  |bol_ptr.__DATA..|
000002f0  00 00 00 00 00 00 00 00  00 10 00 00 01 00 00 00  |................|
00000300  10 00 00 00 00 00 00 00  00 10 00 00 03 00 00 00  |................|
00000310  00 00 00 00 00 00 00 00  06 00 00 00 01 00 00 00  |................|
00000320  00 00 00 00 00 00 00 00  5f 5f 6c 61 5f 73 79 6d  |........__la_sym|
00000330  62 6f 6c 5f 70 74 72 00  5f 5f 44 41 54 41 00 00  |bol_ptr.__DATA..|
00000340  00 00 00 00 00 00 00 00  10 10 00 00 01 00 00 00  |................|
00000350  08 00 00 00 00 00 00 00  10 10 00 00 03 00 00 00  |................|
00000360  00 00 00 00 00 00 00 00  07 00 00 00 03 00 00 00  |................|
00000370  00 00 00 00 00 00 00 00  19 00 00 00 48 00 00 00  |............H...|
00000380  5f 5f 4c 49 4e 4b 45 44  49 54 00 00 00 00 00 00  |__LINKEDIT......|
00000390  00 20 00 00 01 00 00 00  00 10 00 00 00 00 00 00  |. ..............|
000003a0  00 20 00 00 00 00 00 00  f0 00 00 00 00 00 00 00  |. ..............|
000003b0  07 00 00 00 01 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Apercu du fichier a.out (3/4)

```
000003c0  22 00 00 80 30 00 00 00  00 20 00 00 08 00 00 00  |"...0.... ......|
000003d0  08 20 00 00 18 00 00 00  00 00 00 00 00 00 00 00  |. ..............|
000003e0  20 20 00 00 10 00 00 00  30 20 00 00 30 00 00 00  |  ......0 ..0...|
000003f0  02 00 00 00 18 00 00 00  68 20 00 00 04 00 00 00  |........h ......|
00000400  b8 20 00 00 38 00 00 00  0b 00 00 00 50 00 00 00  |. ..8.......P...|
00000410  00 00 00 00 00 00 00 00  00 00 00 00 02 00 00 00  |................|
00000420  02 00 00 00 02 00 00 00  00 00 00 00 00 00 00 00  |................|
00000430  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000440  a8 20 00 00 04 00 00 00  00 00 00 00 00 00 00 00  |. ..............|
00000450  00 00 00 00 00 00 00 00  0e 00 00 00 20 00 00 00  |............ ...|
00000460  0c 00 00 00 2f 75 73 72  2f 6c 69 62 2f 64 79 6c  |..../usr/lib/dyl|
00000470  64 00 00 00 00 00 00 00  1b 00 00 00 18 00 00 00  |d...............|
00000480  e4 38 5c 6a c8 88 38 8c  b1 fe 7d b7 c3 fc 71 18  |.8\j..8...}...q.|
00000490  24 00 00 00 10 00 00 00  00 0b 0a 00 00 0b 0a 00  |$...............|
000004a0  2a 00 00 00 10 00 00 00  00 00 00 00 00 00 00 00  |*...............|
000004b0  28 00 00 80 18 00 00 00  40 0f 00 00 00 00 00 00  |(.......@.......|
000004c0  00 00 00 00 00 00 00 00  0c 00 00 00 38 00 00 00  |............8...|
000004d0  18 00 00 00 02 00 00 00  01 01 c9 04 00 00 01 00  |................|
000004e0  2f 75 73 72 2f 6c 69 62  2f 6c 69 62 53 79 73 74  |/usr/lib/libSyst|
000004f0  65 6d 2e 42 2e 64 79 6c  69 62 00 00 00 00 00 00  |em.B.dylib......|
00000500  26 00 00 00 10 00 00 00  60 20 00 00 08 00 00 00  |&.......` ......|
00000510  29 00 00 00 10 00 00 00  68 20 00 00 00 00 00 00  |).......h ......|
00000520  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000f40  55 48 89 e5 48 83 ec 10  48 8d 3d 3b 00 00 00 c7  |UH..H...H.=;....|
00000f50  45 fc 00 00 00 00 b0 00  e8 0d 00 00 00 31 c9 89  |E............1..|
00000f60  45 f8 89 c8 48 83 c4 10  5d c3 ff 25 a0 00 00 00  |E...H...]..%....|
00000f70  4c 8d 1d 91 00 00 00 41  53 ff 25 81 00 00 00 90  |L......AS.%.....|
00000f80  68 00 00 00 00 e9 e6 ff  ff ff 48 65 6c 6c 6f 20  |h.........Hello |
00000f90  57 6f 72 6c 64 21 0a 00  01 00 00 00 1c 00 00 00  |World!..........|
```

Apercu du fichier a.out (4/4)

```
00000fa0  00 00 00 00 1c 00 00 00  00 00 00 00 1c 00 00 00  |................|
00000fb0  02 00 00 00 40 0f 00 00  34 00 00 00 34 00 00 00  |....@...4...4...|
00000fc0  6b 0f 00 00 00 00 00 00  34 00 00 00 03 00 00 00  |k.......4.......|
00000fd0  0c 00 01 00 10 00 01 00  00 00 00 00 00 00 00 01  |................|
00000fe0  14 00 00 00 00 00 00 00  03 7a 52 00 01 78 10 01  |.........zR..x..|
00000ff0  10 0c 07 08 90 01 00 00  00 00 00 00 00 00 00 00  |................|
00001000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00001010  80 0f 00 00 01 00 00 00  00 00 00 00 00 00 00 00  |................|
00001020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00002000  11 22 10 51 00 00 00 00  11 40 64 79 6c 64 5f 73  |."".Q.....@dyld_s|
00002010  74 75 62 5f 62 69 6e 64  65 72 00 51 72 00 90 00  |tub_binder.Qr...|
00002020  72 10 11 40 5f 70 72 69  6e 74 66 00 90 00 00 00  |r..@_printf.....|
00002030  00 01 5f 00 05 00 02 5f  6d 68 5f 65 78 65 63 75  |.._...._mh_execu|
00002040  74 65 5f 68 65 61 64 65  72 00 21 6d 61 69 6e 00  |te_header.!main.|
00002050  25 02 00 00 00 03 00 c0  1e 00 00 00 00 00 00 00  |%...............|
00002060  c0 1e 00 00 00 00 00 00  02 00 00 00 0f 01 10 00  |................|
00002070  00 00 00 00 01 00 00 00  16 00 00 00 0f 01 00 00  |................|
00002080  40 0f 00 00 01 00 00 00  1c 00 00 00 01 00 00 01  |@...............|
00002090  00 00 00 00 00 00 00 00  24 00 00 00 01 00 00 01  |........$.......|
000020a0  00 00 00 00 00 00 00 00  02 00 00 00 03 00 00 00  |................|
000020b0  00 00 00 40 02 00 00 00  20 00 5f 5f 6d 68 5f 65  |...@.... .__mh_e|
000020c0  78 65 63 75 74 65 5f 68  65 61 64 65 72 00 5f 6d  |xecute_header._m|
000020d0  61 69 6e 00 5f 70 72 69  6e 74 66 00 64 79 6c 64  |ain._printf.dyld|
000020e0  5f 73 74 75 62 5f 62 69  6e 64 65 72 00 00 00 00  |_stub_binder....|
000020f0
```

Table des matières

Notion de variables

Un programme manipule des variables

- Une variable est une entité qui contient une valeur
- Variable stockée en mémoire durant l'exécution du programme
- Occupe un espace **fixe** en mémoire
- doit posséder un **type**, et un **identificateur**

Quatre types de bases pour les variables

- `int` : pour stocker des entiers
- `float` ou `double` : pour stocker des nombres décimaux
- `char` : pour stocker un caractère

Notion de variables

Identificateur d'une variable

- = Nom de la variable
- Ne contient que des caractères alphanumériques, ou _
- Doit commencer par une lettre
- Pas de caractères spéciaux

Avant d'utiliser une variable , besoin de la déclarer

- Syntaxe : *type identificateur*

Une fois déclarée, affectation d'une valeur

- Opérateur =
- Syntaxe : *identificateur = valeur*

Affectation de variables

Fichier programme "affectationVariables.c"

```c
#include <stdio.h>

int main() {
    // déclaration d'une variable a
    int a;

    // affectation de a avec la valeur 6
    a=6;

    // déclaration et affectation sur une ligne
    char b = 'R';

    // déclaration de plusieurs variables une ligne
    int c, d , d2 ;

    // déclaration et affectation de plusieurs variables une ligne
    float e1, e2=40.2 , e3 = 5.0, f ;

    // affectation d'une variable depuis une autre
    c=a;  // c contient désormais la valeur 6

    // affectation en cascade (moins connu)
    d2=d=c;  // d2, d et c contiennent désormais la valeur 6

    return 0;
}
```

Question

Complétez ce programme :

- Inversez les valeurs des variables `castor` et `pollux`.
- Les lignes que vous rajoutez doivent rester inchangées même si les valeurs initiales sont changées (n'utilisez pas de chiffres)
- Vous pouvez rajouter des variables si vous le souhaitez

Fichier programme "`inversion_exo.c`"

```
1  #include <stdio.h>
2
3  int main() {
4      int castor = 50;
5      int pollux = 70;
6
7
8
9
10
11
12
13     return 0;
14 }
```

Question

Complétez ce programme :

- Inversez les valeurs des variables `castor` et `pollux`.
- Les lignes que vous rajoutez doivent rester inchangées même si les valeurs initiales sont changées (n'utilisez pas de chiffres)
- Vous pouvez rajouter des variables si vous le souhaitez

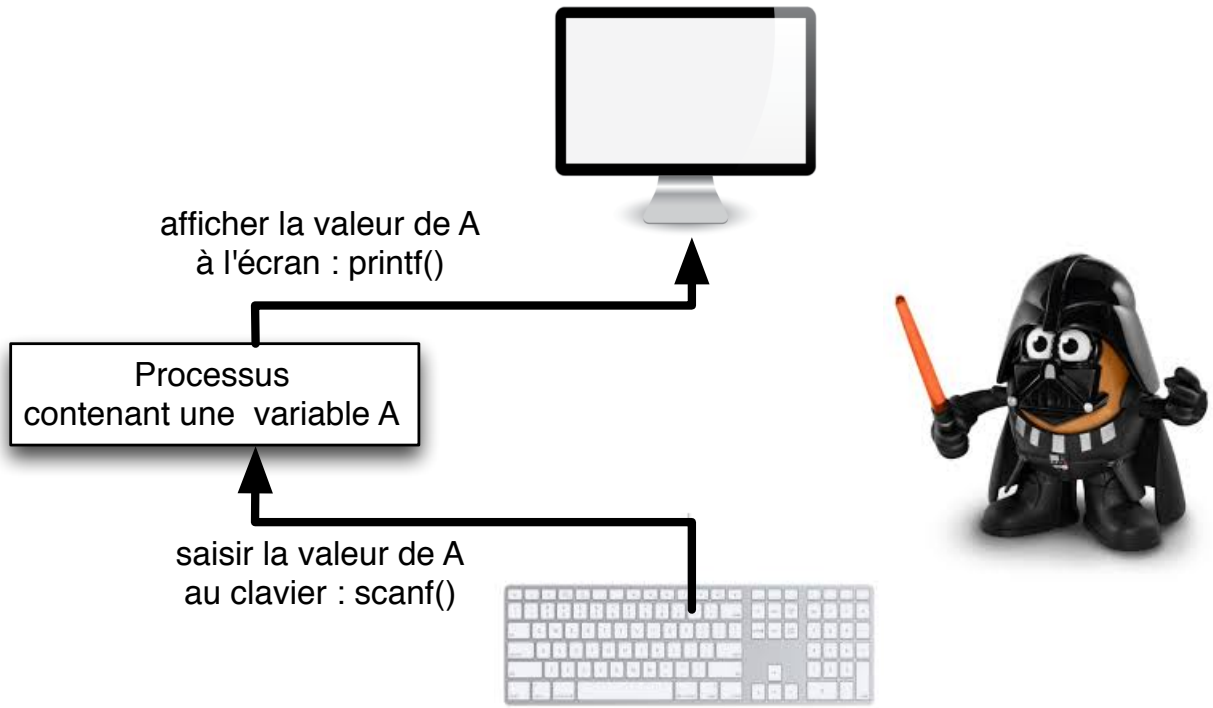
Fichier programme "`inversion_exo.c`"

```
1  #include <stdio.h>
2
3  int main() {
4      int castor = 50;
5      int pollux = 70;
6
7      int temp;          // utilisation d'une variable temporaire
8
9      temp = castor;     // temp contient désormais 50
10     castor = pollux;   // castor contient désormais 70
11     pollux = temp;     // pollux contient désormais 50
12
13     return 0;
14 }
```

Opérations usuelles sur les variables

Table des matières

Interaction avec l'utilisateur



- Affichage de la variable = afficher une valeur
- Saisie de sa valeur = écrire dans une zone mémoire
- Nom de la variable : désigne sa valeur, ou la zone mémoire

affichage et saisie de variables au clavier

Afficher la valeur d'une variable à l'écran

Fonction `printf` avec paramètres et spécificateurs de chaînes

- Spécificateur = suite de caractères spéciaux qui sera remplacé à l'affichage par une valeur passée en paramètre (à la suite)
- i-ème spécificateur remplacé par la valeur du i-ème paramètre
- Exemples de spécificateurs
  - %d : afficher la valeur sous format numérique
  - %c : afficher la valeur sous format caractère
  - %s : afficher la valeur sous format mot (chaine de caractères)
  - %f : afficher la valeur sous format décimal (flottant)

```
1  int var1;
2  char var2;
3  printf("valeur de la variable entiere  var1 = %d  \n", var1);
4  printf("valeur de la variable caractère var2 = %c  \n", var2);
5  printf("valeur des variables var2  = %c et var1 =%d  \n", var2, var1);
```

affichage et saisie de variables au clavier

Saisir la valeur d'une variable au clavier

Fonction `scanf` avec paramètres et spécificateurs de chaînes

- Spécificateur = suite de caractères spéciaux indiquant le format de la valeur à lire, pour affectation aux variables
- i-ème spécificateur remplacé par la valeur du i-ème paramètre
- Exemples de spécificateurs
  - %d : afficher la valeur sous format numérique
  - %c : afficher la valeur sous format caractère
  - %s : afficher la valeur sous format mot (chaine de caractères)
  - %f : afficher la valeur sous format décimal (flottant)

```
1  int var1;
2  char var2;
3  printf("valeur de la variable entiere  var1 = %d  \n", var1);
4  printf("valeur de la variable caractère var2 = %c  \n", var2);
5  printf("valeur des variables var2  = %c et var1 =%d  \n", var2, var1);
```

Saisie de variables au clavier et affichage

Fichier programme "printf_scanf.c"

```c
#include <stdio.h>

int main() {
    int a, b;
    char c;

    // saisie d'un entier et affichage
    printf("Saisissez un entier et appuyez sur entree\n");
    scanf("%d", &a); // besoin de & pour modification de a
    printf("l'entier que vous avez saisi est %d\n", a);

    // saisie conjointe d'un entier et d'un caractère
    printf("Saisissez maintenant un entier et un caractere en les espacant\n");
    scanf("%d %c", &b, &c);
    printf("vous avez rentre les valeurs %d et %c\n", b, c);

    return 0;
}
```

Table des matières

Conditions d'exécution

Possibilité d'exécuter ou non des instructions en fonction d'un test

- Si le test est validé : les instructions sont exécutées
- Si le test est non validé : les instructions sont ignorées

Definition d'un test : mot clé `if` (si) suivi d'une expression à tester

- Test à effectuer entre parenthèses
- Instructions délimitées par `{...}`
- : Bonne pratique : indenter les instructions conditionnées

```
if (test a effectuer) {
  instruction1;
  instruction2;
  instruction3;
}
```

Conditions d'exécution

Définition d'un test

- Comparaison entre variables : `<`, `>`, `<=`, `>=`, `==` , `!=` , `!`
- Evaluation d'une comparaison ou d'une opération d'affectation
- valeur 0 (ou NULL) : test échoué
- valeur différente de 0 : test réussi

Possibilité de faire plusieurs tests successifs dans un test

- `&&` : opérateur ET - les deux tests doivent réussir. Si le premier échoue, le second n'est meme pas regardé.
- `||` : opérateur OU - l'un des deux tests doivent réussir. Si le premier réussi, le second n'est meme pas regardé.
- Imbrication de tests possibles avec des parenthèses `(...)`

Conditions d'exécution

Définition d'un test

- Comparaison entre variables : `<`, `>`, `<=`, `>=`, `==` , `!=` , `!`
- Evaluation d'une comparaison ou d'une opération d'affectation
- valeur 0 (ou NULL) : test échoué
- valeur différente de 0 : test réussi

Possibilité de faire plusieurs tests successifs dans un test

- `&&` : opérateur ET - les deux tests doivent réussir. Si le premier échoue, le second n'est meme pas regardé.
- `||` : opérateur OU - l'un des deux tests doivent réussir. Si le premier réussi, le second n'est meme pas regardé.
- Imbrication de tests possibles avec des parenthèses `(...)`

Exemple de test conditionnel simple

Fichier programme "testSimple.c"

```c
#include <stdio.h>

int main() {
    int a, b;

    // saisie de deux entiers et comparaison de valeurs
    printf("Saisissez un entier et appuyez sur entree\n");
    scanf("%d", &a); // besoin de & pour modification de a

    printf("Saisissez un second entier et appuyez sur entree\n");
    scanf("%d", &b); // besoin de & pour modification de b

    if (a > b) {
        printf ("la premiere valeur entree est plus grande que la seconde\n");
    }

    if (a < b) {
        printf ("la seconde valeur entree est plus petite que la seconde\n");
    }

    if (a ==  b) {
        printf ("les deux valeurs entrees sont identiques\n");
    }
    return 0;
}
```

Exemple de test conditionnel simple

Fichier programme "testSimple2.c"

```c
#include <stdio.h>

int main() {
    int a, b;

    // saisie de deux entiers et comparaison de valeurs
    printf("Saisissez un entier et appuyez sur entree\n");
    scanf("%d", &a); // besoin de & pour modification de a

    printf("Saisissez un second entier et appuyez sur entree\n");
    scanf("%d", &b); // besoin de & pour modification de b

    if ((a > 0) && (b > 0)) {
        printf ("Les deux valeurs entrees sont strictement positives\n");
    }
    if ((a < 0) && (b < 0)) {
        printf ("Les deux valeurs entrees sont strictement negatives\n");
    }

    if (((a >0 ) && (b<0)) || ((a <0 ) && (b>0))) {
        printf ("les deux valeurs entrees ont des signes differents\n");
    }
    return 0;
}
```

Tests conditionnels un peu moins simples

Question

Quels sont les tests qui réussissent ?

Fichier programme "testsMoinsSimples.c"

```c
int main() {
    int a = 60 , b = 85;

    if ((2*a < b) || ((4*b + 40 < 8*a) && ((b+5) >= (a + 120)/2 ) )  ) {
        printf ("Premier test reussi\n");
    }

    if ((a < 130) && (a > 20)  && (3*b > 140)  && (2*b < 130)  ) {
        printf ("Second test reussi\n");
    }

    if ((a < 50 || b ) && ( 2*b> 3*a || b+5 < 100 ) ) {
        printf ("Troisième test reussi\n");
    }

    if (a = b) {
        printf ("Quatrieme test reussi\n");
    }
    return 0;
}
```

Conditions d'exécution

Instructions si le test échoue : mot clé `if ... else` (si ... sinon)

- Ajoute des instructions si le test échoue
- Instructions délimitées par `{...}`
- : Bonne pratique : indenter les instructions conditionnées

```
if (test a effectuer) {
  instruction1; // si le test réussit
  instruction2;
}
else {
  instruction3;  // si le test échoue
  instruction4;
}
```

Conditions d'exécution

Instructions si le test échoue : mot clé `if ... else` (si ... sinon)

Fichier programme "`boucle_entiers.c`"

```c
#include <stdio.h>

int main() {
    int somme=0; // initialisation de la somme
    int borne;
    int i;
    printf("saisissez une valeur :\n");
    scanf("%d", &borne);

    for (i =1; i <= borne; i++) {
        somme = somme + i;
        printf("au tour de boucle %d, somme vaut %d\n", i, somme);
    }

    printf("la somme des entiers de 1 à %d est %d\n", borne, somme);

    return 0;
}
```

Table des matières

Itérations d'instructions

Possibilité de répéter plusieurs fois une suite d'instructions

- Itérations = répétitions = boucles sur instructions
- Deux sortes d'itérations :
  - **déterministes** : nombre de répétitions connues à l'avance
  - **non déterministes** : nombre de répétitions inconnues, déterminées en live lors de l'exécution du programme

Itérations déterministes : boucle `for`

Définition d'une itération déterministe : mot-clé `for` (pour)

```
for ( init; condition;  increm.) {
  instruction1;
  instruction2;
  instruction3;
}
```

- init : instruction d'initialisation : (ex : init. d'un compteur)
- condition : tant que cette condition est vrai, répétition
- incrém. : façon d'incrémenter le compteur chaque tour

```c
int main() {
    int i;
    for (i = 0; i < 10; i++) {
        printf("tour de boucle numero : %d \n", i);
    }
    return 0;
}
```

Utilisation d'une boucle

Fichier programme "boucle_entiers.c"

```c
#include <stdio.h>

int main() {
    int somme=0; // initialisation de la somme
    int borne;
    int i;
    printf("saisissez une valeur :\n");
    scanf("%d", &borne);

    for (i =1; i <= borne; i++) {
        somme = somme + i;
        printf("au tour de boucle %d, somme vaut %d\n", i, somme);
    }

    printf("la somme des entiers de 1 à %d est %d\n", borne, somme);

    return 0;
}
```

Question

Ecrivez un programme qui :

- Demande à l'utilisateur de saisir une valeur entière positive
- Affiche la factorielle de la valeur saisie

Itérations non déterministes : boucle `while`

Itération non-déterministe : mot-clé `while` (tant que)

```
while (condition) {
  instruction1;
  instruction2;
  instruction3;
}
```

- `condition` : tant que cette condition est vrai, répétition

Fichier programme "`boucle_while.c`"

```
1  int main() {
2      int i = 0;
3      while (i < 10) {
4          printf("tour de boucle numero : %d \n", i);
5          i++;
6      }
7      return 0;
8  }
```

Itérations non déterministes : boucle `while`

Itération non-déterministe avec au moins un tour de boucle

```
do {
   instruction1;
   instruction2;
   instruction3;
} while (condition);
```

- `condition` : tant que cette condition est vrai, répétition

Fichier programme "`boucle_whilefin.c`"

```
1  int main() {
2      int i = 0;
3      do {
4          printf("tour de boucle numero : %d \n", i);
5          i++;
6      } while (i < 10);
7      return 0;
8  }
```

Utilisation du `while`

- Plus simple dans la syntaxe
- Mais moins aimée des néophytes (qui préfèrent `for`)
- Pourtant plus puissante

Question

Nombre de lignes affichées par ce programme :

Fichier programme "`boucleExo.c`"

```
1  int main() {
2
3      for (j=0; j < 100; j++) {
4
5          k=j;
6          while (k > 0) {
7              printf("une ligne affichee");
8              k--;
9          }
10
11         for (l=0; l <=j; l++) {
12             printf("une ligne affichee");
13         }
14     }
15     return 0;
16 }
```

Table des matières

Faire des types composés

Faire ses propres types de variables en groupant des types existants

- De base, 4 types simples : char, int, float, double
- Possibilité de grouper des types simples entre eux pour former des types composés : des structures
- Possibilité de grouper des types simples et des types composés entre eux pour former des types composés plus complexes

Une fois le nouveau type défini :

On va pouvoir :

- créer des variables de ce nouveau type
- accéder aux différents éléments composant le nouveau type

Intérêt des types composés

Exemple

on veut stocker trois dates (jour, mois, annee) dans un programme :

Fichier programme "`stockeDateMauvais.c`"

```
1  int main() {
2      // Stockage des dates 17/07/1981 , 10/08/1987, et 31/01/1988
3      int jour1 = 17;
4      int jour2 = 10;
5      int jour3 = 31;
6      int mois1 = 07;
7      int mois2 = 08;
8      int mois3 = 01;
9      int annee1 = 1981;
10     int annee2 = 1987;
11     int annee3 = 1988;
12 }
```

Ce programme marche ... mais n'est pas lisible

idée : regrouper un jour, un mois, et une année dans un nouveau type `date`, puis définir des variables de type `date`

Intérêt des types composés

Exemple

on veut stocker trois dates (jour, mois, annee) dans un programme :

Fichier programme "`stockeDateMauvais.c`"

```
1  int main() {
2      // Stockage des dates 17/07/1981 , 10/08/1987, et 31/01/1988
3      int jour1 = 17;
4      int jour2 = 10;
5      int jour3 = 31;
6      int mois1 = 07;
7      int mois2 = 08;
8      int mois3 = 01;
9      int annee1 = 1981;
10     int annee2 = 1987;
11     int annee3 = 1988;
12 }
```

Ce programme marche ... mais n'est pas lisible

idée : regrouper un jour, un mois, et une année dans un nouveau type `date`, puis définir des variables de type `date`

former des types composés : mot clé `struct`

- Syntaxe :

```
struct IdentificateurNouveauType {
        type1 IdentificateurSousType1;
        type2 IdentificateurSousType2;
        type3 IdentificateurSousType3;
}
```

Intérêt des types composés

Résultat :

Fichier programme "`stockeDateBon.c`"

```
1  struct date {
2      int jour;
3      int mois;
4      int annee;
5  };
6
7  int main() {
8      // Stockage des dates 17/07/1981 , 10/08/1987, et 31/01/1988
9      struct date date1, date2, date3;
10
11     date1.jour = 17;
12     date1.mois = 07;
13     date1.annee = 1981;
14
15     date2.jour = 10;
16     date2.mois = 08;
17     date2.annee = 1987;
18
19     date3.jour = 31;
20     date3.mois = 01;
21     date3.annee = 1988;
22 }
```

Table des matières

Découpage de code en fonctions

Principe des fonctions et procédure

- Grouper un ensemble d'instructions entre elles
- Leur donner un identificateur
- Les exécuter quand l'identificateur est mentionné dans le code
- Possibilité de passer des paramètres
- Possibilité de récupérer (ou pas) une valeur là où la fonction était appelée dans le code
- Une procédure est une fonction qui ne renvoie pas de valeur
- `main()` est une fonction : fonction principale du programme

Appel à une fonction simple (procédure)

Fichier programme "fonctionSimple.c"

```c
// un exemple d'appel simple a une fonction (procedure)
#include <stdio.h>

void maFonction() {  // void = type vide. Pas de valeur retour
    printf("je suis dans la fonction maFonction()\n");
}

int main() {
    printf("debut du programme :\n");
    maFonction();
    maFonction();
    printf("un message intermediaire...\n");
    maFonction();
    printf("fin du programme.\n");
    return 0;
}
```

résultat :

```
debut du programme :
je suis dans la fonction maFonction()
je suis dans la fonction maFonction()
un message intermediaire...
je suis dans la fonction maFonction()
fin du programme.
```

Récupération d'un résultat

Une fonction peut retourner un résultat

- Annonce spécifique du type de valeur à retourner
- Utilisation du mot clé `return`, suivi de la valeur
- Récupération du résultat dans la fonction appelante
- Une procédure est une fonction qui ne renvoie pas de valeur

Appel à une fonction et récupération du résultat

Fichier programme "fonctionRetour.c"

```
1  // appel a une fonction et recuperation d'une valeur
2  #include <stdio.h>
3
4  int retourValeur() {
5      printf("fonction appelee. Valeur 10 retournee\n");
6      return 10;
7  }
8
9  int main() {
10     int i, j , k;
11     i = retourValeur();
12     j = 2* retourValeur();
13     k = i + retourValeur() +  j + retourValeur();
14     printf("resultat : i=%d, j = %d, k=%d\n", i , j , k);
15     return 0;
16 }
```

résultat :

```
1  fonction appelee. Valeur 10 retournee
2  fonction appelee. Valeur 10 retournee
3  fonction appelee. Valeur 10 retournee
4  fonction appelee. Valeur 10 retournee
5  resultat : i=10, j = 20, k=50
```

Appel à une fonction avec passage de paramètres

Fichier programme "fonctionParams.c"

```c
#include <stdio.h>
void maFonction(int a, int b, int c, char d) {
    printf("maFonction() : a= %d, b=%d, c=%d, d='%c' \n", a , b, c ,d);
}

void maFonction2(int a) {
    printf("maFonction2() : a = %d\n", a);
    a = a + 10;
    printf("maFonction2() : a = %d\n", a);
}

int main() {
    int x=30;
    maFonction(x, 20, 2*x, 'T');
    printf("avant maFonction2() : x = %d\n",x);
    maFonction2(x);
    printf("apres maFonction2() : x = %d\n",x);
}
```

résultat :

```
maFonction() : a= 30, b=20, c=60, d='T'
avant maFonction2() : x = 30
maFonction2() : a = 30
apres maFonction2() : x = 30
```

Appel à une fonction avec passage de param. et résultat

Fichier programme "fonctionParamsRetour.c"

```
1  // appel a une fonction avec parametres et recuperation d'une valeur
2  #include <stdio.h>
3
4  int maFonction(int a, int b, char c) {
5      printf("appel : a=%d  b = %d et c=%c\n", a , b, c);
6      return a*(b+1);
7  }
8
9  int main() {
10     int i, j , k;
11     i = maFonction(5, 4, 'R');
12     j = maFonction(2, 1, 'S');
13     k = maFonction(10, 2, 'T');
14     i = maFonction(10, 10, 'U'); // valeur ecrasee
15     printf("i=%d , j=%d  , k=%d\n", i ,j ,k);
16     return 0;
17 }
```

résultat :

```
1  appel : a=5  b = 4 et c=R
2  appel : a=2  b = 1 et c=S
3  appel : a=10  b = 2 et c=T
4  appel : a=10  b = 10 et c=U
5  i=110 , j=4  , k=30
```

Appel entre fonctions avec passage de param. et résultat

Fichier programme "fonctionParamsRetourAppels.c"

```c
#include <stdio.h>

int f() {
    return 10;
}

int f2 (int x, int y) {
    return x*y + f();
}

int f3 (int x, int y, int z) {
    int i = f2(x,y);
    return i+ z;
}
int main() {
    int k = 10;
    k = k + f3(k, 2, 3);
    printf("valeur de k : %d\n", k );
    return 0;
}
```

résultat :

```
valeur de k : 43
```

Question

Quelle est la valeur retournée par ce programme :

Fichier programme "`fonctionsExo.c`"

```c
// appel a une fonction et recuperation d'une valeur
#include <stdio.h>

int maFonction(int i) {
    int retour;
    if (i ==0) {
        retour = 0;
    }
    else {
        retour = i + maFonction(i-1);
    }
    return retour;
}

int main() {
    int t;
    t = maFonction(10);
    printf("valeur de t = %d\n", t);
    return 0;
}
```