Université Mohammed V-Agdal
Faculté des sciences de Rabat
Département d'Informatique

Langage C

Préparé et présenté par
Pr S.ZITI
Pr. M. Benchrifa

Année Universitaire 2009/2010

Plan

- Introduction
- Types de base, Opérateurs et Expressions
- Lecture & écriture des données
- Structures de contrôle
- Tableaux
- Pointeurs en langage C
- Fonctions
- Chaînes de caractère
- Types structures, unions et synonymes
- Fichiers

Introduction

Préliminaire

- Un algorithme est une séquence d'opérations visant à la résolution d'un problème en un temps fini. La conception d'un algorithme se fait par étapes de plus en plus détaillées.

- La traduction de l'algorithme en un langage informatique et on obtient un programme qui est défini comme une suite d'instructions permettant de réaliser une ou plusieurs taches, de résoudre un problème, de manipuler des données.

- Le langage de base compréhensible par un ordinateur, appelé langage machine est constitué d'une suite de 0 et de 1.

- Plusieurs langages de programmation : structurelle (C, Pascal, …), fonctionnelle (Lisp,…), logique (Prolog, …), scientifique (Maple, Matlab,…), objet (C++, Java, …)

Introduction

- **Historique**
  - **En 1972, Ritchie a conçu le langage C pour développer une version portable du système d'exploitation UNIX.**
  - **En 1978, le duo *Ritchie* / *Kernighan* a publié la définition classique du langage C dans le livre « The C programming language »,**
  - **Dans les années 80, le langage C est devenu de plus en plus populaire que ce soit dans le monde académique que celui des professionnels (C avec des extensions particuliers)**
  - **En 1983, l'organisme ANSI chargeait une commission de mettre au point une définition explicite et indépendante de la machine pour le langage C➔ définition de la norme ANSI-C en 1989.**

Introduction

- ## Caractéristiques du langage C

  - **C est universel : permet aussi bien la programmation système que la programmation de divers applications (scientifique, …)**

  - **C est prés de la machine : offre des opérateurs qui sont très proches de ceux du langage machine…**

  - **C est de haut niveau : C est un langage structuré, typé, modulaire et compilé**

  - **C est portable : en respectant le standard ANSI-C, il est possible d'utiliser le même programme source sur d'autres compilateurs.**

Introduction

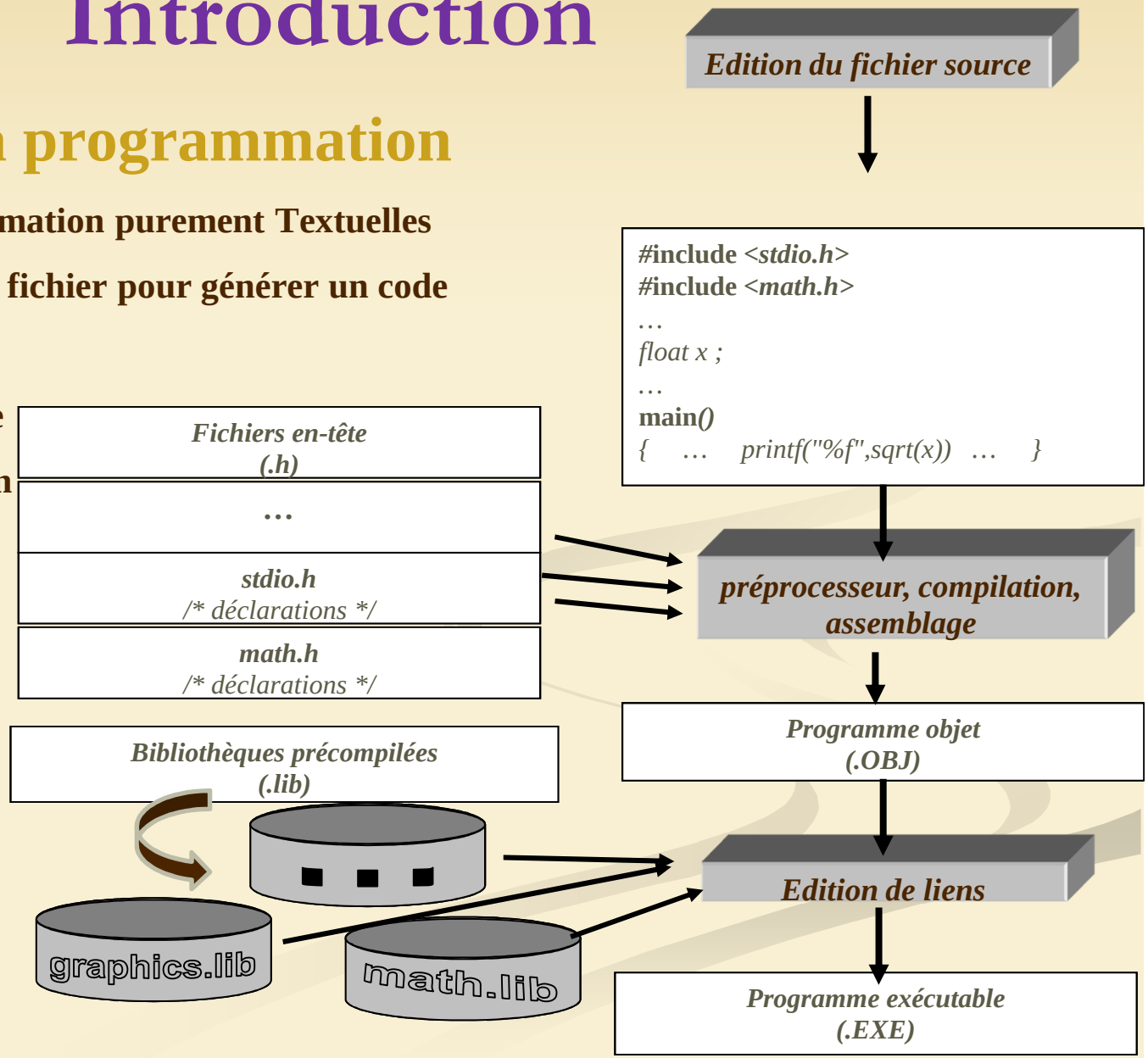
Phases de la programmation

Préprocesseur : transformation purement Textuelles

Compilation : Traduit le fichier pour générer un code en assembleur

Assemblage : transforme le code assembleur en un fichier binaire (*.obj)

Edition de liens : liaison des fichiers objets et production de l'exécutable



Introduction

- ## Composantes du langage C

  - **Fonctions : composée d'une ligne déclarative et un bloc d'instructions**
  - **Fonction main() : fonction principale et obligatoire des programmes en C ;**
  - **Variables : spécifiés par des identificateurs et contenant les valeurs nécessaires pour l'exécution. Ils doivent être déclarés avant leurs appels**
  - **Identificateurs : Les noms des fonctions et des variables en C sont composés d'une suite de lettres et de chiffres, plus le caractère souligné (_). Commençant par une lettre**
  - **Commentaires: programme plus compréhensible, \\ ou * …***

Introduction

- **Exemple de programme en C**

```
/**************************************************************************
 Ce programme affiche le message  Bonjour  tout le monde  et la racine carrée de 20
 **************************************************************************/

#include <stdio.h>  //fichier d'entête contenant la déclaration de la fonction printf
#include <math.h>  //fichier d'entête contenant la déclaration de la fonction sqrt

 int main()
 {
    printf("Bonjour  tout le monde\n"); /

    printf("Voici la racine carrée de 20 : %f\n",sqrt(20)); //  la racine carrée de 20

    return 0;
 }
```

Type de base, opérateurs et expressions

- **Types simples**
- **Types entiers :**
  - 4 variantes d'entiers : *caractères* (char), *entiers courts* (short int), *entiers longs* (long int) et *entiers standards* (int).
  - Caractéristiques *:*

Définition	*Valeur minimale*	*Valeur maximale*	*Nombre d'octets*
Char	-128	127	1
short int	-32768	32767	2
Int	-32768	32767	2
long int	-2147483648	2147483647	4

  - Remarques :
    - Un caractère est un nombre entier (il s'identifie à son code *ASCII*). un char peut contenir une valeur entre -128 et 127 et elle peut subir les mêmes opérations que les variables du type short, int ou long.
    - Un nombre entier de type int est souvent représenté sur *1 mot machine* (16 bits ou 32 bits).
    - Si l'on ajoute le préfixe unsigned à l'une de ces variantes, alors on manipule des entiers non signés :
      - unsigned char : indique des valeurs entières entre 0 et 255.
      - unsigned int (resp. short) : entre 0 et 65535.
      - unsigned long : entre 0 et 4294967295.

Type de base, opérateurs et expressions

- **Types simples**
- **Types réels:**
  - **3 types de réels :**
    - **réels simple précision (float),**
    - **réels double précision (double) et**
    - **réels très grande précision (long double).**
  - **Caractéristiques :**

Définition	*Précision*	**Répartition des bits S , M , E**	*Domaine (approximatif)*	*Nombre d'octets*
float	Simple	**1 , 23 , 8**	$\pm 1{,}1\ 10^{-38}$ à $\pm 3{,}4\ 10^{38}$	4
double	Double	**1 , 52 , 11**	$\pm 2{,}2\ 10^{-308}$ à $\pm 1{,}7\ 10^{308}$	8
long double	Très grande	**1 , 64 , 15**	$\pm 3{,}4\ 10^{-4932}$ à $\pm 1{,}1\ 10^{4932}$	10

Type de base, opérateurs et expressions

Déclaration des types simples

Les ***variables*** et les *constantes* sont les données principales manipulées par un programme.

- En C toute variable utilisée dans un programme doit auparavant avoir été définie. Cette définition consiste à la nommer, à lui donner un type et, éventuellement lui donner une valeur de départ (initialisation)
- Syntaxe de déclaration :

  <type> <NomVar1>, <NomVar2>, …, NomVarN> ;

- Exemple en *C* :
  - **long** x, y ;
  - **short** compteur ;
  - **float** hauteur, largeur ;
  - **double** r ;
  - **char** touche ;

Type de base, opérateurs et expressions

Déclaration des types simples

Constantes

Entière

* Sous forme **décimale** : **100**, **255**.
* Sous forme **octale**: **0144**, **0377**.
* Sous forme **hexadécimale**: **0x64**, **0Xff**

Utilisation des suffixe U, L ou UL
Pour forcer le type

Réelle

* Sous forme **décimale** : **123.4**.
* Sous forme **exponentielle**: **1234e-1**.

Type double; utilisation des suffixes
F ou L pour forcer le type

Caractère

* Sont toujours indiqués entre apostrophes **' ': 'A'**.
* Pour les caractères spéciaux, utiliser ****: **\t**.

Chaîne de caractères

* Une suite de caractères représentées entre guillemets **" "** : **" a"**
* Le caractère nul **'\0'** est rajoutée à toute chaîne pour indiquer sa fin.

Opérateurs, Expressions & Instructions

- Les opérateurs sont des symboles qui permettent de manipuler des variables, c'est-à-dire effectuer des opérations.
- Le Langage C fournis plusieurs opérateurs. Des opérateurs classiques (arithmétique, relationnels, logiques), d'autres moins classiques (manipulation de bits) ou d'opérateurs originaux d'affectation ou d'incrémentation.
- Une expression est un calcul qui donne une valeur comme résultat.
- En C, les constantes et les variables sont des expressions..
- Une expression peut comporter des variables et des constantes combinés entre eux par des opérateurs et former ainsi une expression complexe
- Toute expression suivie d'un point virgule devient une instruction.
- Une instruction en C est un ordre qui sera traduit (par la compilateur) en un ou plusieurs instructions machine.

Opérateurs, Expressions & Instructions

- **Opérateurs classiques**
  - **Opérateur d'affectation simple =**

    <variable> = <expression> ;

    - **L'expression est évaluée puis le résultat est affecté à la variable.**
    - **En C, le terme à gauche de l'opérateur d'affectation est appelé « lvalue » et doit être une référence à un emplacement mémoire dont on pourra effectivement modifier la valeur.**
    - **Les affectations sont interprétées comme des expressions qui retourne la valeur affectée**
    - **Les affectations peuvent être enchaînées, l'évaluation commence de droite vers la gauche**
    - **Exemples :**
      - **const int LONG = 141 ;short val, résultat ; val = LONG ;**
      - **resultat = 45 + 5 * val;**
      - **a=b=c=d ⟷ a=(b=(c=d))**

Opérateurs, Expressions & Instructions

Opérateurs classiques

Opérateur Arithmétique

- + - * /
- L'opérateur % permet d'obtenir le reste de la division entière.
- L'opérateur / retourne un quotient entier si les deux opérandes sont entiers. Il retourne un quotient réel si l'un au moins des opérandes est un réel.

Opérateurs logiques

&&	:	ET logique (and)
||	:	OU logique (or)
!	:	négation logique (not)

- S'appliquent à des expressions booléennes (0 si faux et valeur non nulle si vrai)
- ET retourne la valeur 1 si les deux opérandes sont non nuls, et 0 sinon.
- OU retourne la valeur 1 si au moins un des opérandes est non nul, et 0 sinon.
- Exemples
  - L'expression : 32 && 40 vaut 1
  - L'expression : !65.34 vaut 0

Opérateurs, Expressions & Instructions

- ## Opérateurs classiques
  - ### Opérateur de comparaison
  - = = , != ,<, <=, >, >=.
  - **Opérateurs retournent la valeur 0 si la comparaison est fausse et 1 sinon**
  - **Exemple** **0 || !(32 > 12) retourne la valeur 0.**
  - ### Opérateurs de bits
  - **Ils travaillent sur les bits. Les opérandes doivent être de type entier (char, short, int, long, signés ou non).**
  - **Opérateurs de décalage de bits**

    >> : décalage à droite. << : décalage à gauche.
    - **L'opérande gauche constitue l'objet à décaler et l'opérande droit le nombre de bits de décalage.**
    - **Si la quantité à décaler est signée alors le bit signe est préservée lors d'un décalage à droite, c.-à-d. ce bit se propage de façon à garder le signe de la donnée.**
    - **Si la quantité est non signée, les bits laissés libres sont mis à 0.**

Opérateurs, Expressions & Instructions

- **Opérateurs classiques**
  - **Opérateurs de bits**
  - **Opérateurs de décalage de bits**
    - **Exemple**

```
short j,i = -32768 ;                    /* représ. de i : 1000 0000 0000 0000 */
j = i>>2 ;                              /* représ. de j : 1110 0000 0000 0000 = -8192*/
unsigned  short v,u = 32768 ;           /* représ. de u : 1000 0000 0000 0000 */
v = u>>2 ;                              /* représ. de v : 0010 0000 0000 0000 *
```

  - **Opérateur bit à bit**
  - **& : ET logique; | : OU inclusif**
  - **^ : OU exclusif; ~ : complément à 1.**
  - **Ici, les opérateurs portent sur les bits de même rang.**
  - **Rappel :**

1&1 == 1	1|1 == 1	1^1 == 0
1&0 == 0	1|0 == 1	1^0 == 1
0&1 == 0	0|1 == 1	0^1 == 1
0&0 == 0	0|0 == 0	0^0 == 0

Opérateurs, Expressions & Instructions

- **Opérateurs particuliers en C**
  - **Opérateurs d'affectation étendu**
    - **Pour la plupart des expressions de la forme :**

      lvalue = lvalue OPérateur (expr2)
    - **Il existe une formulation équivalente utilisant un opérateur d'affectation étendu:**

      lvalue OP= expr2
    - **Opérateurs d'affectation utilisables :**

      += -= *= /= %=

      <<= >>= &= ^= |=
    - **Exemples**

      a = a + b s'écrit a += b

      n = n << 2 s'écrit n <<= 2
  - **Opérateurs d'incrémentation et de décrémentation**
  - **<var>++; <var>--;**
  - **++<var>; --<var>**

Opérateurs, Expressions & Instructions

- ## Opérateurs particuliers en C
  - ### Opérateur séquentiel

<expr1> , <expr2>,…, <exprN>

- **Exprime des calculs successifs dans une même expression**
- **Le type et la valeur de l'expression sont ceux du dernier opérande.**
- **Exemple : L'expression : x = 5 , x + 6 a pour valeur 11**

  - ### Opérateur conditionnel

<expression> ? <expr1> : <expr2>

- **<expression> est évaluée. Si sa valeur est non nulle, alors la valeur de <expr1> est retournée. Sinon, c'est la valeur de <expr2> qui est renvoyée.**
- **Exemple max = a > b ? a : b**

  - ### Opérateurs sizeof

sizeof(<type>) ou sizeof(<variable>)

- **Retourne le nombre d'octets occupés en mémoire par le type de données ou la variable spécifiés.**
- **Exemple int x ; sizeof(x) /* retourne la valeur 2 ou 4*/**

Opérateurs, Expressions & Instructions

Priorité et associativité des opérateurs

- **Lors de l'évaluation des différentes parties d'une expression, les opérateurs respectent certaines lois de priorité et d'associativité.**
- **Exemples**
  - **La multiplication a la priorité sur l'addition**
  - **La multiplication et l'addition ont la priorité sur l'affectation.**

- **Tableau des opérateurs et priorité**
  - **La priorité est décroissante de haut en bat dans le tableau.**
  - **La règle d'associativité s'applique pour tous les opérateurs d'un même niveau de priorité. (→ pour une associativité de gauche à droit et ← pour une associativité de droite à gauche).**
  - **Les parenthèses forcent la priorité.**

Priorité 1 (la plus forte):	()	→
Priorité 2:	! ++ --	←
Priorité 3:	* / %	→
Priorité 4:	+ -	→
Priorité 5:	< <= > >=	→
Priorité 6:	== !=	→
Priorité 7:	&&	→
Priorité 9 (la plus faible):	= += -= *= /= %=	←

Opérateurs, Expressions & Instructions

- ## Conversion de type (cast)
  - ### Conversion automatique
  - **Si un opérateur a des opérandes de différents types, les valeurs des opérandes sont converties automatiquement dans un type commun.**
  - **Règle de conversion automatique**

    Lors d'une opération avec :
    - **deux entiers : les types char et short sont convertis en int. Ensuite, il est choisit le plus large des deux types dans l'échelle : int, unsigned int, long, unsigned long.**
    - **un entier et un réel : le type entier est converti dans le type du réel.**
    - **deux réels : il est choisit le plus large des deux types selon l'échelle : float, double, long double.**
    - **Dans une affectation : le résultat est toujours converti dans le type de la destination. Si ce type est plus faible, il peut y avoir une perte de précision ou un résultat erroné.**

Opérateurs, Expressions & Instructions

- **Conversion de type (cast)**
  - **Conversion automatique**
  - **Conversion forcée**
    - **Le type d'une expression peut être forcé, en utilisant l'opérateur cast :**

(<type>) <expression>

  - **Exemple**
    - **Char c; c= c+1;**
    - **char** a = 49; // **a = '1'**

Lecture et écriture de données

Écriture formatée de données : printf()

- La fonction printf permet d'afficher du texte, des valeurs de variables ou des résultats d'expressions sur écran (sortie standard).
- **Forme générale** : **printf("<format>", <expr1>, …, <exprN>) ;**
- La partie **"<format>"** est une chaîne de caractères qui peut contenir du **texte,** des **caractères de contrôle** ('\n' , '\t' , …) et **spécificateurs de format**, **un** pour **chaque expression** <expr1>, … et <exprN>.

Spécificateurs de format pour printf :

Spécificateur	*Rôle (afficher :)*	*Type*
%c	un seul caractère	**char**
%d ou %i	un entier relatif	**int**
%u	un entier naturel (non signé)	**unsigned int**
%o	un entier sous forme octale	**int**
%x	un entier sous forme hexadécimale (a-f)	**int**
%X	un entier sous forme hexadécimale (A-F)	**int**
%f	un réel	**float** ou **double**
%e	un réel en notation exponentielle (e)	**float** ou **double**
%E	un réel en notation exponentielle (E)	**float** ou **double**
%s	une chaîne de caractères	**char** *

Lecture et écriture de données

Écriture formatée de données : printf()

- **Exemples :**
  - La suite d'instructions :
    - int a = 1234 ;
    - int b = 566 ;

      printf("%i plus %i est %i\n", a, b, a + b) ;
    - va afficher sur l'écran :

      1234 plus 566 est 1800

  - La suite d'instructions :
    - char b = 'A' ; /* le code ASCII de A est 65 */

      printf("Le caractère %c a le code %i\n", b, b) ;
    - va afficher sur l'écran :

      Le caractère A a le code 65

Lecture et écriture de données

Lecture formatée de données : scanf()

- **scanf lit depuis le clavier (entrée standard). Elle fait correspondre les caractères lus au format indiqué dans la chaîne de format.**
- **La spécification de formats pour scanf est identique à celle de printf, sauf qu'au lieu de fournir comme arguments des variables à scanf, ce sont les adresses de ces variables que l'on transmet.**
- **L'adresse d'une variable est indiquée par le nom de la variable précédé du signe &.**
- **Forme générale :**

scanf("<format>", <AdrVar1>, <AdrVar2>, …, <AdrVarN>)

Lecture et écriture de données

Lecture formatée de données : scanf()

- **Exemple :**
- **int jour, mois, annee ;**

  scanf("%i %i %i", &jour, &mois, &annee) ;
- **Cette instruction lit 3 entiers séparés par les espaces, tabulations ou interlignes. Les valeurs sont attribuées respectivement aux 3 variables : jour, mois et annee.**

- **int i ;**
- **float x ;**

  scanf("%d %f", &i, &x) ;
- **Si lors de l'exécution, on entre 48 et 38.3e-1 alors scanf affecte 48 à i et 38.3e-1 à x.**

Lecture et écriture de données

- ## Ecriture d'un caractère: putchar()
  - **putchar permet d'afficher un caractère sur l'écran.**
  - **putchar(c) ; est équivalente à printf("%c", c) ;**
  - **Forme générale :**

    putchar(<caractere>) ;
  - **Elle reçoit comme argument la valeur d'un caractère convertie en entier.**
  - **Exemples**
    - **char a = 63 ;**
    - **char b = '\n' ;**

      putchar('x') ; /* affiche la lettre x */

      putchar(b) ; /* retour à la ligne */

      putchar(65) ; /* affiche le caractère de code ASCII = 65: A */
  - **Remarque :**

    putchar retourne la valeur du caractère écrit toujours considéré comme un entier, ou bien la valeur -1 (EOF) en cas d'erreur.

Lecture et écriture de données

- ## Lecture d'un caractère: getchar()
  - **Permet de lire un caractère depuis le clavier.**
  - **c=getchar(c) ; est équivalente à scanf("%c",&c) ;**
  - **Forme générale :**

    <Caractere> = getchar() ;
  - **Remarques :**
    - **getchar retourne le caractère lu (un entier entre 0 et 255), ou bien la valeur -1 (EOF).**
    - **getchar lit les données depuis le clavier et fournit les données après confirmation par la touche "entrée"**
  - **Exemple :**
    - **int c ;**

      c = getchar() ; /* attend la saisie d'un caractère au clavier */

Structures de contrôle

- **Structures de choix**
  - Elle permettent de déterminer quelles instructions seront exécutées et dans quel ordre.
  - **Branchement conditionel (if…else) :**

```
if (expression) {bloc-instruction-1}

else  {bloc-instruction-2}
```

    - **Après évaluation de l'expression, si elle est vraie, alors le 1er bloc d'instructions est exécuté, sinon c'est le 2ème bloc qui est exécuté.**
    - **La partie else est optionnelle, lorsque plusieurs instructions if sont imbriquées, chaque else se rapporte au *dernier* if qui ne possède pas de partie else.**
  - **Exemple :**
    - **If (a>10) { b=11; c=12; }**

Structures de contrôle

- ## Structures de choix
  - ### Branchement multiple (switch)
  - **On l'appelle aussi l'*instruction d'aiguillage*. Elle teste si une expression entière prend une valeur parmi *une suite de constantes entières*, et effectue le branchement correspondant si c'est le cas.**
  - **Format :**

```
switch ( expression )
{
  case  expression_constante1 :  suite d'instructions 1
  case expression_constante2 :  suite d'instructions 2
  ...
  case expression_constanteN : suite d'instructions N
  default :  suite d'instructions
}
```

Structures de contrôle

- ## Structures de choix
  - ### Branchement multiple (switch)
  - **Fonctionnement :**
    - **Après l'évaluation de l'expression, s'il existe un énoncé case avec une constante = expression, le contrôle est transféré à l'instruction qui suit cet énoncé;**
    - **sinon si l'énoncé default existe, alors le contrôle est transféré à l'instruction qui suit l'énoncé default ;**
    - **si la valeur de expression ne correspond à aucun énoncé case et s'il n'y a pas d'énoncé default, alors aucune instruction n'est exécutée.**
    - **Attention. Lorsqu'il y a branchement réussi à un case, toutes les instructions qui le suivent sont exécutées, jusqu'à la fin du bloc ou jusqu'à une instruction de rupture (break).**

Structures de contrôle

Structures de choix

Branchement multiple (switch)

Exemple :

```
#include <stdio.h>
main()
{   short  a,b ,
    char operateur ;
        printf("Entrez un opérateur (+, -, * ou /) : ") ;
        scanf("%c", &operateur) ;
        printf("Entrez deux entiers : ") ;
        scanf("%hd  %hd", &a,&b) ;
        switch (operateur)
        { case '+' : printf("a + b = %d\n",a+b) ; break ;
          case '-'  : printf("a - b = %d\n",a-b) ; break ;
          case '*' : printf(" a * b = %d\n",a*b) ; break ;
          case '/' : printf("a / b = %d\n",a/b) ; break ;
          default  : printf("opérateur inconnu\n") ;
        }
        return 0 ;
}
```

Structures de contrôle

Structures de répétition (boucles)

- Les structures répétitives (ou Boucles) permettent de répéter une série d'instructions tant qu'une certaine condition reste vraie.
- Les instructions **while et do…while**
- Les instructions **while** et **do … while** représentent un moyen d'exécuter plusieurs fois la même série d'instructions.
- La syntaxe :

```
while ( condition )
{       liste d'instructions}

do {    liste d'instructions
} while ( condition );
```

- Dans la structure **while** on vérifie la condition avant d'exécuter la liste d'instructions, tandis que dans la structure **do … while** on exécute la liste d'instructions avant de vérifier la condition

Structures de contrôle

- ## Structures de répétition (boucles)
  - **Les instructions while et do…while**
  - **Exemple:**

1. **Code C pour imprimer les entiers de 1 à 9.**

```
i = 1;

while (i < 10)
{
    printf("\n i = %d",i);
    i++;
}
```

2. Code C pour contrôler la saisie au clavier d'un entier entre 1 et 10 :

```
int a;

do
{
  printf("\n Entrez un entier entre 1 et 10 : ");
  scanf("%d",&a);
}
while ((a <= 0) || (a > 10));
```

Structures de contrôle

- **Structures de répétition (boucles)**
  - **L’ instructions for**
  - **permet d'exécuter plusieurs fois la même série d'instructions.**
  - **La syntaxe de for est :**

```
for ( expression1 ; expression2 ; expression3 )
{
        liste d'instructions      }
```

  - **Dans la construction de for :**
    - **expression1 : effectue les initialisations nécessaires avant l’entrée dans la boucle ;**
    - **expression2 : est le test de continuation de la boucle ; le test est évalué avant l’exécution du corps de la boucle;**
    - **expression3 : est évaluée à la fin du corps de la boucle.**

Structures de contrôle

- **Structures de répétition (boucles)**
  - **L’ instructions for**
  - **Remarque**

```
for ( expr1 ; expr2 ; expr3 )
{          liste d'instruction      }
```

↔

```
expr1;
while (expr2)
{liste instructions;
    expr3;
}
```

  - **Exemple**

1. Programme pour calculer la somme de 1 à 100 :

```
short n, total ;

for ( total  = 0, n = 1 ; n<101 ; n++ )
            total += n ;

printf("La somme des nombres de 1 à 100 est %d\n", total) ;
```

Structures de contrôle

- ## Instructions break et continue
  - ### L’ instructions break
  - **On a vu le rôle de l'instruction break; au sein d'une instruction de branchement multiple switch.**
  - **L'instruction break peut, plus généralement, être employée à l'intérieur de n'importe quelle boucle (for ; while ; do …while). Elle permet l’abandon de la structure et le passage à la première instruction qui suit la structure.**
  - **En cas de boucles imbriquées, break fait sortir de la boucle la plus interne.**
  - **Exemple**

```
for ( ; ; )
{
    printf("donne un nombre (0 pour sortir) : ");
    scanf("%d", &n);
   if (n == 0) break;
   exploitation de la donnée
}
```

Structures de contrôle

Instructions break et continue

L' instructions continue

- L'instruction continue peut être employée à l'intérieur d'une structure de type boucle (for ; while ; do …while).
- Elle produit l'abandon de l'itération courante et fait passer directement à l'itération suivante d'une boucle
- L'instruction continue concerne la boucle la plus proche.
- Exemple

```
int main()
{   int i, j ;
    ... //initialisation de i et j
    for (  ;  i>0 && j>0  ;  i--, j-- )
    {
        if  ( i == 5 )     continue ;
        printf("i : %d et j : %d\n, i, j) ;
        if  ( j == 5 )    break ;
    }
    return 0 ;  }
```

Valeurs introduites	Affichage
i = 2 et j = 3	i : 2 et j : 3 i : 1 et j : 2
i = 6 et j = 3	i : 6 et j : 3 i : 4 et j : 1
i = 3 et j = 5	i : 3 et j : 5

Tableaux

- Introduction et Définition
- Tableaux à une dimension (Vecteurs)
  - Déclaration ; Initialisation ; Accès ; …
- Tableaux à plusieurs dimensions
  - Déclaration
  - Tableaux à deux dimensions (matrices) :
    - *Déclaration ;* Initialisation ; Accès ; …
- *Exemples*
- Représentation en mémoire des tableaux

Tableaux

- **Les variables, telles que nous les avons vues, ne permettent de stocker qu'une seule donnée à la fois.**

- **Pour mémoriser et manipuler de nombreuses données (100, 1000, …), des variables distinctes seraient beaucoup trop lourdes à gérer.**

- **Pour résoudre ce problème, le langage C (ainsi que les autres langages de programmations) propose une structure de données permettant de stocker l'ensemble de ces données dans une "variable commune" appelée :<**

Tableau

Tableaux

Définition

- **On appelle tableau une variable composée de données de même type, stockée de manière contiguë en mémoire (les unes à la suite des autres).**

- **La taille d'un tableau est conditionnée (ou définie) par le type et le nombre de ces éléments :**

Taille tableau (en octet) = taille du type de donnée (en octet) * le nombre des éléments

- **Le type des éléments du tableau peut être :**
  - **simple : char, int, float, double, ...**
    - → tableau à une dimension ou tableau unidimensionnel
  - tableau
    - → tableau à plusieurs dimensions ou tableau multidimensionnel
  - Autres : Pointeurs et Structures

Tableaux

Tableaux à une dimension (vecteur)

Déclaration

La déclaration d'un tableau à une dimension se fait de la façon suivante :

<Type Simple> Nom_du_Tableau [Nombre_Elements];

Type Simple : **définit le type d'élément que contient le tableau (char, int,…)**

Nom_du_Tableau : **est le nom que l'on décide de donner au tableau, le nom du tableau suit les mêmes règles qu'un nom de variable.**

Nombre_Elements : **est une expression constante entière positive.**

Exemples :

```
char      caracteres[12] ;      //Taille en octet : 1 octet  * 12 = 12 octets
float     reels_SP[8] ;         //Taille en octet : 4 octets * 8 = 32 octet
#define   N   10   //define permet d'assigner un nom à une constante
int       entier[N] ;           //Taille en octet : 2 octets * 10 = 20 octets
double    reel_DP[2*N-5] ;      //Taille en octet : 8 octets * 15 = 120 octets
```

Tableaux

Tableaux à une dimension

- Initialisation à la déclaration

Il est possible d'initialiser le tableau à la définition :

<Type> Tableau [Nombre_Elements] = {C1, C2, … , Cn};

Où C1, C2, .., Cn sont des constantes dont le nombre ne doit pas dépasser le Nombre_Elements (n<=Nombre_Elements).

Si la liste de constantes ne contient pas assez de valeurs pour tous les éléments,les éléments restantes sont initialisées à zéro.

Exemples :

```
char   voyelles[6]              = { 'a' , 'e' , 'i' , 'o' , 'u' , 'y' } ;
int    Tableau_entier1[10]  = {10 , 5 , 9 , -2 , 011 , 0xaf , 0XBDE};
float  Tableau_reel[8]         = { 1.5 , 1.5e3 ,  0.7E4 };
short A[3]                        = {12 , 23 , 34 , 45 , 56};          //Erreur !
int Tableau_entier2[] = { 15 , -8 , 027 , 0XABDE }    //Tableau de 4 éléments
```

Tableaux

Tableaux à une dimension (vecteur)

- ### Accès aux composantes d'un tableau

Pour accéder à un élément du tableau, il suffit de donner le nom du tableau, suivi de l'indice de l'élément entre crochets :

Nom_du_Tableau [indice]

Où indice est une expression entière positive ou nulle.

Un indice est toujours positif ou nul ;
L'indice du premier élément du tableau est 0 ;
L'indice du dernier élément du tableau est égal au nombre d'éléments – 1.

Exemple :

```
short A[5]  = {12 , 23 , 34 , 45 , 56};
        A[0]          donne accès   au 1er élément du tableau A
int i = 4;  A[i]      donne accès au dernier élément du Tableau A
int j = 2;  A[2*j-1]   donne accès au 4ème élément de A
```

En revanche la plus part des compilateurs C ne font aucun contrôle sur les indices! Ils laissent passer par exemple : A[20] = 6 ; : Accès en dehors du tableau

Tableaux

Tableaux à une dimension (vecteur)

Remarques

- **Chaque élément (TAB[i]) d'un tableau (int TAB[20]) est manipulé comme une simple variable (lvalue), on peut :**

scanf("%d", &TAB[i]); TAB[i] sera initialisé par un entier saisi depuis la clavier

printf("TAB[%d] = %d", i , TAB[i]); Le contenu de TAB[i] sera affiché sur écran

Apparaître comme opérande d'un opérateur d'incrémentation : TAB[i]++ ou --TAB[i]

- **Pour initialiser un tableau (TAB1) par les éléments d'un autre tableau (TAB2) :**
  - **évitez d'écrire TAB1 = TAB2 (incorrect)**
  - **On peut par exemple écrire :**

```
for( i = 0 ; i < taille_tableau ; i++ )
    TAB1[i] = TAB2[i];
```

Tableaux

Tableaux à plusieurs dimensions

Déclaration

De manière similaire, on peut déclarer un tableau à plusieurs dimensions :

<Type Simple> Nom_du_Tableau [Nbre_E_1] [Nbre_E_2]…[Nbre_E_N];

- **Chaque élément entre crochets désigne le nombre d'éléments dans chaque dimension ;**
- **Le nombre de dimension n'est pas limité.**

Tableaux à deux dimensions (Matrices)

Déclaration

<Type Simple> Nom_du_Tableau [Nombre_ligne] [Nombre_colonne];

Exemple :

short T[3][4] ; //Taille en octet : 3 * 4 * 2 octets = 24 octets

La variable tableau T est une matrice. Si on considère la représentation matricielle alors la disposition des éléments de T est :

Tableau[0][0]	Tableau[0][1]	Tableau[0][2]	Tableau[0][3]
Tableau[1][0]	Tableau[1][1]	Tableau[1][2]	Tableau[1][3]
Tableau[2][0]	Tableau[2][1]	Tableau[2][2]	Tableau[2][3]

Tableaux

Tableaux à deux dimensions (matrices)

Initialisation à la déclaration et accès aux éléments :

- **Les valeurs sont affectées ligne par ligne lors de l'initialisation à la déclaration**
- **Accès aux composantes se fait par : Nom_tableau[ligne][colonne].**

Exemples :

float A[3][2] = { {-1.05,-1.10} , {86e-5, 87e-5} , {-12.5E4} };

int B[4][4] = { {-1 , 10 , 013 , Oxfe} , {+8 , -077} , {} , {011,-14,0XAD} };

A ⇨

-1.05	-1.10
86e-5	87e-5
-12.5E4	0.0

A[0][1] = ?

A[1][0] = ?

A[2][1] = ?

B ⇨

-1	**10**	**013**	**0xFE**
+8	**-077**	0	0
0	0	0	0
011	**-14**	**0XAD**	0

B[0][2] = ?

B[1][3] = ?

B[3][2] = ?

Tableaux

Exemple

- **Saisie et affichage des données d'un tableaux d'entiers de 20 éléments aux maximum.**

- **Déterminer la plus petite valeur d'un tableau d'entiers A. Ecrire un programme C qui remplit un tableau T de N entiers long (taille maximale : 30) et affiche ensuite la valeur et la position du minimum. Si le tableau contient plusieurs minimum, retenir la position du premier minimum rencontré.**

- **Recherche d'une valeur dans un tableau : Etant donnés un tableau de float (taille maximale 50) et une valeur. On recherche la première occurrence de cette valeur dans le tableau. S'il existe, on affiche sa position sinon on affiche un message d'erreur.**
  - **Tableau est non trié (recherche séquentielle)**
  - **Tableau est trié (recherche dichotomique).**

Tableaux

Exemple

- **Saisie et affichage des données entières d'une matrice de M ligne (20 lignes au maximum) et N colonnes (30 colonnes au maximum). M et N sont entrées au clavier.**

- **Produit de deux matrices des données réelles: En multipliant une matrice A de M lignes (au maximum 10 ligne) et N colonnes (au maximum 15 colonnes) avec une matrice B de N lignes et P colonnes (au maximum 20 colonnes), on obtient une matrice C de M lignes et P colonnes :**

  La composante c_{ij} de la matrice C, placée à la $i^{ème}$ ligne et $j^{ème}$ colonne, se calcule de la façon suivante :

$$c_{ij} = \sum_{k=1}^{M} a_{ik}\, b_{kj} \qquad \text{où } 1 \le i \le N \text{ et } 1 \le j \le P$$

Tableaux

Exercice (Extrait du CC version 2008/2009)

Dans cet exercice, on considérera les entiers de l'ensemble [-200,200].

Ecrire un programme C qui :

- Remplit une matrice A de M lignes et N colonnes (au maximum 30 lignes et 20 colonnes) ligne par ligne par des entiers saisis au clavier.
- Puis construit un tableau T par les éléments strictement positifs de la matrice A parcourue colonne par colonne.

Ex : $A=\begin{pmatrix} -2 & -4 & -26 & 6 & -1 \\ 14 & -12 & -124 & 0 & 4 \\ 5 & 8 & -54 & -4 & 7 \\ -20 & -9 & 4 & -7 & -5 \\ 17 & 11 & 8 & -34 & -1 \end{pmatrix}$ ⇨ T = (14, 5, 17, 8, 11, 4, 8, 6, 4, 7)

- Ensuite trie le tableau T selon le critère suivant : tous les entiers pairs doivent être au début du tableau et les entiers impairs à la fin (sans utiliser un tableau intermédiaire).

  Ex :

  - Pour le tableau T = (14, 5, 17, 8, 11, 4, 8, 6, 4, 7)
  - deviendra après le tri T = (14, 4, 6, 8, 8, 4, 11, 17, 5, 7)
- Enfin affiche le tableau T.

Tableaux

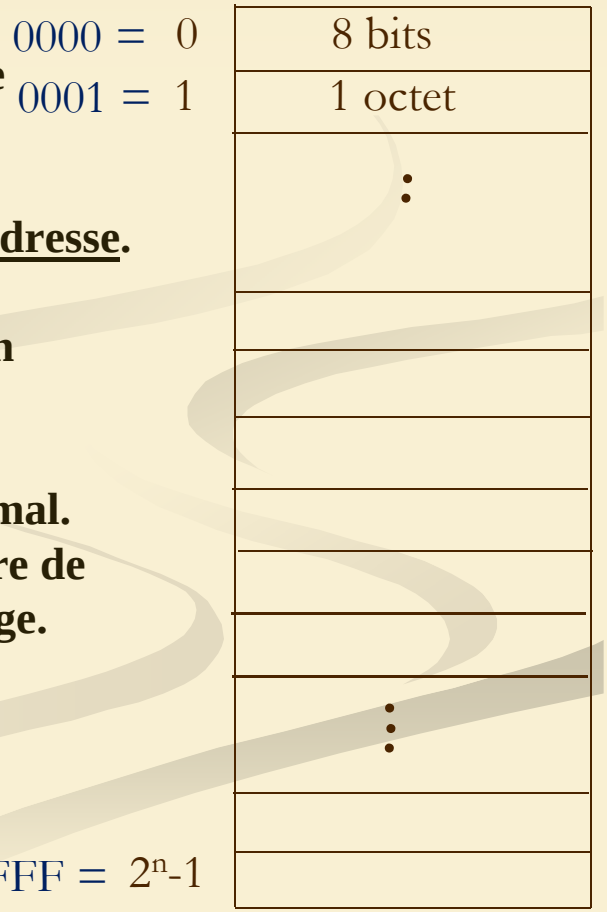
Organisation de la mémoire RAM

- **Rappelons que la mémoire (RAM) est une superposition de <u>cases mémoires</u> ou rangées.**

- **Chaque case mémoire est une suite de 8 bits (1 octets). La lecture ou l'écriture en mémoire se fait par mot machine (2 octets ou 4 octets).**

- **Chaque case mémoire est identifiée par un numéro appelé <u>adresse</u>. Par convention, la première case mémoire est identifiée par l'adresse 0, la seconde par adresse 1, …, jusqu'à 2^n-1 avec n le nombre de bits pour l'adressage.**

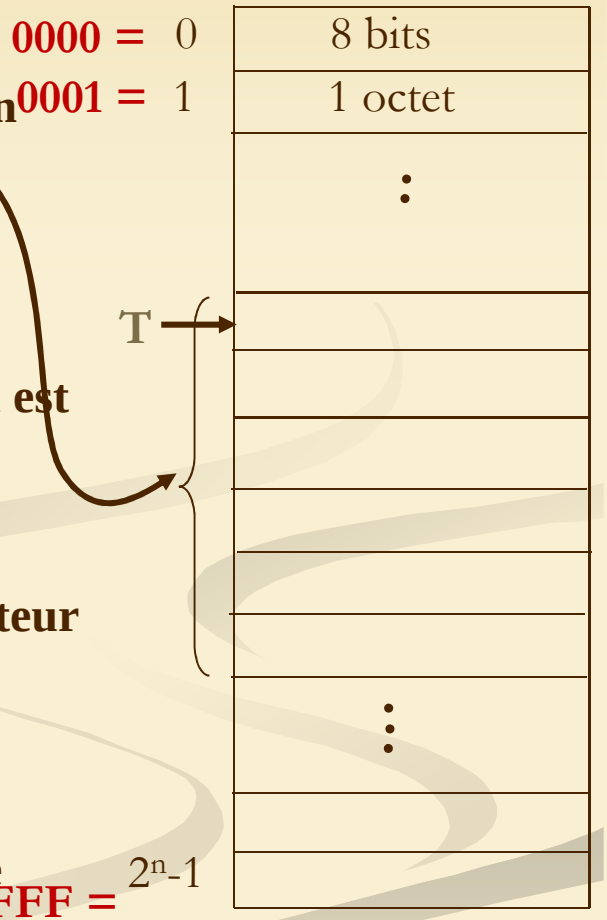
- **Souvent on est ramené à exprimer cette adresse en hexadécimal. Une écriture plus compact proche de la représentation binaire de l'adresse et donne une idée sur le nombre des bits d'adressage. Dans le cas ou n = 16, on dit un adressage sur 16 bits. Dans des machines, on peut avoir un adressage sur n = 32 bits ou n = 64 bits.**



Tableaux

Représentation en mémoire d'un tableau à une dimension

- En C, la déclaration d'un tableau T induit une réservation automatique d'une zone mémoire contiguë (les cases mémoire sont successives).
  Cette zone mémoire sera associée à la variable tableau T le long de l'exécution du programme.
- Dés la déclaration, on connaît la taille d'un tableau T qui est conditionnée par le nombre N des éléments et leur type :
  Taille de T en octet = N * sizeof (type)
- En C, le nom d'un tableau T (l'identificateur) est un pointeur constant qui pointe sur le premier élément du tableau. Il contient l'adresse du premier élément du tableau.
- Etant donnée l'adresse du premier élément du tableau (matérialisée par le nom du tableau T), l'adresse du ième élément de T est donnée par :



&T[i] = &T[0] + sizeof(type) * i ou
&T[i] = T + sizeof(type) * i

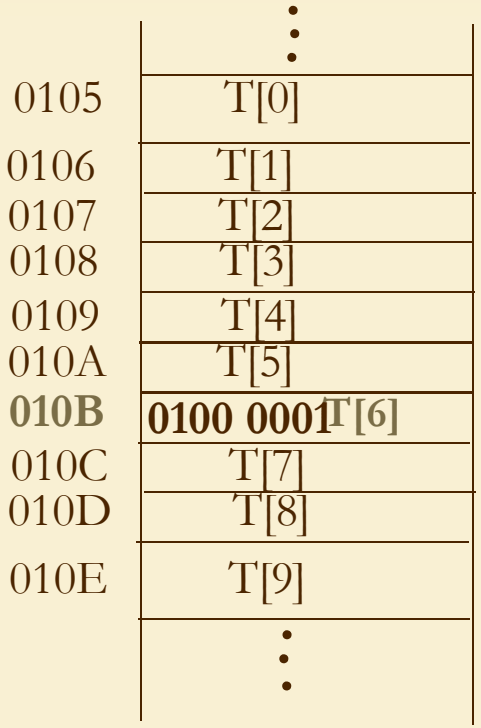
Tableaux

Déclaration et représentation en mémoire d'un tableau à une dimension

Déclaration char T[10]

Taille en mémoire 1octet * 10 = 10 octets

Adresse de début T = 0105 hexa

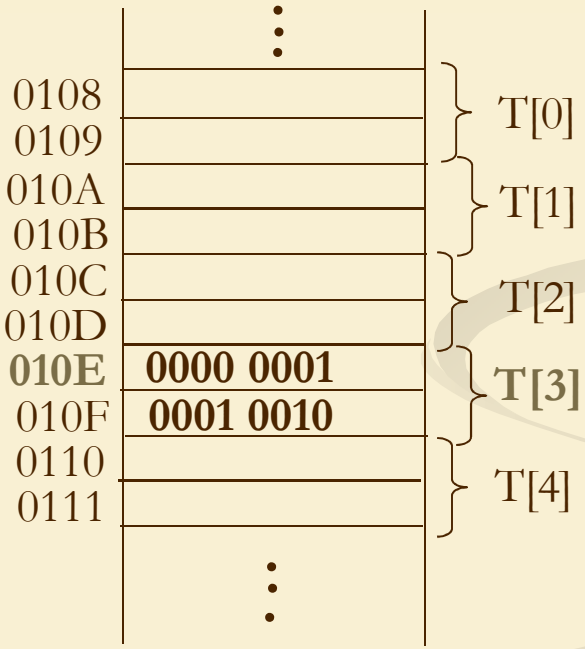


scanf("%c", &T[6]) ← A

short T[5]

2 octets * 5 = 10 octets

T = 0108 hexa



scanf("%hd", &T[3]) ←274

float T[3]

4octets * 3 = 12 octets

T = 0116 hexa

scanf("%f", &T[2]) ← - 2.3 ?

Tableaux

Représentation en mémoire d'un tableau à 2 dimensions

La déclaration d'un tableau T à deux dimensions (matrice) induit la réservation de l'espace mémoire nécessaire pour accueillir tous les éléments du tableau.

Les éléments de la matrice sont répartis en mémoire ligne par ligne.

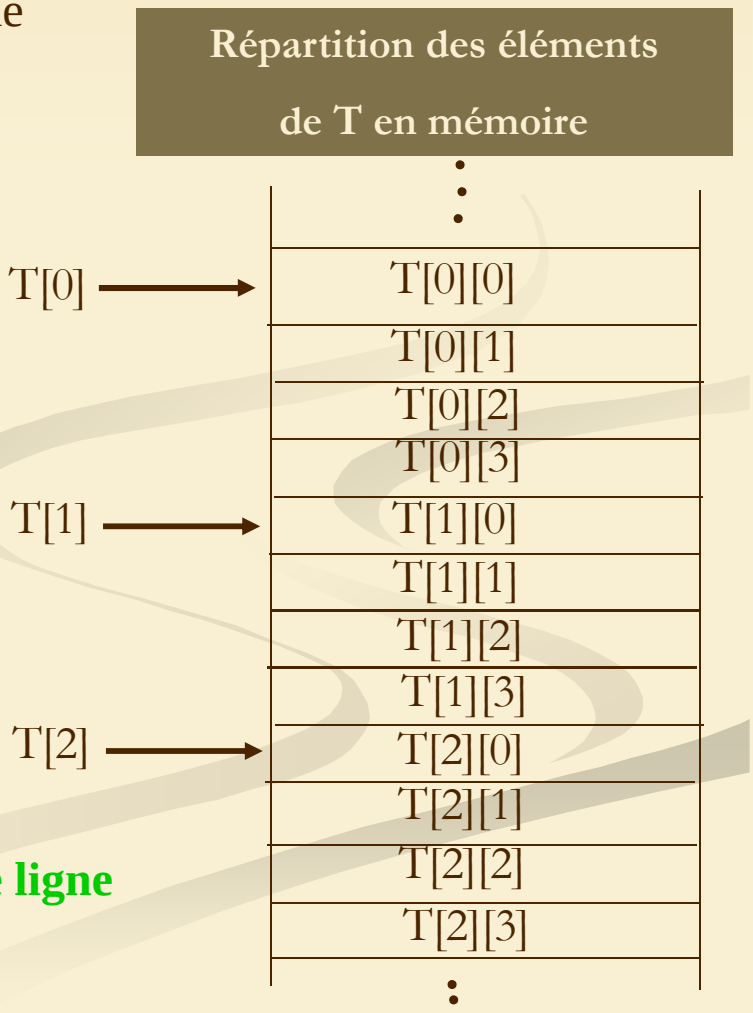
Par exemple, soit T un tableau défini par : char T[3][4]
Alors on a :

T[0] et &T[0][0] : adresse du 1er élément

T[1] et &T[1][0] : adresse du 1er élément de la 2ème ligne

T[2] et &T[2][0] : adresse du 1er élément de la 3ème ligne

T[i] et &T[i][0] : adresse du 1er élément de la ième ligne



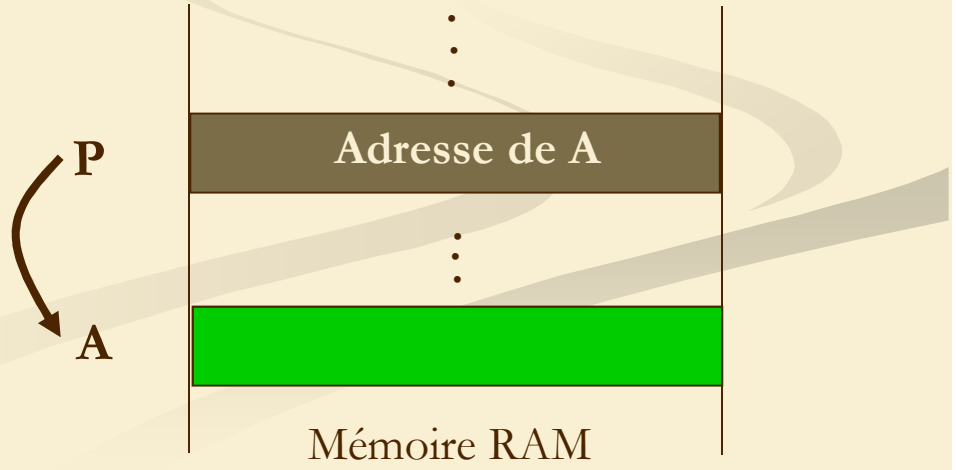
Pointeurs en langage C

- Introduction : Définition et Intérêts
- Déclaration et initialisation d’un pointeur
- Opérations élémentaires sur les pointeurs
- Applications des pointeurs
  - Pointeurs et Tableaux
  - Allocation dynamique de la mémoire

Pointeurs en langage C

Définition

- Un **pointeur** est une variable spéciale qui peut contenir l'**adresse** d'une autre variable.
- En C, chaque pointeur est limité à un type de données. Il peut contenir l'adresse d'une variable de ce type.
- Si un pointeur P contient l'adresse d'une variable A, on dit que **'P pointe sur A'.**



Pointeurs en langage C

Intérêts

- En C, l'utilisation de pointeurs est incontournable car ils sont étroitement liés à la représentation et manipulation des tableaux
- Les principales intérêts des pointeurs résident dans la possibilité de :
  - Allouer de la mémoire dynamique sur le TAS(1), ce qui permet la gestion de structures de taille variable. Par exemple, tableau de taille variable.
  - Permettre le passage par référence pour des paramètres des fonctions
  - Réaliser des structures de données récursives (listes et arbres)
  - …

(1) : Tas (ou heap) est une zone d'allocation dynamique , qui est une réserve de mémoire dans laquelle le programme peut puiser en cours d'exécution grâce à des fonctions prédéfinies (voir suite)

Pointeurs en langage C

Déclaration et initialisation d'un pointeur

Déclaration

- **Un pointeur est une variable dont la valeur est égale à l'adresse d'une autre variable. En C, on déclare un pointeur par l'instruction :**

type *nom_du_pointeur ;

où

- **type est le type de la variable pointée,**
- **l'identificateur nom_du_pointeur est le nom de la variable pointeur et**
- *** est l'opérateur qui indiquera au compilateur que c'est un pointeur.**

- **Exemple : short *p;**

On dira que :

p est un pointeur sur une variable du type short, ou bien

***p est de type short, c'est l'emplacement mémoire pointé par p.**

Pointeurs en langage C

Remarques :

- **A la déclaration d'un pointeur p, il ne pointe a priori sur aucune variable précise : p est un pointeur non initialisé.**

  Toute utilisation de p devrait être précédée par une initialisation.

- **la variable pointeur est représentée en mémoire soit sur 16 bits, 32 bits ou 64 bits.**

- **L'interprétation de la valeur d'une variable pointeur p :**

  Si p pointe sur une variable de type char Alors sa valeur donne l'adresse de l'octet ou cette variable est stockée.

  Si p pointe sur une variable de type short Alors sa valeur donne l'adresse du premier des 2 octets où la variable est stockée

  Si p pointe sur une variable de type float Alors sa valeur donne l'adresse du premier des 4 octets où la variable est stockée

Pointeurs en langage C

Initialisation

Pour initialiser un pointeur, le langage C fournit l'opérateur unaire &. Ainsi pour récupérer l'adresse d'une variable A et la mettre dans le pointeur P (P pointe vers A), on écrit :

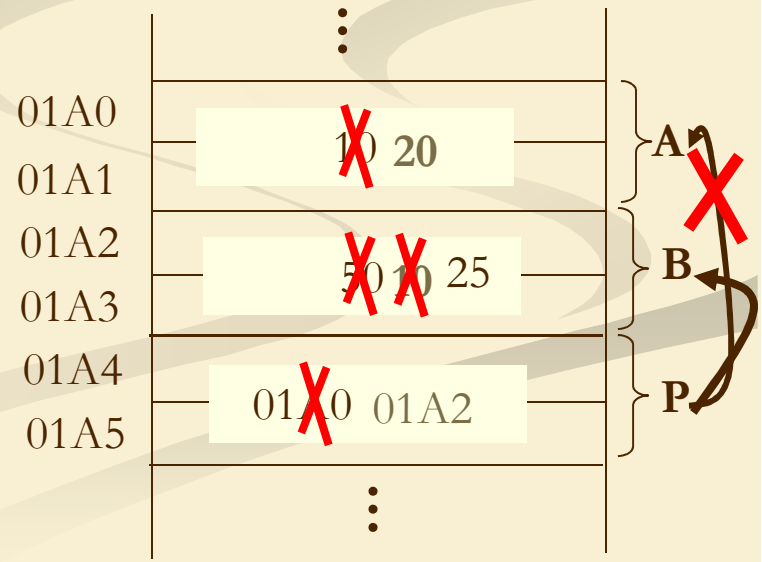
P = & A

Exemple 1 :

short A, B, *P; /*supposons que ces variables occupent la mémoire à partir de l'adresse 01A0 */

A = 10;
B = 50;
P = &A ; // se lit mettre dans **P** l'adresse de **A**
B = *P ; /* mettre dans **B** le contenu de l'emplacement mémoire pointé par **P***/
P = 20;** /*mettre la valeur **20** dans l'emplacement mémoire pointé par **P/
P = &B; // **P** pointe sur **B**

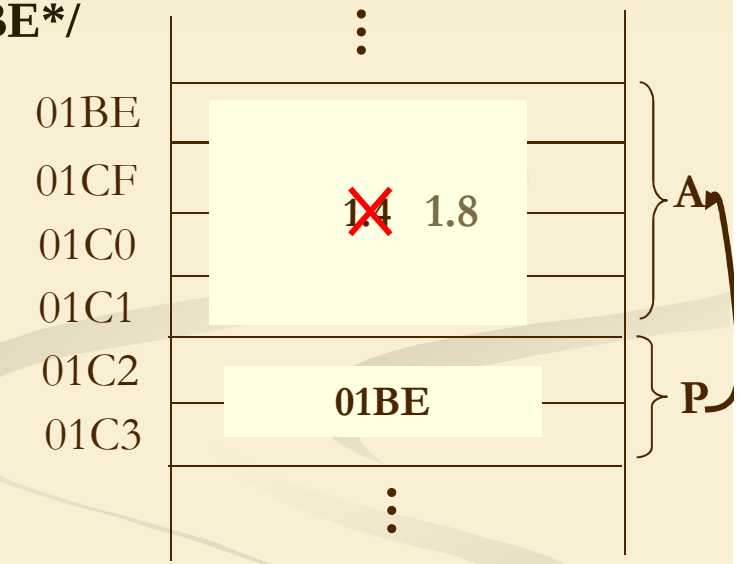
***P += 15;**



Pointeurs en langage C

Exemple 2 :

```
…….
float  a , *p;  /*supposons que ces variables sont représentées
                  en mémoire à partir de l'adresse 01BE*/
…….

  clrscr( ); // pour effacer l'écran → <conio.h>

  p = &a;

  printf("Entrer une valeur réelle:");
  scanf("%f",p);  // on saisie la valeur 1.4

  printf("\nAdresse de a = %x  Contenu de a = %f",p,*p);

  *p += 0.4;
  printf("\na = %f  " , a);
     …….
```



Affichage sur Ecran

Entrer une valeur réelle : 1.4
Adresse de a : 01BE Contenu de a = 1.4
a = 1.8

Pointeurs en langage C

Opérations élémentaires sur les pointeurs

- **L'opérateur & : 'adresse de' : permet d'obtenir l'adresse d'une variable.**
- **L'opérateur * : 'contenu de' : permet d'accéder au contenu d'une adresse.**
- **Si un pointeur P pointe sur une variable X, alors *P peut être utilisée partout où on peut écrire X.**

- **Exemple : long X=1, Y, *P Après l'instruction, P = &X ;**

On a :

Y = X + 1	**équivalente à**	**Y = *P + 1**
X += 2	**équivalente à**	***P += 2**
++X	**équivalente à**	**++ *P**
X++	**équivalente à**	**(*P)++**

Pointeurs en langage C

Opérations élémentaires sur les pointeurs (suite)

- Le seul entier qui puisse être affecté à un pointeur d'un type quelconque P est la constante entière 0 désignée par le symbole NULL défini dans <stddef.h>.

  On dit alors que le pointeur P pointe 'nulle part'.

- Exemple :

```
#include <stddef.h>
...
long  *p,  x , *q;
short  y = 10 , *pt = &y;
p = NULL ;  /* Correct */
p = 0 ;         /* Correct */
x = 0 ;
p = x ;        /* Incorrect ! bien que x vaille 0 */
q = &x ;
p = q ;        /* Correct : p et q pointe sur des variables de même type*/
p = pt ;       /* Incorrect : p et pt pointe sur des variable de type différent */
```

Pointeurs en langage C

- Applications des pointeurs
  - Pointeurs et Tableaux
  - Allocation dynamique de la mémoire

Pointeurs en langage C

Pointeurs et Tableaux

- **En C, il existe une relation très étroite entre tableaux et pointeurs. Ainsi, chaque opération avec des indices de tableaux peut aussi être exprimée à l'aide de pointeurs.**

- **Comme déjà mentionné , le nom d'un tableau représente l'adresse de son premier élément :**
  - **Tableau à une dimension (short T[30]) :**
    - **le nom T du tableau est un pointeur constant sur le premier élément du tableau (du 1er entier)**
    - **T et &T[0] contiennent l'adresse du premier élément du tableau.**
  - **Tableau à deux dimensions(short T[20][30]) :**
    - **le nom T est un pointeur constant sur le premier tableau (d'entiers).**
    - **T[i] est un pointeur constant sur le premier élément du (i+1)^{ème} tableau.**
    - **T et T[0] contiennent la même adresse mais leur manipulation n'est pas la même puisqu'ils ne représentent pas le même type de pointeur.**

Pointeurs en langage C

4.1 Accès aux composantes d'un tableau à une dimension par le biais d'un pointeur

- **En déclarant un tableau A de type short(short A [N]) et un pointeur P sur des variables entière (short *P),**
- **L'expression P = A crée une liaison entre le pointeur P et le tableau A en mettent dans P l'adresse du premier élément de A (de même P = &A[0]).**
- **A partir du moment où P = A, la manipulation du tableau A peut se faire par le biais du pointeur P. En effet :**

p	pointe sur A [0]	*p	désigne A [0]
p+1	pointe sur A [1]	*(p+1)	désigne A [1]
….			
p+i	pointe sur A [i]	*(p+i)	désigne A [i]

où $i \in [0 , N-1]$

Pointeurs en langage C

Exemple (Lecture et Affichage d'un vecteur par le biais d’un

```
#include <stdio.h>
#define N 10

void main()
{
   float T[N] , *pt ;
   int i ;









}
```

```
}
```

Pointeurs en langage C

Accès aux composantes d'une matrice par le biais d'un pointeur

- En déclarant une matrice A de type long (long A[M][N]) et un pointeur P sur des variables entières (long *P),

- l'expression P = A[0] crée une liaison entre le pointeur P et la matrice A en mettent dans P l'adresse du premier élément de la première ligne de la matrice A (P = &A[0][0]).

- A partir du moment où P = A[0], la manipulation de la matrice A peut se faire par le biais du pointeur P. En effet :

p	pointe sur	A[0][0]	et	* p	désigne A[0][0]
p + 1	pointe sur	A[0][1]	et	* (p + 1)	désigne A[0][1]
….					
p + N	pointe sur	A[1][0]	et	* (p + N)	désigne A[1][0]
p + N + 1	pointe sur	A[1][1]	et	* (p + N + 1)	désigne A[1][1]
….					
p + i * N + j	pointe sur	A[i][j]	et	* (p + i * N + j)	désigne A[i][j]

où $i \in [0 , M-1]$ et $j \in [0 , N-1]$.

Pointeurs en langage C

Exemple (Lecture et Affichage d'une matrice matérialisé par un pointeur)

```
#include <stdio.h>
#define M 4
#define N 10

void main()
{
 short A[M][N] ,  *pt ;
 int i, j ;

 /* lecture de la matrice ligne par ligne*/
 pt = &A[0][0] ;/* ou bien pt = A[0] ; */
 for (i = 0 ; i<M ; i++)
 {
     printf("\t ligne n° %d\n", i+1) ;
     for (j = 0 ; j<N ; j++)
        scanf("%hd", pt + i * N + j ) ;
 }
```

```
/* Affichage  de la matrice */
for (i = 0 ; i<M ; i++)
 {
     for (j = 0 ; j<N ; j++)
        printf("%hd\t", *( pt + i * N + j ) ) ;
     printf("\n") ;
 }

}
```

Pointeurs en langage C

Arithmétiques des pointeurs

Affectation par un pointeur sur le même type :

- Soient P1 et P2 deux pointeurs sur le même type de données.
- L'affectation : P1 = P2 ; fait pointer P1 sur le même objet que P2.

Addition et soustraction d'un nombre entier :

- Si P pointe sur l'élément A[i] d'un tableau, alors :
- P+n *pointe sur* A[i+n] et P-n *pointe sur* A[i-n]

Incrémentation et décrémentation d'un pointeur :

- Si P pointe sur l'élément A[i] d'un tableau, alors après l'instruction :
- P++ ; P *pointe sur* A[i+1]
- P += n ; P *pointe sur* A[i+n]
- P-- ; P *pointe sur* A[i-1]
- P -= n ; P *pointe sur* A[i-n]

Comparaison de deux pointeurs :

- On peut comparer deux pointeurs *de même type* par : <, >, <=, >=, == ou !=
- La comparaison de deux pointeurs qui pointent dans le même tableau est équivalente à la comparaison des indices correspondants.

Pointeurs en langage C

Allocation dynamique

- **La déclaration d'une variable tableau définit un tableau "statique" (il possède un nombre figé d'emplacements). Il y a donc un gaspillage d'espace mémoire en réservant toujours l'espace maximal prévisible.**

- **Il serait souhaitable que l'allocation de la mémoire dépend du nombre d'éléments à saisir. Ce nombre ne sera connu qu'à l'exécution : c'est l'allocation dynamique.**

Pointeurs en langage C

Fonctions d'allocation dynamique de la mémoire

- **En C, il existe principalement 3 fonctions prédéfinis pour gérer l' allocation dynamiquement de la mémoire**

Bibliothèque <stdlib.h>	

- **Chacune des fonctions malloc ou realloc, prend un bloc d'une taille donnée dans l'espace mémoire libre réservé pour le programme (appelé *tas* ou *heap*) et affecte l'adresse du début de la zone à une variable pointeur.**
- **S'il n'y a pas assez de mémoire libre à allouer dans le Tas, la fonction renvoie le pointeur NULL.**

Pointeurs en langage C

Fonctions malloc et free

- **malloc**

  <pointeur> = <type> malloc(<taille>);

  <type> est un type pointeur définissant le type de la variable <pointeur>

  <taille> est la taille, en octets, de la zone mémoire à allouer dynamiquement. <taille> est du type unsigned int.

  Si le type unsigned int est codé sur 2 octets alors on ne peut pas réserver plus de 2^{16} octets = 65536 octets à la fois.

  Si le type unsigned int est codé sur 4 octets alors on peut réserver jusqu'à 2^{32} octets = 4 Go à la fois.

  La fonction malloc retourne l'adresse du premier octet de la zone mémoire allouée. En cas d'échec, elle retourne NULL.

- **free**

  - **Si on n'a plus besoin d'un bloc de mémoire réservé dynamiquement par malloc, alors on peut le libérer à l'aide de la fonction free.**

    free(<pointeur>);

  - **Libère le bloc de mémoire désigné par le pointeur <pointeur>**

Pointeurs en langage C

Exemple (Allocation dynamique, Saisie et Affichage d'un tableau)

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

int  main()
{
   short  *pt;
   int N , i;
   printf("Entrez la taille N  du tableau \n") ;
   scanf("%d", &N) ;

   pt = ( short * ) malloc( N * sizeof( short ) );
   if (pt == NULL)
   {
      printf("Mémoire non disponible") ;
      system("pause") ;
      return 1;
    }
```

```
            printf("Saisie du tableau : ");
            for (  i = 0  ; i < N; i++)
                  scanf("%hd", pt + i ) ;

            printf("Affichage du tableau ") ;
            for ( i= 0  ; i < N; i++)
               printf("%hd\t", *( pt + i ) ) ;

            free( pt );

            return 0;
      }
```

Pointeurs en langage C

Exercices

Trouvez les erreurs dans les suites d'instruction suivantes :

a) **short *p , x = 34; *p = x;**

***p = x est incorrect parce que le pointeur p n'est pas initialisé**

b) **long x = 17 , *p = x; *p = 17;**

***p = x est incorrect. Pour que p pointe sur x à la déclaration, on écrit : *p = &x**

c) **double *q; long x = 17 , *p = &x; q = p;**

q = p incorrect. q et p deux pointeurs sur des types différent

d) **short x, *p; &x = p;**

&x = p incorrect. &x n'est pas une variable (lvalue) et par conséquent l'expression &x ne peut pas figurer à gauche d'une affectation.

e) **char mot[10], car = 'A', *pc = &car ; mot = pc;**

mot = pc incorrect.

mot est un pointeur constant (nom d'une variable tableau) par conséquent on ne peut pas changer sa valeur.

Pointeurs en langage C

Exercice :

Soit les déclarations suivantes :

```
int A[ ] = { 12 , 23 , 34 , 45 , 56 , 67 , 78 , 89 , 90 } , *P ;
int B[3][4] = {{-4 , 5 ,+10 ,-3 } , {} , { 12 , 60 , 30 , -10 } } , *Pt  ;
```

```
P = A ;  Pt = B[0] ;
```

Quelles valeurs ou adresses fournissent ces expressions:

```
*P+2  ;  *(P+2) ;  &A[4]-3 ;   A+3 ;  P+(*P-10)  ;  *(P+*(P+8)-A[7])  ;
*Pt+2 ;  *(Pt+2) ; B[2]+3  ;  B+2 ;  Pt+(*Pt+6)  ;  *(Pt+*(Pt+8)-2)  ;
```

Fonctions

- La programmation modulaire
- Définition, appel et déclaration d’une fonction
- Durée de vie des variables
- Passage des paramètres d’une fonction
- Fonctions récursives

Fonctions

La programmation modulaire

- Découper un programme en plusieurs parties appelées modules :
  - Un module est une entité de données et d'instructions fournissant une solution à une partie bien définie d'un problème plus complexe.
  - Un module peut faire appel à d'autres modules, leur transmettre des données et recevoir des données en retour.
  - L'ensemble des modules reliés doit résoudre le problème global.
- Pourquoi la programmation modulaire ? :
  - Un programme sur plusieurs pages devient difficile à comprendre et à maîtriser.
  - Il faut souvent répéter les mêmes instructions dans le texte du programme, ce qui entraîne un gaspillage de la mémoire.
  - Un module, une fois mis au point, peut être réutilisé par d'autres modules ou programmes (notion de réutilisation du code)
  - Un module peut être changé ou remplacé sans devoir toucher aux autres modules du programme.
- En C, la structuration d'un programme en sous-programmes (modules) se fait à l'aide de fonctions.

Fonctions

Définition, appel et déclaration d'une fonction

Définition d'une fonction

Une fonction est définie par un entête et un corps contenant les instructions à exécuter :

```
type  nom_fonction ( type_1 arg_1 , ... ,  type_n arg_n )
 {
        déclarations de variables locales
         liste d'instructions
}
```

- **La première ligne de cette définition est l'en-tête de la fonction. Dans cet en-tête, type désigne le type de la fonction, c'est-a-dire le type de la valeur qu'elle retourne. Une fonction qui ne renvoie pas de valeur est une fonction dont le type est spécifié par le mot clef void.**
- **Les arguments de la fonction, appelés paramètres formels, peuvent être de n'importe quel type. Enfin, si la fonction ne possède pas de paramètres, on remplace la liste de paramètres par le mot clef void.**
- **Le corps de la fonction débute éventuellement par des déclarations de variables, qui sont locales à cette fonction. Il se termine éventuellement par l'instruction de retour à la fonction appelante, return, dont la syntaxe est return expression ;**

Fonctions

Exemples :

- **Écrire :**
  - **une fonction, nommée affiche_bonjour, se contentant d'afficher « Bonjour tout le monde » (elle ne possède aucun argument ni valeur de retour).**
  - **Une fonction, nommée affiche_somme, qui affiche la somme de deux entiers (short) passés comme paramètres (elle ne possède aucune valeur de retour).**
  - **Une fonction, nommée produit, qui reçoit en paramètre deux entiers (int) et retourne leur produit (int).**
  - **Une fonction, nommée imprime_tab, qui affiche les éléments d'un tableau de réels (float). Le tableau ainsi que le nombre d'éléments du tableau sont les paramètres de la fonction (elle ne possède aucune valeur de retour).**

Fonctions

Appel d'une fonction

L'appel d'une fonction se fait par l'expression :

```
nom_fonction ( para_1 , para_2 , ... , para_n )
```

- **L'ordre et le type des paramètres effectifs de la fonction doivent concorder avec ceux donnés dans l'en-tête de la fonction.**
- **Les paramètres effectifs peuvent être des expressions.**
- **La virgule qui sépare deux paramètres effectifs est un simple signe de ponctuation ; il ne s'agit pas de l'opérateur virgule. Cela implique en particulier que l'ordre d'évaluation des paramètres effectifs n'est pas assuré et dépend du compilateur.**

Fonctions

Exemple :

- **Écrire une fonction, nommée puissance, qui calcule et retourne la valeur de x^n (double). Elle possède comme arguments x (double) et n (int).**

- **Écrire un programme C qui fait appel à la fonction puissance.**

```
#include<stdio.h>
 double  puissance(int n, double x)
{
  double p = 1.0 ;   // variable locale
  int i ;                  // variable locale
  for( i = 1 ; i <= n ; i++ )       p *= x ;  //calcul de xn
   return p ;          // valeur retournée
 }
 void main()
 {
  double    y  ,  z;
  scanf("%lf",&z); // appel de  la fonction scanf définie dans <stdio.h >
  y = puissance(3, z+1.0) ; // appel de la fonction  puissance
  printf("(%lf+1)^3 = %lf", z , y ); // appel de la fonction printf  <stdio.h>
 }
```

Fonctions

Déclaration d'une fonction

- **Le C n'autorise pas les fonctions imbriquées. La définition d'une fonction secondaire doit donc être placée soit avant, soit après son appel .**

- **Toutefois, il est indispensable que le compilateur connaisse la fonction au moment ou celle-ci est appelée. Si une fonction est définie après son premier appel , elle doit impérativement être déclarée au préalable.**

- **Une fonction secondaire est déclarée par son prototype, qui donne le type de la fonction et celui de ses paramètres, sous la forme :**

```
type nom_fonction( type_1 , ... , type_n );
```

Fonctions

Exemple :

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

long puissance ( int , int ) ; //Prototype de la fonction puissance

void main()
{    int a = 2, b = 5;
     printf("%ld\n", puissance( a , b ) );    //On affiche a^b
     system("pause");
}

long puissance (int x, int y)   //Définition de la fonction puissance x^y
{   int i ;
   long   p = 1 ;
  for ( i = 0 ; i < y ; i++ )   p*=x;
  return p;
}
```

Fonctions

Durée de vie et Classes d'allocation des variables

Durée de vie des variables

Les variables manipulées dans un programme C ne sont pas toutes traitées de la même manière. En particulier, elles n'ont pas toutes la même durée de vie. On distingue deux catégories de variables : Variables globales et variables locales :

1. toute variable définie à l'extérieur des fonctions, d'un fichier source, est une variable globale : elle est connue et utilisable partout dans n'importe quelle fonction de ce fichier (sauf si elle est masquée par une variable locale). Elle est allouée dans la zone d'allocation statique(segment de données).

2. toute variable définie à l'intérieur d'une fonction est une variable locale. Elle n'est connue qu'à l'intérieur de la fonction dans laquelle elle est définie. Son contenu est perdu d'un appel à l'autre de la fonction. Elle est allouée dans le segment pile. Une variable locale cache la variable globale ayant même nom.

Fonctions

Exemple (variables locale & globale) :

```
#include <stdio.h>
void fct(void)        //fonction prototype
int i ;               //variable globale

void main ( )
{ int k = 5;          //variable locale à main
  i = 3;
  fct();
  fct();
  printf ("k de main : %d \n", k );
  printf ("i : %d \n", i );
}

void fct(void)
{ int k = 1       //variable locale à fct
  printf ("i : %d  ***  k  : %d  \n",i, k );
  i++; k++;
}
```

Affichage

```
i : 3  ***  k : 1
i : 4  ***  k : 1
k de main : 5
i : 5
```

Remarques :

- Il n'existe aucun lien entre la variable k de main et la variable k de fct. Toute modification de l'une n'a aucune influence sur l'autre.
- On peut avoir accès et modifier la variable i, qui est globale, à, partir de main() et fct().
- La variable k de fct() est réinitialisée à 1 lors de chaque entrée dans fct().

Fonctions

Passage des paramètres d'une fonction

A l'**appel** d'une **fonction** avec **paramètres**, la **valeur** ou **l'adresse** **du paramètre effectif** est **transmise au paramètre formel correspondant**.

Passage par valeur :

- **Si le nom d'une variable (sauf le nom d'un tableau) apparaît dans l'appel d'une fonction, comme paramètre effectif, alors la fonction appelée reçoit la valeur de cette variable. Cette valeur sera recopiée dans le paramètre formel correspondant.**
- **Après l'appel de cette fonction, la valeur du paramètre effectif n'est pas modifiée.**

Passage par adresse :

- **Lorsqu'on veut qu'une fonction puisse modifier la valeur d'une variable passée comme paramètre effectif, il faut transmettre l'adresse de cette variable.**
- **La fonction appelée range l'adresse transmise dans une variable pointeur et la fonction travaille directement sur l'objet transmis.**
- **Un tableau est toujours passé par adresse puisque le nom d'un tableau est un pointeur constant (c'est-à-dire une adresse).**

Fonctions

Exemples (passage par valeur & passage par adresse):

```
#include <stdio.h>
void fct_val(int)     //passage d'argument par valeur
void fct_adr(int *) //passage d'argument par adresse
void main( )
{  int i = 4 ;
   fct_val( i );
   printf("Après passage d'argument par valeur   i :
         %d\n", i );
   fct_adr( &i );
   printf("Après passage d'argument par adresse  i :
         %d\n", i );
}
void fct_val ( int a ) { a = a + 3; }
```

Affichage

Après passage d'argument par valeur i : 4

Après passage d'argument par adresse i : 7

Écrire un programme C qui saisie et affiche un tableau d'entiers en utilisant des fonctions

```
#include <stdio.h>
void saisie_N( int *n )
{ printf("n (<=20) : ? " ) ; scanf("%d", n ); }

void saisie_T( int T[ ] , int n)
{int i ; for( i=0 ; i<n ; i++ ) scanf("%d",&T[i]);}

void affichage_T( int T[ ] , int n)
{int i ; for( i=0 ; i<n ; i++ ) printf("%d\t",T[i]);}

int  main( )
{   int T[20] , N;
    saisie_N( &N );
    saisie_T( T , N );
    affichage_T( T , N );
    return 0;
}
```

Fonctions

Exemples (passage par valeur & passage

- Écrire un programme C qui saisie et affiche une matrice d'entiers courts de M ligne et N colonne en utilisant des fonctions

```
#include <stdio.h>

void saisie_M_N( short * m , short  *n )
{
   printf("Entrer   m (<=20) et n (<=30) : ? " ) ;
    scanf("%hd", m , n );
}

void   saisie_A( short  B[ ][30] , int m , int n)
{int  i , j ;
   for( i = 0 ; i < m ; i++ )
    for ( j = 0 ; j < n ; j++ )
       scanf("%hd", &B[ i ][ j ] );
}
```

```
void  affichage_A ( short  C[ ][30] , short m, short n)
{ int  i , j  ;
    for ( i = 0 ; i < m ;  i++  )
   {
     for ( j = 0 ; j < n ; j++ )
        printf( "%hd\t", C[ i ][ j ] );
     printf("\n");
    }
}

int  main( )
{   short  A[20][30] , M , N;

    saisie_M_N( &M , &N );
    saisie_A( A , M , N );
    affichage_A( A , M,  N );
    return 0;
}
```

Fonctions

Fonctions récursives

- **Une fonction est récursive si elle contient dans sa définition un appel à elle-même**
- **L'ordre de calcul est l'ordre inverse de l'appel de la fonction.**
- **Procédé pratique : Pour trouver une solution récursive d'un problème, on cherche à le décomposer en plusieurs sous-problèmes de même type, mais de taille inférieure. On procède de la manière suivante :**
  - **Rechercher un cas trivial et sa solution (évaluation sans récursivité)**
  - **Décomposer le cas général en cas plus simples eux aussi décomposables pour aboutir au cas trivial.**

Exemple : (Calcul de la factorielle d'un entier)

```
unsigned long  fact ( unsigned long  n )
{
    if ( n == 0 )  return 1 ;    /* test d'arrêt évitant la récursivité à l'infini */
    else    return  n * fact(n - 1) ;
}
```

Fonctions

Exemples :

1. **Ecrire une fonction récursive qui calcule x^n , x un réel et n un entier positif.**

2. **Ecrire une fonction récursive qui calcule la valeur du $n^{ème}$ terme de la suite de Fibonacci.**

3. **Ecrire une fonction récursive qui calcule le nombre de combinaison de p sur n ($C_n^p = \frac{n!}{p!(n-p)!}$). On démontre que $C_n^p = C_{n-1}^{p-1} + C_{n-1}^p$ et on remarque que $C_n^0 = 1 \; et \; C_n^n = 1$**

4. **Ecrire une fonction récursive qui retourne le nombre de chiffre d'un entier positif donné.**

5. **Ecrire une fonction récursive qui affiche écrit à l'écran la représentation en binaire d'un entier strictement positif donné.**

Chaînes de caractères

- Définition
- Déclarations et Initialisations d’une chaîne de caractère
- Manipulations des chaînes de caractères
- Exemples

Chaînes de caractères

Définition

- En C, une chaîne de caractère est définie comme un tableau de caractère (alphanumérique, signe de ponctuation, caractère de contrôle,…) , dont le dernier élément vaut '\0'.
- C'est ce '\0' de fin qui est caractéristique des chaînes de caractères et toutes les fonctions permettant d'en manipuler supposent que ce '\0' de fin de chaîne est présent.
- Une chaîne de caractères est un tableau de caractères qui peut être manipulé comme étant un tableau de caractères ou d'une manière globale (sans le faire caractère par caractère) par le biais des fonctions prédéfinies.

Chaînes de caractères

Déclaration et initialisation d'une chaîne de caractère

Déclaration

char NomChaine [longueur] ; /* sous forme de tableau */

Exemple :

char Nom[20] ; /* La variable Nom est une chaîne ne pouvant contenir au plus que 19 caractères utiles */

Remarques :

- **Pour mémoriser une chaîne de N caractères, on a besoin de N+1 octets.**
- **Les traitements classiques (copie , concaténation, comparaison,…) sur les chaînes seront réalisées par le biais de fonctions prédéfinies.**
- **Le nom d'une chaîne de caractères est le représentant de l'adresse du 1er caractère de la chaîne.**

Chaînes de caractères

initialisation d'une chaîne de caractère

De même que les tableaux, les chaînes peuvent être initialisées lors de leur définition.

Exemples :

```
char ch1[8] = {'B', 'o', 'n', 'j', 'o', 'u', 'r', '\0'} ;
char ch2[10] = "Bonjour" ;

char ch3[] = "Bonjour" ;
char ch4[7] = "Bonjour" ;
char ch5[6] = "Bonjour" ;
char *ch6 = "Bonjour" ;
char jours_semaine[7][9]
```

Chaînes de caractères

Manipulation des chaînes de caractères

Les bibliothèques de fonctions de C contiennent une série de fonctions spéciales pour le traitement de chaînes de caractères : **stdio.h, string.h et stdlib.h**.

8.3.1 Fonctions de la Bibliothèque <stdio.h> (affichage de chaînes) :	
printf(..)	à utiliser avec le spécificateur de format %s pour afficher une chaîne de caractères. **Exemple** **char** ch[] = "Bonjour tout le monde" ; **printf**("%s", ch) ; // affichage normal
puts : puts(char *ch) ;	permet d'afficher tous les caractères jusqu au zéro de fin de chaîne. **puts** revient à la ligne en fin de l'affichage. **Exemple** **char** *ch = "Bonjour" ; /* ch pointe sur la chaîne constante "Bonjour" stockée quelque part en mémoire */ **puts**(ch) ; /* est équivalente à printf("%s\n", ch) ; Affiche Bonjour */

Chaînes de caractères

Manipulation des chaînes de caractères

8.3.1 Fonctions de la Bibliothèque <stdio.h> (lecture de chaînes) (suite)	
scanf (…)	à utiliser avec le spécificateur de format %s pour saisir une chaîne de caractères. **Exemple :** **char** lieu[25] ; **printf**("Entrez le lieu de naissance (nbre caractères <=24): ") ; **scanf**("%s", lieu) ;
gets : gets(char *ch) ;	Contrairement à **scanf**, la fonction **gets** permet de saisir des chaînes de caractères contenant des espaces et des tabulations. **Exemple** **char** string[80] ; **printf**("Entrez un texte (nbre caractères <=79): ") ; **gets**(string) ; **printf**("Le texte lu est : %s\n", string) ;

Chaînes de caractères

Manipulation des chaînes de caractères

8.3.2 Fonctions de la Bibliothèque <string.h> (traitement de chaînes) :	
strlen int strlen(char *s) ;	Retourne le **nombre de caractères** présents dans la chaîne s (sans compter '\0'). **Exemple** : char s[] = "Bonjour"; printf ("%d", strlen(s)); //Affiche 7
strcat strcat(char *s1, char *s2) ;	Ajoute une copie de la chaîne s2 à la fin de la chaîne s1. Le caractère final '\0' de s1 est écrasé par le 1er caractère de s2. **Exemple** : char ch1[30] = "Bonjour" , *ch2 = " tout le monde" ; strcat(ch1, ch2) ; printf("%s", ch1) ; //Affiche : Bonjour tout le monde

Chaînes de caractères

8.3.2 Fonctions de la Bibliothèque <string.h> (traitement de chaînes) :	
strcmp int strcmp(char *s1, char *s2)	Compare lexicographiquement les chaînes s1 et s2, et retourne une valeur : = 0 si s1 et s2 sont identiques < 0 si s1 précède s2 > 0 si s1 suit s2 **Exemple** : char ch1[]="abc" , ch2[]="aab" ; if (strcmp(ch1, ch2)==0) printf("identiques\n") ; else if (strcmp(ch1, ch2)>0) printf("%s précède %s\n", ch2, ch1) else printf("%s suit %s\n", ch2, ch1) ;

strcpy
strcpy(char *s1, char *s2) ;
	Copie la chaîne s2 dans s1 y compris le caractère '\0'.
	Exemple
	Char s1[20], s2[10];
	strcpy(s1, "Bonjour");
	strcpy(s2, s1);
	puts(s2); //Affiche Bonjour

Chaînes de caractères

8.3.2 Fonctions de la Bibliothèque <string.h> (traitement de chaînes) (suite)	
strncat char *strncat(char *s1, char *s2, int n) ;	Ajoute au maximum les n premiers caractères de la chaîne s2 à la chaîne s1. **Exemple** **char** ch1[20] = "Bonjour", *ch2 = " tout le monde" ; **strncat**(ch1, ch2, 5) ; // ch1 : Bonjour tout
strncmp int strncmp(char *s1, char *s2, int n) ;	Ici, la comparaison est effectuée sur les n premiers caractères. **Exemple** (Etant donnée deux chaînes ch1 et ch2) if (! strncmp(ch1 , ch2 , 3)) printf("Les 3 premier caractères sont identiques\n") ;
strncpy char *strncpy(char *s1, char *s2, int n) ;	Copie au plus les n premiers caractères de la chaîne s2 dans s1. La chaîne s1 peut ne pas comporter le caractère terminal si la longueur de s2 vaut n ou plus. **Exemple** **char** ch1[8] , *ch2 = "Bonjour" ; **strncpy**(ch1, ch2, 3) ; ch1[3] = '\0' ; **printf**("%s\n", ch1) ; //Affiche Bon

Chaînes de caractères

8.3.2 Fonctions de la Bibliothèque <string.h> (traitement de chaînes) (suite)	
strchr char *strchr(char *s, char c) ;	Recherche la 1ère occurrence du caractère c dans la chaîne s. Retourne un pointeur sur cette 1ère occurrence si c'est un caractère de s, sinon le pointeur NULL est retourné.

strrchr
char *strrchr(char *s, char c) ;
	Identique à **strchr** sauf qu'elle recherche la dernière occurrence du caractère c dans la chaîne s.
	Exemple
	char *ch = "Bonjour" ;
	puts(**strchr**(ch , 'o')) ; //Affiche : **onjour**
	puts(**strrchr**(ch , 'o')) ; //Affiche : **our**

strstr char *strstr(char *s1, char *s2) ;	Recherche la chaîne s2 dans la chaîne s1 . Retourne l'adresse de la première occurence de s2 dans s1 ou NULL si s2 n'est pas trouvé dans s1. **Exemple** #include <string.h> char *s1 = "Bonjour tout le monde" ; char *s2 = "tout" , *pch ; pch = strstr(s1, s2) ; printf("La sous-chaîne est : %s\n", pch) ; //Affiche : **tout le monde**

Chaînes de caractères

8.3.3 Fonctions de la Bibliothèque <stdlib.h> (conversion de nombres en chaînes de caractères) :	

char *itoa(int n, char *s, int b) ;	Convertissent l'entier n, représenté en base de numération b, dans la chaîne s.
	Exemple
	char s[20]; int i = 28;
	itoa(i , s , 2) ; // s : "11100 "
	itoa(i , s , 16); // s : "1C"
	itoa(i , s , 10); // s : "28"

char *ltoa(long n, char *s, int b) ;	
char *ultoa(unsigned long n, char *s, int b) ;	

Remarques :

- Si n est un entier négatif et b = 10, itoa et ltoa (pas ultoa) utilisent le 1er caractère de la chaîne s pour le signe moins.

- Si succès, les fonctions itoa, ltoa et ultoa renvoient un pointeur sur la chaîne résultante. Dans le cas contraire, elles retournent NULL.

Chaînes de caractères

<table>
<tr><th colspan="2">8.3.3 Fonctions de la Bibliothèque <stdlib.h> (conversion de chaînes de caractères en nombres) :</th></tr>
<tr><td>int atoi (char *s) ;</td><td rowspan="4">retourne la valeur numérique représentée par la chaîne s comme un int , long int ou double.
<u>Exemple :</u>
char s1[] = " -12" , s2[] = "+5e-1" ;
int x;
double y;
x = atoi(s1) ; // x : -12
y = atof(s2) ; // y : 0.5</td></tr>
<tr><td>long atol (char *s) ;</td></tr>
<tr><td>double atof (char *s) ;</td></tr>
</table>

<u>Remarques</u> :

- Les espaces au début de la chaîne de caractères s sont ignorés.
- La conversion s'arrête au 1er caractère non valide (c.-à-d. non convertible).
- Si aucun caractère n'est valide, les fonctions retournent zéro.

Chaînes de caractères

Exemples

Exemple 1 :

Lesquelles des chaines de caractères suivantes sont initialisées correctement? Corrigez les déclarations fausses.

a) `char a[5] = "Cinq" ;` Correct

b) ~~`char b[12] = "Un deux trois" ;`~~ → `char b[14] = "Un deux trois" ;`

c) `char c[] = "un\ndeux\ntrois\n" ;` Correct

d) ~~`char d[10] = 'x' ;`~~ → `char d[10] = "x"` ou `char d[10] ={ 'x' , '\0'}`

e) ~~`char e[] = 'abcdefg' ;`~~ → `char e[] = "abcdefg" ;`

f) ~~`char f[4] = { 'a' , 'b' , 'c'} ;`~~ → `char f[4] = { 'a' , 'b' , 'c' , '\0'} ;`

g) `char g[2] = { 'a' , '\0'} ;` Correct

h) `char i[4] = "'o'" ;` Correct

Chaînes de caractères

Exemples

Exemple 2 :

Ecrire un programme C qui demande à l'utilisateur de lui fournir un nombre entier entre 1 et 7 et qui affiche le nom du jour de la semaine ayant le numéro indiqué(lundi pour 1, mardi pour 2, …, dimanche pour 7).

```
#include <stdio.h>

main()
{
    char  jour[7] [9]= {"Lundi" , "Mardi" , "Mercredi" , "Jeudi" , "Vendredi" , "Samedi" ,
"Dimanche" };
    short  i ;

   do
   {
        printf("donner un nombre entier entre 1 et 7 : ");
        scanf("%hd",&i);
    }
    while ( ( i<=0 ) || ( i>7 ) );

   printf("le nom du jour  %d  de la semaine est %s ", i , jour[i-1]);
}
```

Chaînes de caractères

Exemples

Exemple 3 :

Ecrire un programme C qui trie un tableau de noms selon l'ordre alphabétique. Le tableau contiendra au maximum 30 noms et chaque nom ne dépasse pas 19 caractères. La méthode de tri Utilisée est le tri par sélection.

```
#include <stdio.h>
void  lire_noms ( char [ ][ ] , int ) ;
void  tri_noms ( char [ ][ ]  , int ) ;
void  afficher_noms ( char [ ][ ]  , int ) ;

main()
{
    char  noms[30]  [20] ;
    short N ;
    printf("donner le nombre des noms :? ");
    scanf("%hd",&N);
    lire_noms ( noms , N ) ;
    tri_noms ( noms , N ) ;
    afficher_noms ( noms , N ) ;
    system("pause"); return 0;
}
```

```
void  lire_noms ( char noms[ ][20 ] , int N)
{  int i;
   for( i = 0 ; i < N ; i++ )
   {  printf("\nDonner le nom %d",i+1);
      gets(noms[i]); }
}
void  afficher_noms ( char noms[ ][ 20]  , int N)
{   int i;
    printf("\nVoici la liste des noms triée :");
     for( i = 0 ; i < N ; i++ )
      printf("\n%s",noms[i]);
}
```

Chaînes de caractères

Exemples

Exemple 3 (suite):

Ecrire un programme C qui trie un tableau de noms selon l'ordre alphabétique. Le tableau contiendra au maximum 30 noms et chaque nom ne dépasse pas 20 caractères. La méthode de tri Utilisée est le tri par sélection.

```
void  tri_noms ( char noms[ ][20 ]  , int N) //La méthode du tri par sélection
{  char ch[20]; //variable chaine intermédiaire pour effectuer l'échange
    int i , j , pos_min ;
   for( i = 0 ; i < N-1 ; i++ )
   {   pos_min = i;  //le ième noms est le premier candidat
      for( j = i+1 ; j < N ; j++ )  if( strcmp( noms[pos_min] , noms[j] ) > 0 )
pos_min=j;
      if( pos_min != i ) //Permutation pour mettre le minimum à sa position
      {    strcpy( ch , noms[pos_min] );  strcpy( noms[pos_min] , noms[i] );
           strcpy( noms[i] , ch) ;
      }
    }
}
```

Types structures

- Type structure
- Déclaration d'une structure : struct
- Utilisation des structures :
  - champ par champ : accès aux champs via une variable ou par le biais d'un pointeur
  - dans leur ensemble : Opérations possibles sur les structures
- Union

Types structures et unions

Type structure

- **Une structure est un nouveau type de données composé de plusieurs champs (ou membres) qui sert à représenter un objet réel.**

- **Chaque champ est de type quelconque : simple (entier ; réel), pointeur ou composé (tableau ; structure ; …).**

- **Le nom d'une structure n'est pas un nom de variable. C'est un nom de type ou modèle de structure.**

Types structures et unions

Type structure : Exemples

- **Une date est un objet réel défini par : jour (entier), mois (entier ou chaîne) et année(entier).**
- **Un nombre complexe est défini par sa parties réelle (réel) et sa partie imaginaire (réel).**
- **Un étudiant est défini par : nom (chaîne), prénom (chaîne), num_CIN (chaîne), code (entier), num_CNE (entier),….**
- **Un article est défini par : numéro (entier), libellé (chaine), quantité en stock (entier), prix (réel).**

Types structures et unions

Déclaration d'une structure (struct)

```
struct  nom_structure
{
        type1   nom_champ1;
        type2   nom_champ2;
        …
        typeN   nom_champN;
} ;
```

Par l'intermédiaire du nom de la structure, on peut déclarer plusieurs variables de ce type de structure chaque fois que c'est nécessaire.

Types structures et unions

Déclaration d'une structure (Exemples)

Exemple 1 :

Un article est défini par : numéro (entier : short), libellé (chaîne de 29 caractères), quantité en stock (entier : short), prix (réel : float).

```
                                /* Déclaration du type structure article*/
struct article
{               short numero ;  // un numéro qui identifie l'article
                char libelle[30];  // le nom de l'article
                short qte_stock;  //la quantité disponible en stock de l'article
                float prix;   //le prix avec lequel est commercialisé l'article
} ;

                                /*Déclaration des variables du type structure article */

struct article art1 ; // art1 est une variable  structure article
struct article art1 , art2; // art1 et art2 deux variables structure article
struct article *Pt_art; //  Pt_date est une variable pointeur susceptible de pointer une variable
                        //  structure article
struct article tab_art[30] ; //  tab_art est une variable tableau de 30 éléments de type structure
                             // article : tableau des articles
```

Types structures et unions

Déclaration d'une structure (Exemples)

Exemple 2 :

Un étudiant est défini par : code (entier :short) ; nom (chaîne : 29) ; prénom (chaîne : 19) ; adresse (une adresse)

Une adresse est défini par : numéro de domicile (entier : short) ; nom de la rue (chaîne : 29) code postale (entier : short) ; nom de la ville (chaîne : 19) ; nom du pays (chaîne : 19).

```
struct adresse
{
    short n_domicile ;  /* numéro de la maison */
    char rue[30] ;      /* nom de la rue */
    short code_postale; // le code postale
    char ville[20] ;  /* nom de la ville */
    char pays[20] ;  /* nom du pays */
} ; // type structure adresse
```

```
struct etudiant
{
   short code ;  /* code de l'étudiant */
   char nom[30] ;     /* nom de l'étudiant */
   char prenom[20] ;  // prénom de l'étudiant
   struct adresse adr ;  // l'adresse de l'étudiant
} ; // type structure etudiant
```

```
/Déclaration des variables du type structure étudiant*/
struct etudiant etd1 , edt2 ,  // etd1 et edt2 deux variables structure étudiant
                *Pt_etd ,  // Pt_date est une variable pointeur susceptible de pointer une structure étudiant
                tab_etd[30]  ; //  tab_art est tableau de 30 étudiants
```

Types structures et unions

Utilisation de structures

Les structures peuvent être manipulées champ par champ ou dans leur ensemble.

Opérations sur les champs :

L'opérateur point

Accès à un champ d'une structure : `variable_structure.champ_structure`

Exemple :

```
struct article art;
 /* Initialisation, depuis le clavier, des champs de la structure art */
  scanf("%d %d %f",& art.numero , & art.qte_stoc, &art.prix) ;
  gets(art.libelle);

  /* Affichage du contenu de la structure art */
  printf("Cette article a pour :  \n");
  printf("\tnuméro : %d \n",art.numero);
  printf("\tlibellé : %s \n", art.libelle) ;
  printf("\tquantité en stck  : %d \n", art.qte_stock) ;
  printf("\tprix : %f \n", art.prix) ;
```

```
struct article
{
    short numero ;
   char libelle[30];
   short qte_stock;
   float prix;
} ;
```

Types structures et unions

Utilisation de structures :

Opérations sur les champs (suite):

Accès à un champ via un pointeur de structure :

pointeur_structure -> champ_structure

Le symbole moins '-' suivi du symbole supérieur '>'

Exemple :

```
struct article *pt_art, art ;
pt_art = &art ;

/* Initialisation, depuis le clavier, des champs de la structure art via la variable pointeur pt_art*/
  scanf("%d %d %f",&(pt_art->numero) , &(pt_art->qte_stock), &(pt_art->prix) ) ;
  gets(pt_art->libelle);

 /* Affichage du contenu de la structure art via la variable pointeur pt_art*/
printf("Cette article a pour :  \n");
printf("\tnuméro : %d \n", pt_art->numero);
printf("\tlibellé : %s \n", pt_art->libelle) ;
printf("\tquantité en stck  : %d \n", pt_art->qte_stock) ;
printf("\tprix : %f \n", pt_art->prix) ;
```

```
struct article
{
    short numero ;
   char libelle[30];
   short qte_stock;
   float prix;
} ;
```

Remarque : Il y a équivalence entre

art.nom_champ , pt_art->nom_champ et (*pt_art).nom_champ

Types structures et unions

Opérations sur les variables structures :

Initialisation à la déclaration : Il est possible d'initiation une variable structure lors de son déclaration

Exemple :

```
struct article art = {1, "Ecran TFT 19",12 , 2500.00} ;
```

```
struct article
{
    short numero ;
   char libelle[30];
   short qte_stock;
   float prix;
} ;
```

Affectation simple = : Les variables structures doivent être de même type

Exemple :

```
struct article art1, art2 = {1, "Ecran TFT 19" ,12 , 2500.00} ;
art1 = art2 ;
/*après cette affectation la structure art1 contient les mêmes données que art2*/
```

Opérateur d'adresse & : L'opérateur & permet de récupérer l'adresse d'une variable structure.

Exemple :

```
struct article art= {1, "Ecran TFT 19" ,12 , 2500.0} , *pt_art ;
pt_art = &art ;
pt_art->qte_stck = 10; // On modifie la quantité en stock
pt_art->prix = 2000.0;  //On modifie le prix : l'article est en promotion
```

Opérateur sizeof : L'opérateur sizeof permet de récupérer la taille d'un type ou d'une variable structure.

Exemple :

```
struct article art;
printf("taille en octets de la structure article : %d\n", sizeof(art)) ;
/* ou */      printf("taille en octets de la structure article : %d\n", sizeof(struct article)) ;
```

Types structures et unions

Exemple :

Ecrire un programme C qui manipule les nombres complexes : Saisie d'un nombre complexe, Affichage d'un nombre complexe, Addition et Multiplication de deux nombres complexes.

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


struct complexe {  double reelle , imaginaire ;  };

void Saisir (struct complexe *);

void Afficher ( struct complexe );

struct complexe Somme ( struct complexe ,
                        struct complexe );

void Multiplication ( struct complexe ,
          struct complexe  ,  struct complexe * );
```

```
int main()
{   struct complexe x , y , z;

    printf("Saisie de x : \n");   Saisir( &x ); Afficher( x );
    printf("Saisie de y : \n");   Saisir( &y ); Afficher( y );

    printf("Calcul de x+y : \n");
    z=Somme( x , y );
    Afficher( z );

    printf("Calcul de x*y : \n");
    Multiplication ( x , y , &z );
    Afficher ( z );

     system("PAUSE");
     return 0;
}
```

Types structures et unions

```
void Saisir (struct complexe *ptr_NC)
{
    printf("La partie reelle  : ?"); scanf("%lf", &(ptr_NC->reelle ));
    printf("La partie imaginaire  : ?"); scanf("%lf", &(ptr_NC->imaginaire) );
}
void Afficher (struct complexe NC)
{
       printf("La valeur du nombre complexe est %lf + %lf i \n", NC.reelle , NC.imaginaire );
}
struct complexe Somme (struct complexe NC1 , struct complexe NC2)
{  struct complexe NC;
   NC.reelle = NC1.reelle+NC2.reelle;
   NC.imaginaire = NC1.imaginaire + NC2.imaginaire;
   return NC;
}
void Multiplication ( struct complexe NC1 , struct complexe NC2 , struct complexe *ptr_NC )
{
    ptr_NC->reelle = NC1.reelle * NC2.reelle  - NC1.imaginaire * NC2.imaginaire ;
    ptr_NC->imaginaire = NC1.reelle * NC2.imaginaire + NC2.reelle * NC1.imaginaire ;
}
```

Types structures et unions

Union

Déclaration

```
union nom_uion
{
        type1   nom_champ1;
        type2   nom_champ2;
        …
        typeN   nom_champN;
} ;
```

un objet union est constitué d'un seul champ choisi parmi tous les champs définis.

Fichiers

Introduction

- **Le C offre la possibilité de lire et d'écrire des données dans un fichier.**
- **Les accès à un fichier se font par l'intermédiaire d'une mémoire-tampon (*buffer*), ce qui permet de réduire le nombre d'accès aux périphériques (disque...).**
- **Pour pouvoir manipuler un fichier, un programme a besoin d'un certain nombre d'informations : l'adresse de l'endroit de la mémoire-tampon où se trouve le fichier, la position de la tête de lecture, le mode d'accès au fichier (lecture ou écriture) ...Ces informations sont rassemblées dans une structure dont le type, FILE *, est défini dans stdio.h. Un objet de type FILE * est appelé *flot de données* (en anglais, stream).**

Fichiers

Ouverture et fermeture d’un fichier

Fonction fopen

Fonction de type FILE* ouvre un fichier et lui associe un flot de données.

Syntaxe **fopen("*nom-de-fichier*","*mode*")**

- **Elle retourne un flot de données et si erreur elle retourne le pointeur NULL.**
- **Le premier argument de fopen est le nom du fichier en une chaîne de caractères.**
- **Le second argument, *mode*, est une chaîne de caractères qui spécifie le mode d'accès au fichier. Les spécificateurs de mode d'accès diffèrent suivant le type de fichier considéré.**
- **Les *fichiers textes*, pour lesquels les caractères de contrôle (retour à la ligne ...) seront interprétés en tant que tels lors de la lecture et de l'écriture ;**
- **les *fichiers binaires*, pour lesquels les caractères de contrôle ne sont pas interprétés.**

Fichiers

Ouverture et fermeture d’un fichier

Fonction fopen

Modes d’accès

"r"	ouverture d'un fichier texte en lecture
"w"	ouverture d'un fichier texte en écriture
"a"	ouverture d'un fichier texte en écriture à la fin
"rb"	ouverture d'un fichier binaire en lecture
"wb"	ouverture d'un fichier binaire en écriture
"ab"	ouverture d'un fichier binaire en écriture à la fin
"r+"	ouverture d'un fichier texte en lecture/écriture
"w+"	ouverture d'un fichier texte en lecture/écriture
"a+"	ouverture d'un fichier texte en lecture/écriture à la fin
"r+b"	ouverture d'un fichier binaire en lecture/écriture
"w+b"	ouverture d'un fichier binaire en lecture/écriture
"a+b"	ouverture d'un fichier binaire en lecture/écriture à la fin

Fichiers

Ouverture et fermeture d’un fichier

Fonction fopen

Modes d’accès

- Si le mode contient la lettre **r**, le fichier doit exister.
- Si le mode contient la lettre **w**, le fichier peut ne pas exister. Dans ce cas, il sera créé. Si le fichier existe déjà, son ancien contenu sera perdu.
- Si le mode contient la lettre **a**, le fichier peut ne pas exister. Dans ce cas, il sera créé. Si le fichier existe déjà, les nouvelles données seront ajoutées à la fin du fichier précédent.

Flots standart

Ils peuvent être utilisés en C sans les ouvrir ou de les fermer :

- **stdin** (standard input) : unité d'entrée (le clavier) ;
- **stdout** (standard output) : unité de sortie (l'écran) ;
- **stderr** (standard error) : unité d'affichage des messages d'erreur (l'écran).

Fichiers

Ouverture et fermeture d’un fichier

Fonction fclose

Elle permet de fermer le flot qui a été associé à un fichier par la fonction fopen.

Syntaxe **fclose(*flot*)**

- **où *flot* est le flot de type FILE* retourné par la fonction fopen correspondant.**
- **La fonction fclose retourne un entier qui vaut zéro si l'opération s'est déroulée normalement (et une valeur non nulle en cas d'erreur).**

Fichiers

Les E\S formatés

Fonction d’écriture fprintif

Analogue à printf, permet d'écrire des données dans un fichier.

Syntaxe

fprintf(*flot*,"*chaîne de contrôle*",*expression-1*, ..., *expression-n*)

- **où *flot* est le flot de données retourné par la fonction fopen. Les spécifications de format utilisées pour la fonction fprintf sont les mêmes que pour printf.**

Fichiers

Les E\S formatés

Fonction de lecture fscanf

Analogue à scanf, permet de lire des données dans un fichier.

Syntaxe

fscanf(*flot*,"*chaîne de contrôle*",*argument-1*,...,*argument-n*)

- **où *flot* est le flot de données retourné par la fonction fopen. Les spécifications de format utilisées pour la fonction fscanf sont les mêmes que pour scanf.**

Fichiers

Impression et lecture de caractères

- Similaires aux fonctions getchar et putchar, les fonctions fgetc et fputc permettent respectivement de lire et d'écrire un caractère dans un fichier.

- La fonction fgetc, de type int, retourne le caractère lu dans le fichier. Elle retourne la constante EOF lorsqu'elle détecte la fin du fichier.

Prototype : **int fgetc(FILE* *flot*);**

- où *flot* est le flot de type FILE* retourné par la fonction fopen. Comme pour la fonction getchar, il est conseillé de déclarer de type int la variable destinée à recevoir la valeur de retour de fgetc pour pouvoir détecter correctement la fin de fichier.

Fichiers

Impression et lecture de caractères

La fonction fputc écrit *caractere* dans le flot de données :

Prototype **int fputc(int *caractere*, FILE **flot*)**

- Elle retourne l'entier correspondant au caractère (ou la constante EOF en cas d'erreur).

- Il existe également deux versions optimisées des fonctions fgetc et fputc qui sont implémentées par des macros. Il s'agit respectivement de getc et putc. Leur syntaxe est similaire à celle de fgetc et fputc :
- int getc(FILE* *flot*)
- int putc(int *caractere*, FILE **flot*)

Fichiers

Exemple

```
#include <stdio.h>
#include <stdlib.h>
 #define ENTREE "entree.txt"
#define SORTIE "sortie.txt"
int main(void) {

FILE *f_in, *f_out; int c;
if ((f_in = fopen(ENTREE,"r")) == NULL) {
            fprintf(stderr, "\nErreur: Impossible de lire le fichier %s\n",ENTREE);
            return(EXIT_FAILURE); }
if ((f_out = fopen(SORTIE,"w")) == NULL) {
            fprintf(stderr, "\nErreur: Impossible d'ecrire ds le fichier %s\n", \ SORTIE);
            return(EXIT_FAILURE); }
while ((c = fgetc(f_in)) != EOF)
            fputc(c, f_out);
fclose(f_in);
fclose(f_out);
return(EXIT_SUCCESS); }
```

Fichiers

Relecture d'un caractère

- Il est possible de replacer un caractère dans un flot au moyen de la fonction ungetc :

Prototype **int ungetc(int *caractere*, FILE **flot*);**

- Cette fonction place le caractère *caractere* (converti en unsigned char) dans le flot *flot*. En particulier, si *caractere* est égal au dernier caractère lu dans le flot, elle annule le déplacement provoqué par la lecture précédente. Toutefois, ungetc peut être utilisée avec n'importe quel caractère (sauf EOF).

Fichiers

Exemple

```
#include <stdio.h> #include <stdlib.h>
#define ENTREE "entree.txt"
int main(void) {
FILE *f_in; int c;
if ((f_in = fopen(ENTREE,"r")) == NULL) {
        fprintf(stderr, "\nErreur: Impossible de lire le fichier %s\n",ENTREE);
        return(EXIT_FAILURE); }
while ((c = fgetc(f_in)) != EOF) {
        if (c == '0')
                ungetc('.',f_in);
        putchar(c); }
 fclose(f_in);
return(EXIT_SUCCESS); }
```

sur le fichier entree.txt dont le contenu est 097023 affiche à l'écran 0.970.23

Fichiers

Les fonctions entrées-sorties binaires

- Elle permettent de transférer des données dans un fichier sans transcodage. Elles sont donc plus efficaces que les fonctions d'E\S standard, mais les fichiers produits ne sont pas portables (le codage dépend des machines).
- Elles sont utiles pour manipuler des données de grande taille ou ayant un type composé.

Prototypes:

```
size_t fread(void *pointeur, size_t taille, size_t nombre, FILE *flot);
size_t fwrite(void *pointeur, size_t taille, size_t nombre, FILE *flot);
```

- où *pointeur* est l'adresse du début des données à transférer, *taille* la taille des objets à transférer, *nombre* leur nombre.
- le type size_t, défini dans stddef.h, correspond au type du résultat de l'évaluation de sizeof. Il s'agit du plus grand type entier non signé.
- La fonction fread lit les données sur le flot *flot* et la fonction fwrite les écrit. Elles retournent toutes deux le nombre de données transférées.

Fichiers

- **Les fonctions entrées-sorties binaires**
- #include <stdio.h>
- #include <stdlib.h>
- #define NB 50
- #define F_SORTIE "sortie"
- int main(void) {
- FILE *f_in, *f_out; int *tab1, *tab2; int i;
- tab1 = (int*)malloc(NB * sizeof(int)); tab2 = (int*)malloc(NB * sizeof(int));
- for (i = 0 ; i < NB; i++) tab1[i] = i;
- if ((f_out = fopen(F_SORTIE, "w")) == NULL) {
- fprintf(stderr, "\nImpossible d'ecrire dans le fichier %s\n",F_SORTIE);
- return(EXIT_FAILURE); }
- fwrite(tab1, NB * sizeof(int), 1, f_out);
- fclose(f_out);
- if ((f_in = fopen(F_SORTIE, "r")) == NULL) {
- fprintf(stderr, "\nImpossible de lire dans le fichier %s\n",F_SORTIE);
  return(EXIT_FAILURE); }
- fread(tab2, NB * sizeof(int), 1, f_in);
- fclose(f_in);
- for (i = 0 ; i < NB; i++) printf("%d\t",tab2[i]); printf("\n");
- return(EXIT_SUCCESS); }

Fichiers

Positionnement dans un fichier

- Les fonctions d'E\S permettent d'accéder à un fichier en *mode séquentiel* : les données du fichier sont lues ou écrites les unes à la suite des autres. Il est également possible d'accéder à un fichier en *mode direct*, c'est-à-dire que l'on peut se positionner à n'importe quel endroit du fichier.
- La fonction fseek permet de se positionner à un endroit précis ;

Prototype : int fseek(FILE **flot*, long *deplacement*, int *origine*);

- Où *deplacement* détermine la nouvelle position en nombre d'octets par rapport à l'origine dans le fichier, *origine* peut prendre trois valeurs :
- SEEK_SET (égale à 0) : début du fichier ;
- SEEK_CUR (égale à 1) : position courante ;
- SEEK_END (égale à 2) : fin du fichier.
- La fonction int rewind(FILE **flot*); permet de se positionner au début du fichier. Elle est équivalente à fseek(*flot*, 0, SEEK_SET);
- La fonction long ftell(FILE **flot*); retourne la position courante dans le fichier (en nombre d'octets depuis l'origine).