Programmation C

Entrées, Sorties

ING1

CY Tech

Aide en ligne

Utilisation du manuel

- Obtenir de l'aide
  - **internet** : peut ne pas être actif (pendant les examens, ...) ; peut contenir des erreurs plus ou moins grave ; peut permettre de résoudre un problème en particulier
  - **voisin** : peut ne pas être disponible (pendant les examens, ...) ; peut ne pas connaître la réponse
  - `man` : toujours disponible (si installation bien faite) ; ne contient pas d'erreur ; contient de nombreuses réponses

Aide sur une commande

Obtenir de l'aide sur une commande

- `man printf` : donne l'aide sur la commande shell `printf`
- Inutile pour la programmation C
  - `man -a printf`
  - `man 3 printf` [1]
- cf exemple

1. Pour plus de renseignement sur `man` : tapez `man man`

Section du manuel

- NAME : toutes les fonctions qui correspondent à la page de manuel
- SYNOPSIS : le prototype des fonctions, les librairies à inclure, voire même les liens à utiliser
- DESCRIPTION : description des fonctions
- ERRORS : quelles erreurs sont traitées
- RETURN VALUE : ce que retourne la fonction ainsi que la signification éventuelle
- SEE ALSO : d'autres commandes similaires
- BUGS : les bugs connus

Sortie formatée

Sortie formatée

- Sortie : affichage écran (sortie standard)
- Formatée : contrôle la forme et le format de l'affichage
- Sortie formatée en C : `printf`
  - Fonction pré-codée : pas besoin de la recoder
  - Partie de la bibliothèque d'entrée-sortie standard : `stdio`
  - Besoin d'inclure `stdio.h`
- Attention sortie bufferisée
- Syntaxe

```
int printf (const char *format, ...);
/* Plus generalement on considere */
printf("format", [argument1,[argument2,[...]]]);
```

Affichage d'entiers : "%d"

```c
int int_exemple; //entier pour l'exemple
int_exemple = 3;
printf("%d",5);
printf("%d\t", int_exemple);
printf("%d + %d = %d", 5, int_exemple,
  int_exemple+5);
printf("\nLa variable d'exemple vaut : %d\n",
  int_exemple);
```

```
53      5+3=8
La variable d'exemple vaut : 3
```

Autres affichages d'entier possible : "%o", "%x", "%X" et "%u"

```c
int int_exemple; //encore l'exemple
int_exemple = 17;
printf("%d en octale vaut %o\n",
  int_exemple, int_exemple);
printf("%d en hexadecimal vaut %x\n",
  int_exemple, int_exemple);
```

```
17 en octale vaut 21
17 en hexadecimal vaut 11
```

Affichage d'un caractère : "%c"

```
printf("Un caractere s'affiche avec : %c\n", 'a');
printf("Cependant un entier < 255 peut etre
  interprete comme un caractere\n");
printf("Exemple : %c\n", 65);
printf("Inversement : %d\n", 'a');
```

```
Un caractere s'affiche avec : a
Cependant un entier < 255 peut etre interprete
comme un entier
Exemple : A
Inversement : 97
```

Autres affichages

- e, E, f, F, g, G : réels
- s : chaîne de caractères
- p : adresse mémoire (pointeur)
- % : pour afficher le signe %

Entrée formatée

Entrée formatée

- Entrée : clavier (entrée standard)
- Formatée : contrôle la forme et le format de la saisie
- Entrée formatée en C : `scanf`
- Même remarques que pour le `printf`
- Syntaxe quasi identique

```
int scanf (const char *format, ...);
```

Utilité et valeur de retour

- Permet de demander une valeur, ou plusieurs, à l'utilisateur
- Stocke l'information dans une variable, ou plusieurs
- Lecture de l'entrée
- Interprétation en fonction de format
- Retourne le nombre de données interprétées correctement

Exemple (incorrect) d'utilisation

```
int saisie ;
int toto ;
scanf("%d %d", &saisie, &toto);
```

Spécificités CY Tech

Utilisation restreinte

- Interdiction de lire plus d'une variable par `scanf`
- Interdiction de mettre du texte ou autre chose dans le `scanf` à part : `%...`
- Obligation de prendre en compte la valeur de retour de `scanf`
- La lecture d'un double se fait par le format : `%lf`

Utilisation correcte

Exemple correct

```
int int_valeur ; // valeur utilisateur
int int_retour ; //valeur de retour

printf("Entrez une valeur : ");
int_retour = scanf("%d", &int_valeur);
printf("Le nombre de valeur correctement lue : %d",
  int_retour);
printf("\nValeur lue : %d\n", int_valeur);
```

Chaîne de caractères

- Déclaration d'une chaîne : `char str_nom[30];`
- Saisie : `... = scanf("%s", str_nom);`
- Attention pas de &

Écriture des expressions algébriques et logiques

Opérateurs algébriques

- Addition : +
  - `int_x = 2 + 3;`
- Soustraction : -
  - `int_x = 2 - 3;`
- Multiplication : *
  - `int_x = 2 * int_x;`
  - `int_x = 2int_x; // Erreur !`

Opérateurs algébriques

- Division : /
  - `int_x = int_x / 2; // Attention : division entière`
  - `flt_x = flt_x / 2.0;`
  - `int_x = int_x / 2.0;`
  - `flt_x = int_x / 2.0;`
- Modulo : %
  - `int_x = 10 % 2;`
  - `int_x = 10.32 % 4;`
  - `int_x = 10.32 % 4.2;`

Notion de booléen

- Booléen : données qui ne peut prendre que deux états (*Vrai* ou *Faux*)
- N'existe pas en C
- Utilisation du type `int`
  - 0 : *Faux*
  - $\neq 0$: *Vrai*

Opérateurs de comparaison

- Égalité : ==
  - `int_x == 5`
  - `int_x == 2.0`
  - `'a' == 'b'`
  - `'a' == "oui"`
- Inférieur : <
  - `int_x < 5`
  - `int_x < int_y`
  - `'a' < 'b'`
  - `"non" < "oui"`
- Supérieur : >
- Inférieur ou égal : <=
- Supérieur ou égal : >=
- Différent : !=

Opérateurs logiques

- NON : !
  - `!(int_x < int_y)` $\Leftrightarrow$ `(int_x >= int_y)`
- ET : &&
  - `cond1 && cond2` : si cond1 est *Faux* alors cond2 non évaluée
- OU : ||
  - `cond1 || cond2` : si cond1 est *Vrai* alors cond2 non évaluée

Conseils

- "Parenthésez" vos expressions le plus possible
- Normalement évaluation :
  - Opérations numériques
  - Opérations de comparaisons
  - Opérations logiques
- Faites attention à ne pas confondre == avec =
  - *suggest parentheses around assignment used as truth value*
  - Affectation : vaut la valeur de celle-ci
  - $3 == (\texttt{int\_x} = 3) \Leftrightarrow$ *Vrai* et `int_x` vaut 3

Traduction des structures de contrôles standards

Instructions

- Affectation : `int_valeur = 5;`
- Appel de fonctions : `printf("Hello");`
- Affectation et appel de fonctions : `int_retour = scanf(...);`
- Se termine toujours par `;`

Séquences

- Plusieurs instructions à la suite
- Nécessite la création d'un bloc
  - Commence par un {
  - Termine par un }
  - Peut ne contenir qu'une seule instruction

Conditionnelle simple

si condition **alors**
  traitements
fin si

```
if (condition) {
  traitements ;
}
```

Alternative

si condition **alors**
  traitement1
sinon
  traitement2
fin si

```
if (condition) {
   traitement1 ;
} else {
   traitement2 ;
}
```

Aiguillage multiple

selon expr **de**
  const1:
    traitement1
  const2:
    traitement2
  défaut :
    traitementn
fin selon

```
switch (expr) {
  case const1 :
    traitement1 ;
    break;
  case const2 :
    traitement 2 ;
    break;
  .../...
  default :
    traitementn;
    break;
}
```

Aiguillage multiple

- Permet de remplacer dans certains cas une imbrication de `if else`
- `expr` : doit être une variable de type entier, ou une expression entière

Test en tête

tant que condition **faire**
  traitements
fin tant que

```
while (condition) {
  traitements ;
}
```

Test en queue

faire
traitements
tant que condition

```
do {
   traitements ;
} while (condition);
```

- Attention à la `condition`
- Termine par un ;

Répéter

pour i $\leftarrow$ init **à** final **faire**
  traitements
fin pour

```
for(init;cond;iter) {
   traitements ;
}
```

- Alternative syntaxique au *while*
- Déroulement :
  1. Exécution de `init`
  2. Si `cond` vaut *Faux* alors on quitte la boucle. Sinon
  3. Exécution de `traitements`
  4. Exécution de `iter`
  5. Retour à l'étape 2

Spécificités CY Tech

Obligation

- Tester le retour des fonctions (`scanf, ...`)
- Tester les paramètres d'entrée des fonctions (utilisation d'`assert`[2])
- Parenthéser les expressions
  - `int_x * int_x + 5` $\Leftrightarrow$ `((int_x * int_x) + 5)`
- Pas de test implicite
  - `if!int_x` $\Leftrightarrow$ `if (int_x == 0)`
- Utilisation d'une boucle `for` uniquement lorsque la valeur de début et de fin de l'itérateur sont connues à l'avance
- Obligation de mettre un `break` dans chaque `case` d'un `switch`

2. Vu plus tard

Exemple

Cf exemple.c