Faculté des Sciences de Gabès

Cours Atelier de Programmation

Chapitre 4 : Initiation Programmation C

Faîçal Felhi

felhi_fayssal@yahoo.fr

Algorithmes et programmes

- Programme :
  - codage d'un algorithme afin que l'ordinateur puisse exécuter les actions décrites
  - doit être écrit dans un langage **compréhensible** par l'ordinateur
    - → langage de programmation (Assembleur (micropro), Basic, C, Fortran, Pascal, Cobol …)
- Un programme est donc une suite ordonnée d'instructions élémentaires codifiées dans un **langage** de programmation

Faîçal Felhi - felhi_fayssal@yahoo.fr

Langages de programmation

- L'ordinateur
  - construit autour d'un ensemble de circuits électroniques (le courant passe, le courant ne passe pas)
  - traite donc que des signaux assimilables à 0 ou 1
  - une opération élémentaire → suite de 0 et de 1 = suite de bits (BInary digiT) ! Un champ de 8 bits constituant ce qu'on appelle 1 byte ou 1 octet. Importance des unités en science. Rappel: k(2^10) M et G .
- Pour que les programmes et les données soient compréhensibles par l'ordinateur il faut effectuer un codage binaire

Faîçal Felhi - felhi_fayssal@yahoo.fr

Langages de programmation

- Langage machine
  - langage binaire
  - ses opérations sont directement compréhensibles par l'ordinateur
  - **propre à chaque famille d'ordinateur**
  - *Pour pouvoir manipuler du langage machine, on est obligé de passer par de l'Assembleur.*
- Ecriture des premiers programme en langage machine

Faîçal Felhi - felhi_fayssal@yahoo.fr

Langages de programmation

Rédiger un programme consiste à préparer le travail à FAIRE FAIRE à la machine, sous forme d'une liste d'instructions.

Les instructions que peut exécuter l'unité de traitement sont codées en langage binaire spécifique à chaque machine (langage machine).

Langage du programmeur $\rightarrow$ langage machine (code binaire exécutable). La traduction est effectuée par un **compilateur** (qui est lui aussi un programme …).

Faîçal Felhi - felhi_fayssal@yahoo.fr

Langages de programmation

TYPES DE LANGAGES → **STYLE** DE PROGRAMMATION

- Langages impératifs (Fortran, Pascal, C …) : Il s'agit de faire exécuter une suite d'ordres par une machine bête mais disciplinée.
- Langages Déclaratifs: l'activité de programmation consiste essentiellement à décrire le ***rapport*** qui existe entre les données et les résultats que l'on veut obtenir, plutôt que la séquence de traitements qui mène des unes aux autres
  - fonctionnels (Lisp, Scheme …)
  - logiques (Prolog …)
- Langages objets (C++, VisualBasic, Delphi, Java …)

Faîçal Felhi - felhi_fayssal@yahoo.fr

Algorithmique / C

- L'algorithmique sert à réfléchir a l'algorithme
  - Trouver les bonnes variables
  - Imbriquer les boucles dans le bon sens
  - Enchainer les si-alors sinon dans le bon ordre
  - Etc.
- Le langage C sert à laisser un programme formel
  - Pour tester l'utilisation des variables, la syntaxe
  - Tester le programme avec des valeurs concrètes
  - Se convaincre de la justesse de l'algorithme
  - Donner le code source de votre programme pour convaincre vos utilisateurs (open source)

Langage de programmation C

```
#include <stdio.h>                    //bibliothèque

main()  {                             //entête
  const type nom = valeur ;           // bloc déclaration

  type1   nom1, nom2 ;
  type2   nom3 ;
  instruction;
   ...                                //bloc d'instructions
  instruction;
}
```

Syntaxe : symboles, mots, règles

- symboles spéciaux
  - [ ] \{ } . , ; : # =< > - * / ( ) !
- mots réservés
  - **if else int char float double while for switch case**, **const** etc.
- règles syntaxiques
  - point virgule après chaque instruction
  - accolade ouvrante au début { et fermante } à la fin de chaque fonction (y compris « main »), de chaque bloc d'instructions
- La syntaxe d'un programme est définie par une grammaire.

Autres règles

- Contraintes imposées par le langage
  - Toute variable apparaissant dans le bloc d'instructions doit être déclarée.
- Contraintes imposées par l'usage
  - Tout programme doit être commenté !
  - Un commentaire est du texte encadré par des symboles de début /* et de fin */ ou une ligne commençant par //
  - Ignoré lors des traitements du programme

```
/* Tout l'algo repose sur la recherche du plus grand
   nombre premier après le carré parfait */
int i = 0; // sert d'indice dans la boucle
```

Bloc déclaration

Type nom-variable ;

- Syntaxe
  - nom-variable est un identificateur :
    - les caractères sont les lettres (A..Z,a..z) et les chiffres 0..9 et le soulignement (pas de caractères spéciaux, pas de blancs) mais ne commencent pas par un chiffre
    - ne commence pas par un chiffre
    - minuscules et majuscules sont différentes fred≠Fred
    - longueur maximum = 31 (plus de caractères sont tolérés, mais ils sont ignorés)
- Déclarer une variable sert à
  - désigner un récipient par son nom
  - spécifier le domaine des valeurs que peut «contenir » cette variable

Types

- Généralités
  - Un type est un nom pour un ensemble de valeurs. Un type est muni d'opérateurs. Donc :
  - Déclarer une variable sert aussi à connaître les opérateurs applicables à (la valeur de) la variable

Types

- En c, La lettre qui suit les "%" dans le format correspond à un type de variable :

Type	Lettre
int	%d
long	%ld
float/double	%f / %lf
char	%c
string (char*)	%s
short	%hd
entier hexadécimal	%x

Les types entiers

- Pour manipuler les entiers, C propose 6 types (Table ci-dessous). D'autres existent.
- Les opérations sur les entiers sont l'addition +, la soustraction -, la multiplication *, la division / et les comparaisons (==, !=, >; >=, etc.)

TYPE	Intervalle	Codage
unsigned char	[0,255]	1 octet (= 8 bits)
char	[-128,127]	1 octet
unsigned short	[0, 65536]	2 octets
short	[-32768, 32767]	2 octets
unsigned int	[0, 4294967295]	4 octets
int	[-2147483648, 2147483647]	4 octets

Les types réels

- Pour manipuler les réels, C propose 2 types (d'autres existent) présentés dans la table ci-dessous
- Les opérateurs sur les réels sont l'addition +, la soustraction -, la multiplication *, la division /, les comparaisons (==, !=, >, >=, etc.)

TYPE	Intervalle	Codage	Chiffres significatifs
float	$[-2^{-150}, -2^{128}]$, 0, $[+2^{-150}, +2^{128}]$	4 octets	7 chiffres décimaux
double	$[-2^{-1075}, -2^{1024}]$, 0, $[+2^{-1075}, +2^{1024}]$	8 octets	15 chiffres décimaux

Type booléen

- Le type booléen n'existe pas en C.
- Le booléen faux est représenté par l'entier 0, et le booléen vrai par tout entier différent de 0.
- Les opérations de comparaison produisent 0 quand la condition est fausse et 1 quand la condition est vraie.
- Les opérateurs sur les expressions booléennes sont le ET : **&&**, le OU : **||** et le NON : **!**

Type caractère

- Le type caractère n'existe pas de manière indépendante en C
- Les caractères sont représentés par des « char » correspondant au codage ASCII des caractères alphanumériques (lettres et chiffres), typographiques (ponctuation), etc.
- Ce sont en fait des nombres entre 0 et 255 avec une convention (ASCII ou UNICODE)
- Les caractères sont entrés entre quotes 'a'(==97), 'b'(==98), '0'(==48), 'A'(==65), etc.

Déclarations de constantes

- **const type** nom-constante = valeur
  - nom-constante est un identificateur qui sert à nommer une valeur.
  - Une constante sert souvent à simplifier la lisibilité d'un programme.
  - Le nom donné à la valeur correspondant à l'utilisation de cette valeur dans un contexte particulier (ici le programme).
- Exemples
  - **const float** PI = 3.14159; // PI est la valeur 3.14159
  - **const float** euro 6.56; // euro est le réel 6.56
  - **const int** duo = 2; // duo est synonyme de 2
- Avertissement
  - Il est impossible de changer la valeur 2
  - De la même manière il est impossible de toucher à la constante **duo** dans le programme !

Affectation

nom-variable = expression ;

- sémantique
  - seule la notation change par rapport au langage algorithmique
    i←i+1

  le type de l'expression à droite de = doit être identique au type de la variable à gauche

- Exemples

  I = 0 ;

  I = I + 1 ;

  res = (J = =I*I) ; // res = 1 ou res = 0

Attention : if (i=3){…} change la valeur de i !

Affectation

- Attention : le compilateur C fait des conversions pour que la valeur affectée corresponde au type de la variable à gauche.
  - Exemple :
    - int n; float x=15.4;
    - n=x; // Les deux types sont différents
    - printf("n=%d \n", n); // résultat affiché : n=15

Instructions d’entrée-sortie

scanf("FORMAT", &nom-variable);

- Permet de saisir (lire) des données tapées au clavier
- FORMAT permet de spécifier le type de la variable lue. Par exemple, "%d" pour un entier, "%f" pour un réel…
  (d = décimal, f = floating point)

L'exécution de l'instruction ci-dessus

- Attend que l'utilisateur tape une valeur au clavier
- Cette valeur est affectée à la variable (idem au pluriel)
- La variable doit avoir été declarée (avec le bon type) (const)

Exemple

```
int I= 234 ;
scanf ("%d",&I) ;
```

- Si l'utilisateur tape 33, la valeur de la variable I est 33 après exécution des deux instructions.

Instructions d’entrée-sortie

scanf ("FORMATS", liste de variables);

Exemple

scanf ("%d %d %d", &I, &J, &N)

si l'utilisateur tape 33 44 22, la valeur de la variable I est 33, celle de J est 44 et celle de N est 22 après exécution.

Avertissement

Si la valeur saisie n'est pas du type de la variable alors une **erreur d'exécution** se produit.

Si la valeur n'est pas saisie, alors l'exécution du programme attend !

Instructions d’entrée-sortie

printf ("FORMAT", expression)

printf ("FORMATS", liste d'expressions)

- permet d'afficher des valeurs (résultats de calcul)à l'écran.

Exemples

```
    float res = 2.2+1.05;
  int I =1 ;                          //la valeur de I est 1
  printf("%d", I) ;                   //affichage de 1 à l'écran
  printf("%d", 5+7) ;                 //affichage de 12 à l'écran
  printf("valeur de I= %d", I) ; //affichage de  valeur de I= 1
    printf("pour I= %d res=%f", I, F) ;
            //affichage de  pour I=1 res=3.25
```

printf ("FORMAT \n", expression) affiche le résultat de l'évaluation de l'expression puis effectue un retour à la ligne

Caractère spéciaux

le langage C utilise ce que l'on appelle une « séquence d'échappement ». Cela veut simplement dire qu'ils sont précédés d'une contre barre \.

- \n : nouvelle ligne (NL=New Line)
- \r : retour chariot (CR= carriage return)
- \t : tabulation horizontale (HT= horizontale tabulation)
- \f : saut de page (FF= form feed)
- \v : tabulation verticale (VT= vertical tabulation)
- \a : signal d'alerte (un bip)
- \b : retour arrière (BS = BackSpace)
- \\ : caractère d'échappement (affiche l'antislash \)
- \YY affiche le caractère dont le code ASCII est représenté par sa valeur hexadécimale. YY étant la valeur comprise entre 00 et FF.
- \YYY affiche le caractère de code octal YYY

Programmation en C du Test-Carré-Parfait

```
main() {
  int I, N;
  ...
  printf ("Donnez l'entier a tester : \n");
  scanf("%d", &N);          // saisie de la valeur de N au clavier
  I = 0;                    // initialisation de I
  ...
  printf( "Oui la valeur que vous avez entré…");
}
```

- Avertissement
  - Toute instruction est suivie d'un point virgule ;

Conditionnelle

if (condition)
    instruction1 ;

Erreur à éviter (le point virgule après la condition) :

if (condition)**;**
    instruction1 ;

condition est une expression booléenne

L'exécution de l'instruction globale

- évalue la condition
- si la condition est vraie, exécute l'instruction 1.

Attention : si la condition est fausse, il ne se passe rien dans ce cas.

- Exemple

```
N = 4 ; I=2;
if (N==I*I)
  printf ("L'entier %d est un carré parfait", N);
```

Affichage à l'écran de : L'entier 4 est un carré parfait

Conditionnelle

if (condition) instruction1;
if (condition) instruction1; **else** instruction2;
if (…)**;** instruction1; EST TOUJOURS EXECUTE A CAUSE DU ;

- Permet d'introduire des branchements d'instructions.
- L'exécution de l'instruction globale
  - évalue la condition
  - si la condition est vraie, exécute l'instruction 1
  - sinon exécute l'instruction 2.

- Exemple

  N = 5; I=2;
  if (N= =I*I) **printf** ("L'entier %d est un carre parfait", N);
  else **printf**("L'entier %d n'est pas un carre parfait", N);

  Affichage à l'écran de L'entier 5 n'est pas un carre parfait

Conditionnelle

if (condition) bloc-instruction1 **else** bloc-instruction2

Un bloc d'instructions est une liste d'instructions encadrée par les mots clé **{** et **}**

- Exemple

```
N = 5 ;
if (N % 2 = =0)    printf("%d est pair", N);
else {
        N = N-1 ;
        printf ("%d est pair", N);
}
```

Affichage à l'écran : **4 est pair**

% est le reste de la division entière

Reste de la division entière

- 423%100 = 23 (ou 118%8 pages ;-)
  - Car lorsqu'on cherche à diviser 423 par 100
    - Il y a 4 blocs de 100 qui se divisent bien par 100
    - Il reste 23 qui vont faire des chiffres après la virgule
  - 4 est le résultat de la division entière
    - En C elle s'ecrit / (ex int a = 423/100;)
  - 23 est le reste de la division entière
    - En C elle s'ecrit % (ex int b = 423%100;)
  - La division réelle s'écrit float c = 423.0f / 100;

Conditionnelle

```
switch  (expression) {
   case  expression-constante : bloc-instruction 1;
   break;
   case  expression-constante : bloc-instruction 2;
   break;
    ...
   case expression-constante : bloc-instruction n;
   break;
   default :  bloc-instruction;  break;
}
```

- le cas **default** est facultatif.
- Pas de break signifie que les 2 cas sont traités ensemble
- L'instruction **break** provoque une sortie immédiate du **switch**

Conditionnelle

- Exemple

```
N=5
switch (N%2) {
   case 1 : printf ("%d est impair", N) ; break;
   case 0 : printf ("%d est pair", N) ; break;
   }
```

- Exemple

```
switch (C) {
   case  '0' : case  '2' : case  '4' : case  '6' : case  '8' :
           printf("%d est le code d'un chiffre pair"  , C);  break;
   case  '1' : case  '3' : case  '5' : case  '7' : case  '9' :
           printf("%d est le code d'un chiffre impair", C);  break;
     default :
           printf ("%d n'est pas le code d'un chiffre, c'est %c", C, C);
}
```