Introduction au langage C

Introduction

- Le langage C est développé dans les années 1970 par par Dennis Ritchie et Brian Kernighan aux laboratoires Bell d'AT &T.
- Le C est un langage multi plate-forme : Windows, Mac, Linux, nombreux micro-processeurs
- C'est un langage relativement simple à apprendre et à mettre en œuvre. Il reste la référence en matière de programmation.

Introduction

- **C** est un langage compilé (par opposition aux langages interprétés).
- Cela signifie qu'un programme **C** est d´écrit par un fichier texte, appelé fichier source non exécutable par le microprocesseur il faut le traduire en langage machine
- Cette opération est effectuée par un programme appelé compilateur.

Création d'un programme

- **Edition du programme :** écriture du programme source sous un éditeur de texte soit quelconque, soit spécialisé (l'éditeur généralement associé à l'environnement C utilisé) création d'un ou de plusieurs fichiers source avec une extension ".c".

- **Compilation du programme** : traduction du programme source en langage machine ou code objet interprété par le processeur. Cette compilation s'effectue en deux étapes :

Création d'un programme

- **Le préprocesseur :** le fichier source est analysé par le préprocesseur qui examine toutes les lignes commençant par le caractère # et réalise des manipulations sur le code source du programme (remplacement de texte, inclusion de fichiers, compilation conditionnelle).

- **La compilation :** cette étape est effectuée par le compilateur qui réalise en fait une vérification syntaxique du code source et s'il n'y a pas d'erreur, il crée un fichier avec une extension ".obj". Le fichier objet est incomplet pour être exécuter car il contient par exemple des appels de fonctions ou des références à des variables qui ne sont pas définies dans le même fichier.

Edition des liens : cette étape est effectuée par le linkeur qui réalise les liens entre différents programmes objets pour obtenir un programme exécutable.

Un exemple de programme en langage C

Voici un exemple de programme en langage C,

```
#include <stdio.h>
#include <math.h>
#define NFOIS 5

      // mon premier programme
      /* main function: must return an integer */

int  main()
{
   printf("Hello world !\n");
/* print on the screen */

   return 0; /* return value */
}
```

Directives pour le préprocesseur

Commentaires en ligne avec //

Commentaires en ligne avec /* */

Corps du programme principal

Généralités.

❖ Fichiers Headers.

- ils sont inclus avec la commande #include (ex : #include <stdio.h>
- Les fichiers .h inclus au début sont les headers
- Ils permettent d'utiliser des fonctions déjà programmées (ex : printf)

❖ La fonction main.

- C'est le point d'entrée de tout programme C
- Quand un fichier est exécuté, le point de départ est la fonction main
- A partir de cette fonction, le programme se déroule selon les choix du programmeur
- Il peut y avoir d'autres fonctions appelée dans le main
- Tout programme C doit avoir une et une seule fonction main

Généralités.

❖ Les commentaires.

- Les commentaires sont ignorés par le compilateur
- Les commentaires s'écrivent entre : /* */

  /* commentaire */
- On peut aussi utiliser les commentaires C++ : //

  // commentaire

❖ définition de macros : #define.

- se déclare juste après les **incude**
- par convention, les macros ont un nom majuscule.

Principales caractéristiques

- Programmation modulaire multi-fichiers : un ensemble de programmes déjà mis au point pourra être réuni pour constituer une librairie.
- Langage déclaratif c'est-à-dire que tout objet (variables, fonctions) doit être déclaré avant d'être utilisé.
- Langage transportable c'est-à-dire séparation entre ce qui est algorithmique (déclarations, instructions) et tout ce qui est en interaction avec le système (allocation mémoire, entrées-sorties).
- Jeu d'opérateurs très riche.
- Faible contrôle des types de données.

Eléments de base : Variables

➢ Variables

Une variable est un emplacement de la mémoire dans lequel est stockée une valeur. Chaque variable porte une nom et c'est ce nom qui sert a identifié l'emplacement de la mémoire représente par cette variable.

➢ Déclaration

Le langage C est un langage déclaratif, toutes les variables doivent être déclarées avant utilisation. La déclaration se fait à l'aide de 2 paramètres :

<type> <identificateur>

Eléments de base : Variables

Exemple de déclaration d'une variable en C

```
#include<stdio.h>
int main ( )
    {
        int variable1 , variable2 ;
        int variable3 , variable4;
        return 0 ;
    }
```

l'exemple ci dessous est basées sur les variables de type numérique entier. Le type qui y correspond, en C, est int .

Eléments de base : Identificateur

➢ Identificateur

Un identificateur est un nom donné aux diverses objets d'un programme

- Il est formé de lettres et de chiffres. Le 1er caractère doit obligatoirement être une lettre ou bien le caractère (_).
- Il peut contenir jusqu'à 31 caractères minuscules et majuscules mais pas de caractères accentués
- Il y a un certain nombre de noms de variables réservés. Leur fonction est prévue par la syntaxe du langage C et ils ne peuvent pas être utilisés dans un autre but.

Eléments de base : Mots réservés

- **Mots réservés (mots-clés)**

auto	extern	sizeof
break	float	static
case	for	struct
char	goto	switch
const	if	typedef
continue	int	union
default	long	unsigned
do	register	void
double	return	volatile
else	short	while
enum	signed	

Eléments de base : les types de base

Le langage C contient des types de base qui sont les entiers, les réels simple et double précision et les caractères que l'on identifie à l'aide des mots-clés **int**, **float**, **double** et **char** respectivement.

Il existe aussi un type ensemble vide : le type **void**.

Les mots-clés **short** et long permettent d'influer sur la taille mémoire des entiers et des réels.

Eléments de base : les types de base

Syntaxe	Type	Taille mémoire
char	caractere	1
Short int	entier court	2
int	entier par defaut	4
Long int	entier long	8
float	reel simple precision	4
double	reel double precision	8
Long double	reel precision etendue	10

Eléments de base : les types de base

- La taille d'un entier par défaut est soit 2 soit 4 octets (dépend de la machine). 4 octets est la taille la plus courante.
- La taille d'un entier court est en général 2 octets et celle d'un entier long 4 octets.
- La taille d'un entier court est inferieure ou égale a la taille d'un entier par défaut qui est elle-même inferieure ou égale a celle d'un entier long.
- Les types short int et long int peuvent être abrégés en short et long.
- Le type char occupe un octet. Un caractère est considère comme un entier qu'on pourra donc utiliser dans une expression arithmétique.

Eléments de base : les types de base

- Les deux mots-clés unsigned et signed peuvent s'appliquer aux types caractère et entier pour indiquer si le bit de poids fort doit être considère ou non comme un bit de signe.

- Les entiers sont signes par défaut, tandis que les caractères peuvent l'être ou pas suivant le compilateur utilise.

- Une déclaration telle que unsigned char permettra de designer une quantité comprise entre 0 et 255, tandis que signed char désignera une quantité comprise entre -128 et +127.

- De même unsigned long permettra de designer une quantité comprise entre 0 et 2^{32}-1, et long une quantité comprise entre -2^{31} et 2^{31}-1.

Déclaration des objets (constantes et variables)

➢ **Constante:**

Une constante est une valeur portant un nom, contrairement aux variables, elles ne sont pas modifiables. Les constantes en C sont non typées, on les définit dans l'entête de la source, juste en dessous des **#include**.

La syntaxe est

#define <NOM CONSTANTE> <valeur Constante>

Exemple :

```
#define  N  50
```

définit une constante N qui a la valeur 50

Déclaration des objets (constantes et variables)

Il est aussi possible de déclarer que la valeur d'une variable ne doit pas changer lors de l'exécution du programme.

Par exemple, avec :

```
const int n = 20 ;
```

Déclaration des objets (constantes et variables)

Les constantes chaînes de caractères

- Une constante chaînes de caractères est une suite de caractères entourée du signe ".
- Toutes les notations de caractères (normale, octale, hexadécimale) sont utilisables dans les chaînes de caractères.

Exemple :

"Bonjour"

Déclaration des objets (constantes et variables)

➢ Les énumérations

Les **énumérations** sont des types définissant un ensemble de constantes qui portent un nom que l'on appelle **enumérateur**.

Elles servent a rajouter du sens a de simples numéros, a définir des variables qui ne peuvent prendre leur valeur que dans un ensemble ni de valeurs possibles identifiées par un nom symbolique.

Syntaxe

```
enum { liste des identificateurs }.
```

Déclaration des objets (constantes et variables)

➢ Les énumérations

Les constantes figurant dans les **énumérations** ont une valeur entière affectée de façon automatique par le compilateur en partant de **0** par défaut et avec une progression de **1**.

Exemple :

```
enum {LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE};
```

définit les identificateurs LUNDI, ... DIMANCHE comme étant des constantes de type int, et leur donne les valeurs 0, 1, ... 6. Par défaut, le premier identificateur est associé à la valeur 0.

On peut donner des valeurs particulières aux constantes en écrivant par exemple :

```
enum {LUNDI = 20, MARDI = 30, MERCREDI, JEUDI = 40, VENDREDI, SAMEDI, DIMANCHE};
```

Les opérateurs

Le langage C comporte de très nombreux opérateurs.

➢ Les opérateurs arithmétiques

Opérateur	Signification
+	**Addition**
-	**Soustraction**
*	**Multiplication**
/	**Division**
% ou mod	**Modulo : le reste de la division de 2 valeurs entières**

Leurs opérandes peuvent être des entiers ou des flottants hormis pour l'opérateur % qui agit uniquement sur des entiers.

lorsque les types des deux opérandes sont différents alors il y a conversion implicite dans le type le plus fort suivant certaines règles.

Les opérateurs relationnels

Pour exprimer les conditions, on utilise les opérateurs relationnels suivants :

Opérateur	Signification
==	Égal
<	Inférieur
>	Supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal
!=	différent

Les opérateurs logiques

C dispose de trois opérateurs logiques classiques que sont:

Opérateur	Signification
&&	**Et logique**
||	**Ou logique**
!	**Négation logique**

Le type booléen n'existe pas en C. Le résultat d'une expression logique vaut 1 si elle est vraie et 0 sinon. Réciproquement, toute valeur non nulle est considérée comme vraie et la valeur nulle comme fausse.

Les opérateurs logiques

- **(a<b) && (c<d)**

prend la valeur 1 (vrai) si les deux expressions a<b et c<d sont toutes deux vraies (de valeur 1), et prend la valeur 0 (faux) dans le cas contraire.

- **(a<b) || (c<d)**

prend la valeur 1 (vrai) si l'une au moins des deux conditions a<b et c<d est vraie (de valeur 1), et prend la valeur 0 (faux) dans le cas contraire.

- **! (a<b)**

prend la valeur 1 (vrai) si la condition a<b est fausse (de valeur 0) et prend la valeur 0 (faux) dans le cas contraire. Cette expression est équivalente à : a>=b.

En C, les opérateurs produisent un résultat numérique (de type int)

L’opérateur d'affectation

En C, l’affectation est un operateur à part entière. Elle est symbolisée par le signe =. Sa syntaxe est la suivante :

Variable = expression

Le terme de gauche de l’affectation peut être une variable simple, un élément de tableau mais pas une constante.

Cette expression a pour effet d’évaluer expression et d’affecter la valeur obtenue à *variable*. De plus, cette expression possède une valeur, qui est celle *expression*.

Ainsi, l’expression **i = 5** vaut 5.

L’opérateur d'affectation

i = 5 est une expression qui réalise l'affectation de la valeur 5 à i. Cet opérateur (=) peut faire intervenir d'autres expressions comme, c=b + 3

La faible priorité de cet opérateur = (elle est inférieure à celle de tous les opérateurs arithmétiques et de comparaison) fait qu'il y a d'abord évaluation de l'expression b + 3. La valeur ainsi obtenue est ensuite affectée à c.

En revanche, il n'est pas possible de faire apparaître une expression comme premier opérande de cet opérateur =.

Ainsi, l'expression suivante n'aurait pas de sens :c +5 =x

Les opérateurs d'affectation

A part le signe "=", il y a des opérateurs d'affectation combinés qui modifient la valeur courante de la variable intervenant dans l'évaluation de l'expression.

opérateurs	opérations équivalentes
+=	i = i + 20, on peut écrire i += 20 i = i + k, on peut écrire i += k
-=	i = i - 20, on peut écrire i -= 20
*=	i = i * 3, on peut écrire i *= 3
/=	i = i / 3, on peut écrire i /= 3
i++	peut s'écrire i = i+1 (post-incrémentation) incrémentation après utilisation de la valeur
++i	pré-incrémentation c'est-à-dire incrémentation avant utilisation de la valeur
i--	peur s'écrire i = i-1 (post-décrémentation)
--i	pré-décrémentation

Les opérateurs d'incrémentation (++) et de décrémentation (--) sont des opérateurs unaires permettant respectivement d'ajouter et de retrancher 1 au contenu de leur opérande. Cette opération est effectuée après ou avant l'évaluation de l'expression suivant que l'opérateur suit ou précède son opérande.

Instructions d'entrée et de sortie

L'échange d'information entre l'ordinateur et les périphériques tels que le clavier et l'écran est une étape importante que tout programme utilise. En C, les bibliothèques de fonctions comprises dans les fichiers d'en-tête (header), sont incluses dans le fichier source avec la directive: **#include**.

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

Fonctions usuelles comprises dans stdio.h

printf (…) : fonction de sortie. - écrit à l'écran les données fournies par l'ordinateur.
scanf (…) : fonction d'entrée (dans le programme) des données saisies à l'écran (lues).
getchar(…) : lit un caractère en entrée.
putchar (…) : affiche un caractère à l'écran.
gets (…) : lit une chaîne en entrée.
puts (…) : affiche une chaîne à l'écran.

La fonction printf

Elle permet d'afficher à l'écran un texte avec les variables. Sa syntaxe est la suivante :

printf(format, liste d'expression);
printf("premier argument %format_arg2 % format_ arg3", arg2, arg3);

Le premier argument de printf contient une chaîne de caractères qui est affichée à l'écran à l'exception des mots commençant par le caractère % .

Example:

```
int i = 6;
float f = 2.3456;
printf("Rang : %d , valeur %10.5f", i, f);
```

affiche à l'écran

Rang : 6, valeur 2.34560

La fonction printf

Exemples de spécificateurs de format

%c : affiche un caractère unique

%d ou %i : un entier signé sous forme décimale

%f : affiche une valeur réelle avec un point décimal.

%e ou %E : affiche une valeur réelle avec un exposant.

%x ou %X : affiche un entier hexadécimal.

%u : affiche un entier en notation décimale non signée.

%s : affiche une chaîne de caractères (string).

%g ou %G : affiche une valeur réelle avec affichage de type e ou f selon la valeur.

La fonction printf

Largeur minimale de champs

%4d : 4 digits "au moins" réservés pour l'entier.

%.2f : précision de 2 rangs décimaux.

Les arguments de *printf* sont :

Des constantes.

Des variables.

Des expressions.

Des appels de fonctions.

```
#include <stdio.h>
#include <math.h> //pour sqrt ()
main ()
    {
        float i = 2.0, j = 3.0
        printf ("%f %f %f %f", i, j, i+j, sqrt(i+j));
    }
```

La fonction scanf

La fonction scanf permet de saisir des données au clavier et de les stocker aux adresses spécifiées par les arguments de la fonction. C' est une fonction définie dans stdio.h

Syntaxe

scanf("*chaîne de controle*",*argument-1*,...,*argument-n*)

La chaîne de contrôle indique le format dans lequel les données lues sont converties.

Après la chaîne de format, scanf n'accepte que des adresses.

La fonction scanf

Exemple :

```
main()
{
        int  a, b;
        float quotient;
        printf("Entrez 2 nombres:");
        scanf("%d %d", &a, &b);
        quotient = a / b;
        printf("Le quotient est %f \n",
        quotient);
}
```

L'opérateur d'adresse "&" passe les adresses de a et de b à scanf. On écrit à l'écran deux nombres séparés par un espace blanc.
Si virgule entre les spécificateurs : scanf ("%d ,%d", &a, &b); => on écrit deux nombres séparés par une virgule.
Entre le pourcentage (%) et la lettre (d, f, s, x, e) on peut inclure d'autres spécifications:

Un premier programme C

On va rédiger le programme correspondant à l'algorithme qui lit trois nombres entiers, calcule et affiche leur somme, leur produit et leur moyenne.

La fonction scanf

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

 int main()
    {
        int  N1, N2, N3, S P;
        float  Moy ;
        printf("Entrez trois nombre : \n");
        scanf("%d%d%d", &N1,&N2,&N3);
        S=(N1+N2+N3) ;
        Moy=S/3;
        P= N1*N2*N3 ;
         printf("Somme:%d, moyenne:%f, produit:%d:\n",S, Moy,P);
        return 0;
    }
```

Les structures de contrôle en C

Les structures de contrôle en C

Voici les points que nous allons aborder:

L'intérêt des structures de contrôle

1. L'alternative : « if … else … »
2. Le choix multiple « switch … case …» et le break
3. La répétition

3.1 La boucle « for »

3.2 La boucle « while »

3.3 La boucle « do … while »

1. L'alternative en C

En langage C, l’alternative se traduit par l’instruction suivante:

```
If (expression)
        instruction1
else
        instruction2
```

1. L'alternative en C

Remarques :

S’il y a plus d’une instruction à exécuter, il faut les encadrer par des accolades

```
if(<condition >)
   {
      <instructions1>
   }
else
   {
      <instructions2 >
   }
```

1. L'alternative en C

***Exercice d'application*:**

Une société paie ses employés chaque semaine. Le salaire étant calculé sur la base d'un taux horaire, écrire un programme qui calcule et affiche le salaire sachant qu'il y a une majoration de 20% si le nombre d'heures est supérieur à 35.

1. L'alternative en C

```
#include <stdio.h>
#include <math.h>
main()
{     int nbre_heures, taux_horaire ;
      float salaire_hebdo  ;
      printf ("donnez le nombre d''heures : ") ;
      scanf ("%d" , &nbre_heures) ;
      printf("donnez le taux horaire : ") ;
      scanf ("%d" , &taux_horaire) ;
      if (nbre_heures <= 35)  /* pas de majoration */
            salaire_hebdo = nbre_heures*taux_horaire ;
      else/* majoration de 20%, i.e. 0,2 du salaire hebdomadaire*/
       salaire_hebdo = nbre_heures*taux_horaire * 1.2 ;

       printf("Le salaire hebdomadair est:%.2f\n",salaire_hebdo);

}
```

2- Le choix multiple en C

- La traduction du choix multiple en C est assez restrictive, puisque la valeur de l'expression conditionnant le choix doit être entière (char, short, int).

- L'instruction switch permet de mettre en place une structure d'exécution qui permet des choix multiples parmi des cas de même type et faisant intervenir uniquement des **valeurs constantes entières**.

2 - Le choix multiple en C

En **C** cette la structure de choix est codée par l'instruction **Switch**

Le **switch** en C s'écrit, avec la syntaxe suivante :

```
switch(<nomvariable >)
{
        case <valeur 1> : <instructions1 > ; break ;
        case <valeur 2> : <instructions2 > ; break ;
        case <valeur n> : < instructions n > ; break ;
        default : /* instructions */
}
```

2 - Le choix multiple

- Supposons que l'on veuille demander à l'utilisateur de choisir dans un menu une des 3 possibilités offertes.
- Le choix présenté ne se limite pas à une alternative (soit - soit).
- Mais plutôt à une expression du type « selon que… »

2 - Le choix multiple en C

```
#include <stdio.h>
int i;
main()
{
      printf("Entrez votre choix : ");
      scanf("%d",&i);
      switch(i)
       {
            case 1: printf("premier choix\n");   break;
            case 2: printf("deuxième choix\n");  break;
            case 3: printf("troisième choix\n"); break;
            default:printf("Autre choix que choix 1, 2 ou 3\n");
       }
}
```

2 - Le choix multiple en C

Maintenant, reprenons l'exemple du cours précédent en C:

N'oubliez surtout pas les break ! Si par exemple, nous voulons afficher littéralement un chiffre saisi au clavier, on écrit :

```
switch ( chiffre )
    {
      case 1 : printf ( " un " ) ; break ;
      case 2 : printf ( " deux " ) ; break ;
      case 3 : printf ( " trois" ) ; break ;
      case 4 : printf ( " quatre " ) ; break ;
      case 5 : printf ( " cinq" ) ; break ;
      case 6 : printf ( " six " ) ; break ;
      case 7 : printf ( " sept " ) ; break ;
      case 8 : printf ( " huit" ) ; break ;
      case 9 : printf ( " neuf " ) ; break ;
      case 0 : printf ( " zéro " ) ; break ;
      default : printf ( " ce n''est pas un chiffre !");
  }
```

3. La répétition

3.1 La boucle avec compteur

En C, la boucle avec compteur « pour » se traduit par « for »
Déclaration:

```
for(initialisation; condition; modification)
                    <instruction>;
Ou

for (initialisation; condition; modification)
  {
          <instructions;>
}
```

3.1 La boucle avec compteur

- **initialisation**:
  - est une instruction d'initialisation ; elle est exécutée avant l’entrée dans la boucle
- **condition**:
  - C’est la condition de continuation ; elle est testée à chaque passage, y compris lors du premier ; l’instruction ou les instructions composant le corps du for sont répétées tant que le résultat de l’expression **condition** est VRAI
- **modification** :
  - C’est une instruction de rebouclage ; elle fait avancer la boucle ; elle est exécutée en fin de boucle avant le nouveau test de passage.

3.1 La boucle avec compteur

Exemple :

Voici un algorithme qui affiche les entiers compris entre 1 et 100.

```
int i ;
main()
{
      for (i =1; i<=100 ;i++)
      printf ("%d " , i ) ;
}
```

3.1 La boucle avec compteur

Exercice d'application:

Ecrire un programme qui calcule la somme des n premiers entiers, n donné.

3.1 La boucle avec compteur

```
#include <stdio.h>
#include <math.h>
int i, n, s;
main()
{
     printf ( " Saisissez un entier : " ) ;
     scanf("%d", &n);
     s = 0;
     for (i=1; i<= n; i++)
       {
           s = s + i;
       }
printf(" La somme est : %d", s) ;
}
```

3.2 La Boucle While

La boucle à répétition « **TANT QUE ….FAIRE** » se traduit en C par «while »

Déclaration :

```
while (< condition > )
    {
        < instructions >
    }
```

3.2 La Boucle While

```
#include<stdio.h>
 main ( )
   {
      int i = 1 ;
      while ( i <= 5)
          {
             printf( "%d " ,
    i ) ;
             i++;
          }
      printf ("\n" ) ;
  }
```

3.2 La Boucle While

Exercice d'application

Ecrire un programme permettant la saisie de la réponse à la question : « Aimez vous l'algorithmique (o/n) ? ». Le programme doit afficher un message en cas de mauvaise réponse, et renouveler la question jusqu'à ce que la réponse soit correcte.

3.2 La Boucle While

```
#include<stdio.h>
int main()
{
    char rep ;
    printf("Aimez vous l'algorithmique (o/n) ? \n") ;
    scanf("%c", &rep) ;
    while(rep != 'o'  && rep != 'n')
    { printf("Veuillez repondre par oui(o) ou non(n) svp \n");
      printf( "\n" ) ;
      printf("Aimez vous l'algorithmique (o/n) ? \n");
      scanf("%c",&rep) ;
      printf( "\n" ) ;
    }
 }
```

3.3 La boucle « do … while »

Le langage C propose également une autre forme de la boucle « tant que » qui permet d'exécuter au moins une fois le corps de la boucle qui peut être l'équivalent de la boucle « répéter » .

Déclaration:

```
dO
   <instruction;>
while ( <Condition>);

Ou

dO {
        <instructions;>
   }
while ( <Condition>);
```

3.3 La boucle « do … while »

Exercice d'application:

Ecrire un programme qui permet de saisir une valeur entière positive et de calculer sa racine carrée.

3.3 La boucle « do … while »

```
#include<stdio.h>
#include<math.h>
  main()
    {
              int entier_positif;
              float Racine;
               do
                 {
                     printf("donner un entier positif :");

                     scanf("%d", &entier_positif);
                 }
              while(entier_positif<0);

           Racine = sqrt(entier_positif) ;
        //{sqrt est une fonction prédéfinie en C, elle calcule la racine carrée d'un entier positif}
         printf("la racine carrée de %d est:%f",entier_positif,
                          Racine);
         printf( "\n" ) ;
    }
```

Chapitre 3 : Les tableaux

Notion de tableau

Soit un entier n positif.

Supposons qu'on veuille saisir n valeurs réelles afin de calculer leur moyenne, ou de trouver leur minimum, ou de les afficher par ordre croissant.

Pour des valeurs petites de n, on peut déclarer n variables réelles pour résoudre le problème.

Mais si n est assez grand, on se rend compte que cela devient impropre, fastidieux, voire impossible.

Il faudrait, dans ce cas, utiliser une variable permettant de représenter les n valeurs. Le type de données de cette variable serait le type tableau.

Notion de tableau

Exemple

Saisir la liste des 12 notes sur 30

16 23 8 19 28 20 18 14 10 9 15 24

Voici la liste de ces notes sur 20

10.67 15.33 5.33 12.67 18.67 13.33 12 9.33 6.67 6 10 16

- Problème :
  - Déclaration de 12 variables différentes.
  - Tâche fastidieuse avec 30 ou 40 notes
  - Variables possédant des noms différents
    - Impossible d'utiliser une boucle.
    - Code devient très répétitif et redondant
- Solution :
  - Utiliser une SD avec un nom commun pour toutes les variables et de les repérer par un numéro.
  - Les tableaux

Notion de tableau

Définition :

Un tableau est une collection séquentielle d'éléments de même type, où chaque élément peut être identifié par sa position dans la collection. Cette position est appelée indice et doit être de type scalaire.

Notion de tableau

Déclaration :

Pour déclarer un tableau, il faut donner :

- son nom (identificateur de la variable)
- sa taille : le nombre d'élément du tableau.
- le type des éléments le composant.

Syntaxe :

variable Tab : tableau [taille] d'entiers
ou
Variable nom : tableau[<taille >] de <type des composants>

Exemple :

variable tab : tableau[10] de
réels

Notion de tableau

En C, la déclaration se met sous la forme :

< type > < identificateur >[<taille >]

<type> <nomdutableau>[<taille >] ;

Exemple :

```
int tab [10];  // déclaration d'un tableau de 10 entiers

float tab [10];  // déclaration d'un tableau de 10 réels
```

Notion de tableau

Schématiquement, on va représenter la variable tab comme suit:

0	1	2	3	4	5	6	7	8	9
8.4	3.5	12	20	10	13.34	50	100	30.1	60.9

Dans la mémoire centrale, les éléments d'un tableau sont stockés de façon linéaire, dans des zones contiguës.

Le tableau ci-dessus est de dimension 1, nous verrons un peu plus loin que l'on peut représenter des tableaux à 2 dimensions, voire même plus. L'élément n° 1 sera représenté par l'expression tab [1].

Dans notre exemple, tab[1] peut être traité comme une variable réelle. On dit que le tableau t est de taille 10.

Les éléments d'un tableau à n éléments sont indicés de 0 à n -1.

Création d'un tableau

La création d'un tableau consiste au remplissage des cases le composant. Cela peut se faire par saisie, ou par affectation.

Par exemple, pour remplir le tableau t précédent, on peut faire :

t[1] = 8.4 ; **t[2] = 3.5** ; … **tab[10] = 60.9 ;**

Si on devait saisir les valeurs, il faudrait écrire :

```
Pour i ← 0 à 9  faire
      lire(t[i] );
```

```
for( i = 0 ; i<10 ; i++)
      scanf("%d" , &t[i] );
```

Affichage d’un tableau

Afficher un tableau revient à afficher les différents éléments qui le composent. Pour cela, on le parcourt (généralement à l’aide d’une boucle avec compteur) et on affiche les éléments un à un.

Exemple :

```
for( i = 0 ; i<10 ; i++)
      printf("%d" , t[i] );
```

Exercice d’application

Ecrire un programme qui permet de créer un tableau d’entiers t1 de taille 20 par saisie, et un tableau t2 de même taille en mettant dans t2[i] le double de t1[i], i appartenant à {0, .., 19 }.

Exercice d'application

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

main()
          //{ création de t1 }
 {
     int Tab[10],Tab2[10],i;
           for(i=0;i<10;i++)
           {
                printf("donnée Tab[%d] :\n",i);
                scanf("%d",&Tab[i]);
           }
          // création de t2
         for(i=0;i<10;i++)
            {
                Tab2[i]=2*Tab[i];
                printf("Tab2[%d] = %d\n",i,Tab2[i]);
             }
   }
```

Traitement d’un tableau

Après avoir créé un tableau, on peut y effectuer plusieurs opérations comme le calcul de la somme ou de la moyenne des éléments, la recherche du plus petit ou du plus grand élément du tableau, le test d’appartenance d’un objet au tableau, …

Pour la suite, on considère un tableau d’entiers t déclaré comme suit :

Int tab[n]

Somme des éléments d'un tableau

On effectue la somme des éléments du tableau tab, le résultat est dans la variable S :

```
S=0 ;
for(i=0;i<10;i++)
    S = S + tab[i];
    printf("la somme des elements de tab est %d: ", S);
```

Minimum d’un tableau

On cherche le plus petit élément du tableau tab, le résultat est dans la variable min :

```
min = tab[0] ;

for(i=0;i<10;i++)
      if (tab[i] < min)
         min = tab[i];
        printf(« Le minimum des éléments de
tab est: \n», min) ;
```

Test d’appartenance

On cherche si l’entier x appartient à tab, le résultat est mis dans la variable booléenne appartient :

```
appartient = 0;

for (i=0; i < n; i++)
   if (tab[i]==x)
   appartient = 1;
   if (appartient==1)
      Printf("x appartient à tab")
  else
      printf("x n’’appartient pas à tab");
```

Test d'appartenance

On remarque que l'on peut arrêter les itérations (la recherche) si l'on rencontre l'élément x dans tab. Pour cela, il faut utiliser une boucle *while* ou *do while*:

```
i = 0;
while ((tab[i]!=x) && (i<n))
    i=i+1;
    if (i==n)
        printf("x n''appartient pas à tab");
    else
        printf("x appartient à tab");
```

```
i = 0 ;
do
    i=i+1;
while (tab[i]!=x) && (i<n));
    if (i>n)
        printf("x n''appartient pas à tab");
    else
        printf("x appartient à tab");
```

Dans les deux cas, si x appartient à tab, la valeur de i est l'indice de la case qui le contient.

Ajout d'un élément

On suppose que le tableau tab est « rempli » et qu'il reste une case non occupée à la fin (la $n^{ième}$ case).

Si on veut alors ajouter un entier x à la fin du tableau, il suffira d'écrire

tab[n] = x ;

Mais, si on veut ajouter x dans une case dont l'indice k est différent de n, il faudra décaler les éléments tab[k], tab[k+1], …, tab[n-1] vers la droite pour libérer la case d'indice k :

```
for( i = n ;i>=k; i--)
    tab[i] = tab[i-1];
tab[k] = x;
```

Suppression d’un élément

Pour supprimer l’élément se trouvant à la position k de tab, on l’écrase en faisant décaler les éléments placés après lui vers la gauche :

```
for (i = k; i< n; i++)
   tab[i] = tab[i+1];
```

Il faut noter qu’après une telle opération, l’élément se trouvant à la dernière position n’est plus significatif.

Tableaux de caractères

Une chaîne de caractères est soit une chaîne vide, soit un caractère suivi d'une chaîne de caractères ; en un mot c'est une collection de caractères.

Exemples :

"Bonjour", "L'ESTM se situe à Castor", "A", "2017 "

Les chaînes de caractères sont stockées sous forme de tableaux de caractères. Le compilateur complète la chaîne avec un caractère NULL ('\0') qui est ajouté à la suite du dernier caractère utile constituant la chaîne de caractères. Il faut donc que le tableau ait au moins un élément de plus que le nombre de caractères de la chaîne littérale.

- Par exemple, la phrase "Toto" sera codée de la sorte :

'T'	'o'	't'	'o'	0	…

Tableaux de caractères

Déclaration d'une chaîne de caractères

Une chaîne de caractères peut donc être déclarée de la manière suivante :

```
char ch[10]; // chaîne de 9 caractères + caractère NULL
```

Exemple :

```
char c [ 5 0 ] = "Toto" ;
```

Cette instruction déclare une chaine de caractères c initialisée à "Toto". Les 5 premiers éléments du tableau seront occupés par les 4 caractères de la chaîne ainsi que par le caractère null, les autres contiendront des valeurs non significatives.

Tableaux de caractères

Déclaration d'une chaîne de caractères

Observez bien l'exemple suivant :

```
char c [ 4 ] = "Toto" ;
```

Cette déclaration engendrera un warning à la compilation et probablement une erreur à l'exécution car l'affectation du caractère nuln à la 5-eme position du tableau donnera lieu à un débordement d'indice.

Tableaux de caractères

Accès aux éléments

Le code suivant permet d'afficher une chaîne caractère par caractère :

```
int  i = 0 ;
while(c[i]!=0)
     printf("%c" , c[i++]);
```

Notez que le corps de la boucle while est itéré jusqu'à ce que le caractère null soit rencontré. Il est donc impératif que votre chaîne se termine par le caractère nul et que le caractère nul se trouve dans la plage d'indices du tableau .

Tableaux de caractères

Initialisation à la saisie au clavier

On peut utiliser la fonction scanf et le format %s mais on utilisera de préférence la fonction gets non formatée.

```
char texte[10];
      printf("Entrer un texte : ");
      scanf("%s", texte);
      est équivalent à gets(texte);
```

Une chaîne étant un pointeur, on n'écrit pas le symbole &.

Remarque: scanf ne permet pas la saisie d'une chaîne comportant des espaces. Les caractères saisis à partir de l'espace ne sont pas pris en compte (l'espace est un délimiteur au même titre que (line feed) LF('\n')) mais ils sont rangés dans le buffer d'entrée. Pour saisir une chaîne avec des espaces, il faut utiliser l'instruction gets. A l'issue de la saisie d'une chaîne de caractères, le caractère de retour chariot est remplacé par le caractère de fin de chaîne '\0'.

La bibliothèque string.h

Cette bibliothèque propose des fonctions de maniement de chaînes de caractères, à savoir :

- strcmp : comparer deux chaînes.
- strlen : longueur d'une chaîne de caractère
- strsubs : rechercher une sous-chaîne
- strcat : concaténer deux chaînes
- strcpy : copier une chaîne

Il vous est conseillé d'examiner et de comprendre comment fonctionnent ces fonctions

La bibliothèque string.h

strlen(<s>)	fournit la longueur de la chaîne ***sans*** compter le '\0' final	
strcpy(<s>, <t>)	copie <t> vers <s>	
strcat(<s>, <t>)	ajoute <t> à la fin de <s>	
strcmp(<s>, <t>)	compare <s> et <t> lexicographiquement et fournit un résultat:	
	négatif	si <s> précède <t>
	zéro	si <s> est égal à <t>
	positif	si <s> suit <t>
strncpy(<s>, <t>, <n>)	copie au plus <n> caractères de <t> vers <s>	
strncat(<s>, <t>, <n>)	ajoute au plus <n> caractères de <t> à la fin de <s>	

Les tableaux à deux dimensions

Pour traiter les notes obtenues par un étudiant à 10 épreuves on peut utiliser un tableau de 10 réels. Pour traiter les notes obtenues par 5 étudiants, on pourrait utiliser 5 tableaux de 10 réels chacun.

Mais puisqu'on va effectuer très probablement les mêmes traitements sur ces tableaux, il est préférable de les regrouper dans une seule variable qui sera un tableau de 5 lignes et 10 colonnes.

Chaque élément de ce tableau multidimensionnel sera identifié par deux indices (i,j) : la position i indiquant la ligne et la position j indiquant la colonne.

Les tableaux à deux dimensions

Déclaration :

La déclaration se met sous la forme :
< type > < identificateur > < taille1 > < taille2 >

On accède à un élément du tableau en donnant un indice par dimension (entre crochets).

Exemple :

```
#define LIGNE  2 // matrice de 2 lignes et 3 colonnes
#define COLONNE  3

int i, j; // i est l'indice de ligne et j est l'indice de colonne
int tab[LIGNE] [COLONNE]; // déclaration d'un tableau de 6 entiers
```

Les tableaux à deux dimensions

Exemple :

// affichage des 6 éléments du tableau

```
for (i = 0; i < ligne; i++)
        for (j = 0; j < colonne; j++)
          printf("%d\n", tab[i][j]);
```

Les tableaux à deux dimensions

Exercice d’application

Ecrire un programme qui permet de créer (par saisie) et d’afficher un tableau tab à deux dimensions d’entiers de taille 3x5.

Les tableaux à deux dimensions

```
#include <stdio.h>

 main()  // creation de t1
    {
      int Tab[3][5],i,j;

      //saisie du tableau
      for(i=0;i<3;i++)
            for(j=0;j<5;j++)
                {
                     printf("donner Tab[%d][%d]",i,j);
                     scanf("%d",&Tab[i][j]);
                 }
           //(affichage de t )
           for(i=0;i<3;i++)
                  {
                     for(j=0;j<5;j++)
                             printf("%d",Tab[i][j]);
                             printf("\n");
                   }
 }
```

Chapitre 4: LES POINTEURS