UNIVERSITÉ CHEIKH ANTA DIOP DE DAKAR

ECOLE SUPÉRIEURE POLYTECHNIQUE

DÉPARTEMENT GÉNIE INFORMATIQUE

PROGRAMMATION EN C

COURS INTRODUCTIF

Formateur

Dr Mouhamed DIOP

mouhamed.diop@esp.sn

OBJECTIFS

- A l'issue du cours, l'étudiant devra :
  - Connaitre la syntaxe de base du langage C
  - Pouvoir écrire des programmes en C
  - Pouvoir traduire un algorithme en C
  - Pouvoir compiler, exécuter, débugger un programme écrit en C

CONTENU

- Syntaxe de base
- Types simples, variables et opérateurs
- Les structures de contrôle
- Les tableaux et les chaines de caractères
- Les structures, unions et énumérations
- Fonctions et procédures
- Les pointeurs
- Les fichiers
- La programmation modulaire

Le langage C

- Le Langage C peut être vu comme un langage de bas niveau
  - Il donne accès à des données telles que des bits, des adresses, etc.
  - Il permet donc l'écriture de systèmes d'exploitation et de logiciels
- Il peut également être vu comme un langage de haut niveau
  - Il dispose de structures de base nécessaires à la conception d'applications structurées
- Il est l'un des premiers langages offrant des possibilités de programmation modulaire

Un peu d'histoire

- Inventé en 1972 par Dennis Ritchie et Ken Thompson (AT&T Bell Laboratories)
  - Pour réécrire Unix et développer des programmes dessus
- Le livre de référence qui le décrit a été publié en 1978 par Brian Kernighan et Dennis Ritchie
  - Il s'intitule « The C programming language »
- Le succès de Unix (écrit en C) facilitera son adoption
  - Plusieurs compilateurs C sont ainsi créés et largement disponibles dans les distributions Linux
- Il a été normalisé en 1989 par l'ANSI, puis standardisé ISO depuis 1990.

Structure d'un programme en C

- Le format du code source est libre et la mise en page n'a aucune signification pour le compilateur
  - Elle permet toutefois une bonne lisibilité du programme.
- Certaines règles constituent toutefois sa syntaxe de base

  liste des déclarations de variables

  liste des déclarations de fonctions et de procédures
- Les déclarations des variables peuvent être omises
  - Mais il est nécessaire de donner au moins la définition d'une fonction particulière qui est main()

Structure d'un programme en C

```c
// inclusion de source
#include <stdio.h>
// déclaration de variables
int a, b;
float x, y;
// déclaration d'une fonction
int carre(int nombre)
{
    ...
}
// déclaration de la fonction principale
int main()
{
    ...
}
```

Les commentaires

- Ils détaillent le programme et contribuent à sa lisibilité ainsi qu'à sa compréhension
- Les lignes de code commentées ne sont pas pris en compte par le compilateur
  - Ils débutent par un // (commentaire sur une seule ligne) ou sont délimités par /* et */ (commentaire sur plusieurs lignes)
  - Exemple:
    - // Ceci est un commentaire sur une ligne
    - /* Ceci est un commentaire qui peut s'étendre sur plusieurs lignes */
- Les commentaires ne peuvent être imbriqués

Les inclusions de source

- L’inclusion de source permet d’inclure du code déjà implémenté ou d’autres bibliothèques dans un programme
- La syntaxe d’inclusion est la suivante :

  #include <nom du fichier> ou #include "nom du fichier"
- La commande d’inclusion doit être placée au début du programme
  - Cette ligne sera remplacée par le pré-processeur qui mettra à cette place le contenu du fichier correspondant
- Exemples:
  - #include <stdio.h>
  - #include "fonctions.h"

Les inclusions de source

```c
// inclusion de source
#include <stdio.h>
int main()
{
    printf("Mon premier programme en C");
}
```

Types de base et opérateurs

Types et variables

- Le C est un langage fortement typé
  - Chaque donnée en C (constantes, variables) a un type
- Le choix du type de donnée est trés important
  - Définit l'espace mémoire à réserver ainsi que le format de représentation de la donnée
- Nous allons d'abord voir les types de données primitifs
  - Les types complexes (tableaux, structures, types personnalisés, pointeurs,…) seront abordés plus tard

Les types de base : les entiers

- Le char ou unsigned char (1 octet)
  - Permet de représenter un caractère
  - Contient le code ASCII du caractère
- L'ajout du mot clé « unsigned » permet de ne prendre en compte que les entiers positifs
  - Un char stocke un nombre entre -128 et 127 alors qu'un unsigned char se trouve entre 0 et 255
- Le type wchar_t existe dans le fichier <stddef.h> pour gèrer des alphabets de plus de 255 caractères

Les types de base : les entiers

- Le type short ou unsigned short (2 octets):
  - Stocke des entiers : -32768 à 32767
  - unsigned short: 0 à 65535
- Le type long ou unsigned long (4 octets):
  - Entiers de -2.147.843.648 à 2.147.843.647
  - unsigned long: 0 à 4.294.967.295
- Le type int ou unsigned int :
  - C’est le type entier et peut correspondre au long ou au short.
  - Généralement, il se confond au int et est donc représenté sur 4 octets.

Les types de base : les rééls

- Le type float (4 octets):
  - 23 bits pour la mantisse, 8 bits pour l'exposant, 1 bit pour le signe
  - Permet de représenter des réels : 3.4 *10 exp(-38) à 3.4*10 exp(38)
- Le type double (8 octets):
  - 52 bits pour la mantisse, 11 bits pour l'exposant, 1 bit pour le signe.
  - Permet de représenter des réels : 1.7 *10 exp(-308) à 1.7*10 exp(308)
- Le type récent long double est représenté sur 10 octets.

Les types de base : constantes de type entier

- Elles peuvent être entrées sous forme décimale, octale ou hexadécimale.
  - Les constantes décimales s'écrivent de la manière usuelle (ex : 455)
  - Les constantes octales commencent par un 0 (ex : 0577)
  - Les constantes hexadécimales par 0x ou 0X (ex : 0x9a2f)
- 455u, 0577U, 0x9a2fu pour dire qu'ils sont non signés (unsigned)
- 455L, 0577l, 0x9a2fL pour spécifier qu'ils sont de type long

Les types de base : constantes de type réel

- En C
  - 2.5 et non 2,5
  - 2. correspond à 2.0
  - .3 correspond à 0.3
  - 5 .4 correspond à 5.4
  - 2e7 ou 2.e7 correspond à $2x10^7$

Les types de base : constantes de type caractères

- Une constante caractère s'écrit entre apostrophes (exemple 'j').
- Attention à 1 et '1'
  - Ils ne sont pas de même type (int vs char)
- Certains caractères peuvent avoir des fonctionnalités en C
  - Il faut ajouter le signe \ pour spécifier cette fonctionnalité
    - \n : retour à la ligne
    - \t tabulation
- Pour écrire le caractère \ donc nous mettrons '\\'.

Les types de base : chaine de caractères

- Les constantes de type chaine de caractères
  - Une constante chaine de caractères s'écrit entre guillemets
  - Exemple : "Ceci est une chaîne"
  - Différence entre 'd' et "d" ?

Les variables

- Une variable est une donnée du programme qui peut être modifiée
  - Une variable est définie par un nom et par un type
  - Elle est chargée en mémoire centrale (RAM)
  - Le nom d'une variable est un identificateur et doit respecter les règles s'y appliquant.
  - Il faut nécessairement définir une variable avant de l'utiliser
  - Plusieurs variables peuvent être définies en même temps
- Exemples:
    - int age;
    - float notes, poids, taille;

Les constantes

- Une constante est une donnée dont la valeur ne change pas durant toute l'exécution d'un programme
  - Le nom d'une constante est un identificateur et est généralement écrit en majuscule
- Définition d'une constante

  #define nom_constante valeur_constante ou bien

  const type_constante nom_constante = valeur_constante
  - Exemples:

    #define PI 3.14 // pas de ; à la fin

    const float TAUX_CONVERSION = 655.957 ;
  - Différence entre les deux formes?

Opérations sur les variables

- **Initialisation:**
  - Comme son nom l'indique, il s'agit ici de donner dés la définition une valeur à la variable
    - int i = 1;
- **Affectation:**
  - Semblable à l'initialisation, elle permet d'assigner une valeur à une variable à n'importe quelle étape du programme.
- Lors de l'affectation et de l'initialisation, il faut attribuer à la variable une valeur de même type.
- Des opérations comme **l'impression et la saisie formatée** sont également possibles sur une variable (cf slides suivants)

L'impression formatée

- Sortie formatée :
  - Contrôler la forme et le format de l'affichage des données sur la sortie standard
- Utiliser la fonction printf().
  - Syntaxe
    - printf("format", argument_1, argument_2, ..., argument_n);
- La chaîne qui donne le format est composée de % et d'une lettre qui donne le type de l'argument à afficher
- On peut afficher plusieurs arguments en même temps,
  - Il suffit de spécifier le format autant de fois

L'impression formatée

- **Affichage d'entiers décimaux**
  - **Pour afficher un nombre entier, il faut utiliser la lettre d**

```
...
int age = 8;
printf("Vous avez %d ans", age);
int min, max;
min = 5;
max = 10;
printf("\nL'intervalle est : [%d, %d]", min, max);
```

L'impression formatée

- Contrairement au décimal, l'affichage d'un octal ou d'un hexadécimal considère les données comme non signées
- **Affichage d'entiers octaux :**
  - Prendre la lettre o
- **Affichage d'entiers hexadécimaux :**
  - Prendre la lettre x ou X
- Exemple

printf("Le nombre 10 s'écrit en octale %o et en hexadécimal %x ou %X", 10,10,10);

L'impression formatée

- **Affichage d'un caractère**
  - **Pour afficher un caractère, il faut utiliser la lettre c**

```
char a = 'a';
printf(" %c ", a);
printf("Bonjour %c tout le monde\n", 'a');
char b = 'b';
printf("Le caractère b est suivi de %c", b+1);
```

L'impression formatée

- **Exercice**
  - **Donner le code ASCII en décimal, octal, hexadécimal des 26 caractères majuscules de l'alphabet français**
  - **Faites le même exercice en affichant cette fois-ci les informations sous forme de tableau**

L'impression formatée

- **Affichage de réels**
  - Les lettres f, e et g pour les types float et double et %Lf pour le type long double
    - La lettre f est la plus utilisée
  - Exemple :

```
float r=1.2;
printf("%f", r);
```

L'impression formatée

- **Affichage de réels**
  - La fonction affichera 6 chiffres après la virgule
    - Il est possible d'ajouter un nombre entre % et la lettre pour donner la place que va prendre l'affichage (gabarit)
    - Il est également possible de spécifier le nombre de chiffres après la virgule en précédant la chaine de format d'un point suivi du nombre de chiffres souhaité

```
float moyenne = 17.5;
// affichage de 6 chiffres après la virgule
printf("Vous avez une moyenne de %f\n", moyenne);
// affichage de 2 chiffres après la virgule
printf("Vous avez une moyenne de %.2f\n", moyenne);
```

L'impression formatée

- Affichage de chaînes de caractères
  - Se fait en utilisant la lettre s
    - Exemple
    - printf("Vous vous appelez %s", "Fatou");

L'impression formatée

- **Tableau des formats pour printf**

Symboles	Types décrits
%d	Entier (entier décimal)
%o	Entier (entier en octal)
%x ou %X	Entier (entier hexadécimal)
%u	Entier (entier non signé)
%f	Réel
%e ou %E	Réel (format exponentiel)
%g ou %G	Réel (soit e ou f)
%c	Caractère
%s	Chaîne de caractères

La saisie formatée

- Comme pour l'impression, il s'agit de contrôler le type de données que l'on veut saisir.
  - Syntaxe: scanf("format", argument_1, argument_2,..., argument_3);
- Même syntaxe que printf, mais diffère sur un point
  - argument_1 représente l'adresse de l'argument correspondant
- Exemple
  - Pour enregistrer, de l'entrée standard (clavier), la valeur de i :
    - scanf("%d ",&i)

L'impression formatée

- **Tableau des formats pour scanf**

%d	Entier décimal (int)	%u	Décimal (non signé)
%hd	Entier décimal (short)	%hu	Décimal (short non signé)
%ld	Entier décimal (long)	%lu	Décimal (long non signé)
%o	Entier octal (int)	%f	Réel (float)
%ho	Octal (short)	%lf ou %Lf	Réel (double ou long double)
%lo	Octal (long)	%e ou %E	Réel (float en exponentiel)
%x	Entier hexadécimal (int)	%le ou %LE	Réel (double ou long double)
%hx	Hexadécimal (short)	%g ou %G	Réel(float)
%lx	Hexadécimal (long)	%lg ou %LG	Réel (double ou long double)
%c	Caractère	%s	Chaîne de caractères

Les fonctions putchar et getchar

- `int getchar()` : lecture d'un caractère. Renvoie le code du caractère lu, `EOF` si problème. Équivalent à `scanf("%c",&c_lu)`
- `int putchar(int c)` : écriture d'un caractère. Renvoie le code du caractère écrit, `EOF` si problème. Équivalent à `printf("%c",c_a_ecrire)`
- Fonctions plus simples et plus rapides

Les opérateurs

- Trois types d'opérateurs
  - Opérateurs unaires
  - Opérateurs binaires
  - Opérateurs ternaires

Les opérateurs

- **Opérateurs unaires**
  - Opérateur moins unaire : - inverse le signe
  - Opérateur plus unaire : + ne fait rien.
  - Opérateur d'incrémentation : ++
    - i++ (post-incrémentation) ⟷ ++i (pré-incrémentation) ⟷ i = i+1
    - j = ++i ➔ j et i prennent la valeur de i+1
    - j = i++ ➔ j prend la valeur de i (avant incrémentation) et i vaut i+1
  - Opérateur de décrémentation : --
    - Fonctionne de la même manière que l'incrémentation

Les opérateurs

- **Opérateurs unaires (suite)**
  - Opérateur de taille : sizeof(...)
    - Exemple : int i; sizeof(i)= ?
  - Non logique : !
    - Exemple : si EXP est vraie alors !EXP est fausse et inversement
  - Complément à un : ~ (tilde) s'applique au bits
  - Opérateur d'adresse : &
  - Opérateur d'indirection d'adresse : *
  - Opérateur de conversion ou cast : (type) EXP

Les opérateurs

- **Opérateurs binaires**
  - Opérateurs arithmétiques
    - addition, soustraction, multiplication, division : + , -, *, /
    - reste de la division : %
  - Opérateurs de comparaison
    - inférieur, inférieur ou égal, supérieur, supérieur ou égal : <, <=, >, >=
    - égal, différent : ==, !=
  - Opérateurs Logiques
    - Niveau bit: et, ou, ou exclusif : &, | , ^
    - Niveau expression : et, ou : &&, ||

Les opérateurs

- **Opérateurs binaires (suite)**
  - Opérateurs de décalage
    - Décalage vers la droite: >>
    - Décalage vers la gauche: <<
  - Opérateur de succession: ,
    - Exemple : i= (j=7, k=2);
  - Opérateur d'affectation: =
    - peut être précédé de +, -, *, /, &, |, ^, <<, >>
    - Exemple : age += 5 équivaut à age = age + 5

Les opérateurs

- **Opérateur ternaire**
  - Expression1 ? Expression2 : Expression3
  - Exemple

    int autorisation = -1, age = 25;

    autorisation = age < 18 ? 0 : 1;
    - autorisation prendra 0 ou 1 suivant que l'age soit inférieur ou non à 18
    - Dans l'exemple, autorisation va prendre 1

Notion d'expression

- Une Expression est une suite syntaxiquement correcte d'opérateurs et d'opérandes
  - Exemples: e+f; a&b; sizeof(int); …
- Une expression renvoie toujours une valeur
- Une instruction est une expression suivie de « ; » ou une instruction de contrôle
- Une instruction peut aussi être une suite d'instructions délimitées par les signes «{» et «}»

Les structures de contrôle

Les structures de contrôle

- Il s'agit principalement des :
  - Structures conditionnelles
    - if
    - if...else
    - switch
  - Structures itératives
    - for, while, do...while
  - Instructions de branchement
    - break, continue, goto

Les structures conditionnelles

- Permet l'exécution d'instructions suivant qu'une (des) condition(s) soi(en)t vérifiée(s) ou non

```
if (condition)
{
    instructions_à_exécuter_si_la_condition_est_vérifiée
}
```

  - Les accolades peuvent être omises si une seule instruction est présente dans la structure conditionnelle
  - Plusieurs conditions peuvent également être définies
    - Il faudra les lier avec des opérateurs logiques (et, ou, ...)

Les structures conditionnelles

- Il est possible d'exécuter du code alternatif si la condition n'est pas vérifiée

```
if (condition)
{
    instructions_à_exécuter_si_la_condition_est_vérifiée
}
else
{
    instructions_à_exécuter_si_la_condition_n'est_pas_vérifiée
}
```

Les structures conditionnelles

- Exemple

```c
if (solde > montant)
{
    printf("La transaction est autorisée\n");
}

if (age < 18)
{
    printf("Vous êtes mineur\n");
}
else
{
    printf("Vous êtes majeur\n");
}
```

Les structures conditionnelles

- Il est possible d'enchainer plusieurs conditions
  - Le dernier else est optionnel

```
if (condition_1)
{
    instructions_à_exécuter_si_la_condition_1_est_vérifiée;
}
else if (condition_2)
{
    instructions_à_exécuter_si_la_condition_2_est_vérifiée;
}
...
else if (condition_n)
{
    instructions_à_exécuter_si_la_condition_n_est_vérifiée;
}
else
{
    instructions_à_exécuter_si_aucune_condition_n_est_vérifiée;
}
```

Les structures conditionnelles

- Exemple d'enchainement de conditions

```
if (code_operateur == 70)
{
    printf("Vous êtes un client de Expresso\n");
}
else if (code_operateur == 76)
{
    printf("Vous êtes un client de Free\n");
}
else if (code_operateur == 77 || code_operateur == 78)
{
    printf("Vous êtes un client de Orange\n");
}
else
{
    printf("Désolé, le code fourni nous est inconnu\n");
}
```

Les structures conditionnelles

- On peut utiliser switch pour remplacer plusieurs if...else if...else

```
switch (expression)
{
    case constante_1: instructions_si_expression_egale_constante_1; break;
    case constante_2: instructions_si_expression_egale_constante_2; break;
     ...
    case constante_n: instructions_si_expression_egale_constante_n; break;
    default : instructions_si_aucune_correspondance
}
```

Les structures conditionnelles

- Exemple d'utilisation de Switch

```
switch(code_operateur)
{
    case 70 :
        printf("Vous êtes un client de Expresso\n");
        break;
    case 76 :
        printf("Vous êtes un client de Free\n");
        break;
    case 77 :
        printf("Vous êtes un client de Orange\n");
        break;
    case 78 :
        printf("Vous êtes un client de Orange\n");
        break;
    default:
        printf("Le code fourni nous est inconnu\n");
}
```

Les structures conditionnelles

- Exemple d'utilisation de Switch (regroupement de cas)

```
switch(code_operateur)
{
    case 70:
        printf("Vous êtes un client de Expresso\n");
        break;
    case 76:
        printf("Vous êtes un client de Free\n");
        break;
    case 77: case 78:
        printf("Vous êtes un client de Orange\n");
        break;
    default:
        printf("Désolé, le code fourni nous est inconnu\n");
}
```

Les structures itératives

- La boucle while

```
while (condition)
{
    instructions_à_exécuter_à_chaque_itération
}
```

  - Les accolades peuvent être omises si une seule instruction est présente dans la structure itérative
  - La boucle peut ne pas s'exécuter si la condition n'est pas vérifiée au début
  - Plusieurs conditions peuvent également être définies
    - Il faudra les lier avec des opérateurs logiques (et, ou, ...)

Les structures itératives

- La boucle while : Exemple

```c
int i = 1, table = 2;
while(i <= 10)
{
    printf("%d x %d = %d\n", table, i, table * i);
    i++;
}
```

Les structures itératives

- La boucle do...while

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

  - Le point-virgule (;) à la fin est obligatoire
  - Les accolades peuvent être omises si une seule instruction est présente dans la structure itérative
  - La boucle est exécutée au moins une fois
  - Plusieurs conditions peuvent également être définies
    - Il faudra les lier avec des opérateurs logiques (et, ou, ...)

Les structures itératives

- La boucle do...while : Exemple

```c
int i = 1, table = 2;
do
{
    printf("%d x %d = %d\n", table, i, table * i);
    i++;
}while(i <= 10);
```

Les structures itératives

- La boucle for

```
for (initialisation; condition; post_iteration)
{
    instructions_à_exécuter_à_chaque_itération
}
```

  - L'initialisation s'exécute une seule fois avant qu'on ne rentre dans la boucle
  - La condition est testée chaque fois avant de rentrer dans la boucle
    - On ne rentre dans la boucle que si la condition est vérifiée
  - La partie post-itération est exécutée après chaque itération

Les structures itératives

- La boucle for (suite)
  - La présence des différentes parties de la boucle For n'est pas obligatoire
    - L'instruction `for( ; ; );` crée une boucle infinie
  - Les accolades peuvent être omises si une seule instruction est présente dans la structure itérative
  - Plusieurs conditions peuvent également être définies
    - Il faudra les lier avec des opérateurs logiques (et, ou, ...)

```
int i = 0, table = 2;
for (i = 1; i <= 10; i++)
{
    printf("%d x %d = %d\n", table, i, table * i);
}
```

Les instructions de branchement

- Break
  - Il s'utilise à l'intérieur d'une boucle ou d'un switch
  - Provoque l'arrêt du déroulement de la boucle (ou de l'instruction switch) en cours et donne le contrôle à l'instruction située après la boucle (ou l'instruction switch)
  - Cette instruction n'a d'intérêt que si son exécution est conditionnée par un choix

Les instructions de branchement

- Continue
  - Elle doit être utilisée à l'intérieur d'une boucle
  - Elle provoque l'exécution immédiate de la prochaîne itération de la boucle
    - Elle provoque donc l'arrêt de l'itération en cours
  - Les instructions situées entre continue et la fin de la boucle sont ignorées
  - Elle n'a d'intérêt que si son exécution est régie par une condition

Les instructions de branchement

- Exemple d'utilisation de break et continue

```c
#include <stdio.h>
int main()
{
    int i;
    for (i = 0; i < 20; ++i)
    {
        if (i % 2 == 0)
        {
            continue;
        }
        printf("Iteration N°%d\n", i);
        if (i > 10)
        {
            break;
        }
    }

    return 0;
}
```

- Affiche les itérations 1, 3, 5, 7, 9 et 11

Les instructions de branchement

- Goto
  - Lorsqu'elle est rencontrée au cours d'un programme, elle provoque le branchement immédiat dans l'emplacement indiqué
    - Syntaxe : goto label;
  - Le label est une instruction qui identifie un emplacement du programme pour la suite de l'exécution
    - Il est constituée d'un identificateur (étiquette) suivi de deux points, puis d'une instruction C
      - label : instructions;

Les instructions de branchement

- Goto : Exemple

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    int age;
    printf("Veuillez renseigner votre age\n");
    scanf("%d", &age);
    if (age < 18)
    {
        goto acces_refuse;
    }
    else
    {
        printf("Vous êtes majeur et vous avez accès au programme\n");
        goto fin;
    }

    acces_refuse:
        printf("Désolé, ce programme n'est pas accessible au mineur\n");
    fin:
        printf("Au revoir\n");
    return 0;
}
```

Les instructions de branchement

- Goto (suite)
  - L'instruction goto et son étiquette peuvent se trouver dans des blocs différents
    - L'étiquette associée à une instruction goto peut se situer avant ou après cette instruction dans le code
  - L'utilisation de Goto est déconseillée
    - Le résultat qu'il produit peut généralement être obtenu à travers l'usage des autres structures du langage
    - Son usage intensif mène à la production de codes difficiles à lire
      - Production de codes affreux communément appelés code Kangourou ou code Spaghetti

Imbrication de boucles

- Il est possible de définir des boucles dans d'autres boucles
  - On dit que les boucles sont imbriquées

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    int table, i;
    printf("Affichage des tables 1 à 10\n");

    for (table = 1; table <= 10; table++)
    {
        for (i = 1; i <= 12; i++)
        {
            printf("%d x %2d = %d\n", table, i, table * i);
        }
        printf("--------------------\n");
    }
    return 0;
}
```

Imbrication de boucles

- Les boucles imbriquées peuvent ne pas être de même type
  - On peut avoir une boucle For dans une boucle while par exemple
- Quand break ou continue est appelée au sein d'une boucle imbriquée
  - Elle s'applique uniquement au niveau de la boucle où elle est définie
- Goto permet d'aller vers n'importe quel endroit du code
  - Du moment qu'une étiquette y est définie au préalable

Les types complexes

Les tableaux

- Un tableau en C est une variable de type complexe
  - Il permet de stocker, les uns à la suite des autres, des éléments de même type
  - Il simplifie dés lors l'accès aux données et leurs traitements
- Les éléments du tableau sont de n'importe quel type du langage
  - Types primitifs
  - Type tableau
  - Type pointeur
  - Type structure
- On distingue des tableaux unidimensionnels et multidimensionnels

Les tableaux unidimensionnels

- Ils contiennent des éléments de types élémentaires
- Définition
  - type nom_du_tableau [nombre_element]
- Exemple
  - int scores [5] : définit un tableau pouvant contenir cinq entiers
- La taille d'un tableau correspond à son nombre d'éléments
  - Elle doit être connue statiquement
    - #define TAILLE 5
    - float notes [TAILLE]; // possible
    - int taille = 5; float notes [taille]; // impossible

Les tableaux unidimensionnels

- Il est possible d'initialiser les éléments du tableau lors de la déclaration
- Exemples
  - char t[7]={'B','o','n','j','o','u','r'};
  - int t[7]={1,2,3,4,5,6,7} ;
  - int t[7]={1,2}; // initialise les deux premiers éléments et fixe les autres à 0
  - int t[7] = {0} ; // initialise tous les élements du tableau à 0

Les tableaux unidimensionnels

- Il est possible de ne pas renseigner la taille du tableau lors de la déclaration
  - Le compilateur prend alors le nombre d'éléments spécifiés dans l'initialisation
- Exemple
  - int t[] = {1,2,3}; // crée un tableau de 3 entiers
  - char t[] = {'B','o','n','j','o','u','r'}; // crée un tableau de 7 caractères
- Cas particulier des constantes de chaines de caractères
  - le compilateur ajoute le caractère de fin de chaine (caractère nul)
  - Exemple
    - char c[]="bonjour"; // a une taille de 8 éléments

Les tableaux unidimensionnels

- Pour accéder à un élément d'un tableau, on utilise la syntaxe suivante : nom_tableau [indice]
- L'indice est une expression devant renvoyer une valeur entière positive ou nulle
  - Elle correspond à la position de l'élément dans le tableau
  - Les indices se trouvent entre 0 et N-1, N étant la taille du tableau
- Exemple
  - t[0]; permet d'accéder au premier élément du tableau
  - t[i] permet d'accéder à l'élément à la position i + 1

Les tableaux unidimensionnels

- L'élément récupéré a le même type que le tableau
  - Il peut être manipulé comme n'importe quelle autre variable
- Exemples:
  - int i = 2;
  - float notes[] = {14, 18.5, 16};
  - notes[0] = 14.75; //modifie la première note
  - printf("Dernière note : %f", notes[i]); // affiche la dernière note
  - scanf("%f", &notes[1]); // permet de saisir la seconde note
- A quels éléments correspondent notes [--i], notes [i--], notes[i++] et notes [++i] ?

Les tableaux multidimensionnels

- Ils permettent de stocker des ensembles de plusieurs dimensions
  - Exemple : matrice, coordonnées géographiques sur trois dimensions, etc.
- Les éléments de tels tableaux correspondent à des tableaux
- Définition : type nom_tableau [nb_elts1] [nb_elts2]...[nb_eltsN];
  - Elle se fait de la même manière que les tableaux unidimensionnels
    - La seule différence réside dans le nombre de dimensions
- Exemple
  - int t[3][5] crée un tableau de 15 éléments de type int

t[0] [0]	t[0] [1]	t[0] [2]	t[0] [3]	t[0] [4]
t[1] [0]	t[1] [1]	t[1] [2]	t[1] [3]	t[1] [4]
t[2] [0]	t[2] [1]	t[2] [2]	t[2] [3]	t[2] [4]

Les chaines de caractères

- En C, les chaines de caractères correspondent à des tableaux de caractères
- Contrairement aux constantes de chaines de caractères que nous avons déjà vues (comme "maChaine"), les tableaux de caractères créent une suite de caractères modifiables
- Exemples
  - char prenom [20]; // déclare une variable « prénom » pouvant prendre jusqu'à 20 caractères
  - char nom[15] = {'D', 'i', 'o', 'p'}; // seuls les 4 premiers éléments sont initialisés
  - char profession [] = "etudiant"; // contient 9 éléments (ajout \0)

Les chaines de caractères

- Pour afficher une chaine de caractères
  - Il faut utiliser l'option %s avec la fonction printf
- Pour lui affecter la valeur saisie par l'utilisateur
  - Il faut utiliser l'option %s avec la fonction scanf
  - Spécifier directement la chaine de caractères sans l'opérateur d'adresse
- Exemple

```
char prenom[20];
printf("Votre prenom : ");
scanf("%s", prenom);
printf("Votre prénom est : %s \n", prenom);
```

Les chaines de caractères

- D’autres fonctions de lecture et d’affichage existe
- Gets
  - lit une chaîne de caractères dans l’entrée standard et la stocke dans le tableau de caractères associé
- Puts
  - Affiche la chaîne associée sur la sortie standard et va à la ligne suivante
- Exemple
  - char adresse[45];
  - printf(”Donner votre adresse”); gets(adresse);
  - puts(adresse);

Opérations sur les chaines de caractères

Longueur d'une chaine

- strlen (nom_du_tableau)
  - Donne le nombre d'éléments du tableau
- Exemple
  - char prenom [45] = "Moussa";
  - int longueur = strlen(prenom); // vaut 6, le nombre de caractères de la chaine
  - int espaceOccupee = sizeof(prenom); // vaut 45, le nombre d'octects occupés par le tableau

Opérations sur les chaines de caractères

Concaténation de chaine

- strcat (nom_premier_tableau, nom_deuxième_tableau);
  - concatène la seconde chaîne à la fin de la première
  - Le premier doit avoir suffisamment d'espace pour accueillir le résultat de la concaténation
- Exemple
  - char chaine1[25] = ''Bonjour à '';
  - char chaine2[14] = ''tout le monde'';
  - strcat(chaine1,chaine2);
  - puts(chaine1);?
  - puts(chaine2);?

Opérations sur les chaines de caractères

Concaténation de chaine

- strcmp (nom_premier_tableau, nom_deuxième_tableau);
  - Compare les chaines, caractère par caractère dans l'ordre lexicographique, jusqu'à la rencontre d'une différence ou du caractère nul
  - Elle renvoie une valeur nulle si les deux chaînes sont semblables, sinon une valeur négative si la première chaîne est inférieure à la seconde et une valeur positive dans le cas contraire.
- Exemple:
  - char c1[40]=''Bonjour à'';
  - charr c2[40]=''Bonsoir'';
  - strcmp(c1,c2); ?

Les structures

- Jusque là, nous utilisions des variables simples, de types élémentaires
- Ces types ne sont pas adaptés à la manipulation de toutes les données
  - C'est le cas si on souhaite représenter des objets qui comportent plusieurs caractéristiques à traiter
    - Par exemple, des étudiants décrits par leur nom, prénom, sexe, date de naisssance, …
  - Représenter de telles informations avec les types de données élémentaires complique leur manipulation
    - Traitement complexe quand les données à manipuler augmentent
- D'où la nécessité de types structurés en C

Définition d'une structure

- Pour créer un type structuré en C, il faut donner le nom ainsi que ses différents composants

```
struct ma_structure {
        type_1 variable_1;
        type_2 variable_2;
        ...
        type_n variable_n;
};
```

- Exemple

```
struct Personne {
        char nom[20];
        char prenom[30];
        int age;
        char sexe;
};
```

Définition d'une variable de type structuré

- On définit une variable de type structuré comme on le fait avec les autres variables
  - Il faut penser à ajouter le mot struct
- Exemple
  - struct Personne p1;
  - struct Personne p2, p3;
  - struct Personne personnes[20];

Accès aux champs d'une structure

- L'accès aux champs de la structure se fait avec l'opérateur « . »
  - Il faudra le placer entre le nom de la variable et le nom du champ auquel on souhaite accéder
  - Les variables internes à la structure vont donner leurs noms aux champs de la structure
- Exemples
  - p1.sexe // accès au champ sexe
  - p1.prenom; // accès au champ prénom
  - personnes[5].nom; // accès au nom de la 6ème personne présente dans le tableau « personnes »

Initialisation des champs d'une structure

- Cela peut se faire via les champs eux-mêmes ou par une affectation directe (comme avec les tableaux)
- Exemples
  - struct Personne p1 = {"Sarr","Fatou", 24, 'F'};
    - Il faudra respecter l'ordre des champs fourni à la définition
  - struct Personne p2;
  - p2.age = 24;
  - p2.sexe = 'F';
  - p2.nom = "Sarr"; // will it work ?
  - p2.prenom = "Fatou"; // will it work ?

Manipulation des champs d'une structure

- La manipulation se fait comme si on disposait des éléments de types élémentaires
- Exemples
  - struct Personne p1;
  - scanf(”%s”, p1.nom);
  - scanf(”%d”, &p1.age);
  - printf(”%s a %d ans”, p1.nom, p1.age);

Utilisation de typedef

- Il n'est pas très commode d'inclure à chaque fois le mot clé struct lors de la déclaration de variables d'un type structuré
  - Pour contourner cela , nous utiliserons typedef pour associer le type à un nom
- De manière générale, typedef permet de nommer un type comme on le souhaite
  - typedef type_donné nom_voulu;
  - Exemple : typedef int entier; entier i;
- Pour renommer, struct Personne en Personne
  - Il faudra faire : typedef struct Personne Personne;
  - On pourra ainsi déclarer une personne en faisant : Personne p;

Utilisation de typedef

- On peut également utiliser typedef à la définition de la structure

```
typedef struct {
    int matricule;
    char designation[25];
} Piece;
```

- Une fois définie, on peut également déclarer des variables de type Piece en procédant ainsi :
  - Piece p1,p2,p[12];

Les énumérations

- Permettent de créer des types adaptés à un certain contexte
  - Pour la création de types censés contenir un ensemble de valeurs bien déterminé
    - Par exemple, un type pour les jours de la semaine, le sexe d'une personne (masculin / féminin), le niveau d'étude (licence / master / doctorat), ou encore les mois de l'année
- Pour déclarer un type énuméré, il faut lister les valeurs que peut prendre une variable de type énumération
  - enum nom_du_type { valeur_1, valeur_2, valeur_3, ..., valeur_n };

Définition d'une énumération

- Exemple :
  - enum JourOuvrable { LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI}
- Il est possible d'affecter un entier à chaque valeur énumérée
  - enum Sexe { MASCULIN = 1, FEMININ };
  - enum Volume { MUET = 0, FAIBLE = 10, MOYEN = 50, ELEVE = 100 };
- L'affectation d'entiers aux différentes valeurs énumérées n'est généralement pertinente que si on compte les utiliser

Déclaration d'une énumération

- La déclaration d'une variable de type énuméré se fait comme si on disposait de variables de types élémentaires
  - Il faudra penser à inclure le mot clé enum
  - Exemple : enum Sexe sexe_bebe = MASCULIN;
- On peut utiliser typedef pour se débarrasser du mot clé enum dans les déclarations de variables
  - On fait typedef enum Sexe Sexe;
  - Ou on le fait à la déclaration du type énuméré en faisant:

```
typedef enum {
        MASCULIN, FEMININ
} Sexe;
```

Les énumérations

```c
#include <stdio.h>

typedef enum Jour Jour;

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

enum Sexe {
    MASCULIN = 1,
    FEMININ
};

int main () {
    Jour jour_naissance = LUNDI; // jour_naissance = 0
    Jour jour_anniversaire = 1; // jour_anniversaire = MARDI;
    enum Sexe sexe = FEMININ; // sexe = 2;

    if (jour_naissance == LUNDI) {
        printf("Vous êtes né un Lundi\n");
    }
    if (sexe == 2) {
        printf("Vous êtes de sexe féminin\n");
    }
    printf("Naissance : %d\n", jour_naissance);
    printf("Sexe : %d\n", sexe);

    return 0;
}
```

Les fonctions

Définition

- Une fonction est un sous-programme, un bloc d'instructions qui permet d'effectuer la même action plusieurs fois, ou de réaliser cette même action sur des objets différents
  - Simplifie la lecture et la comprehension d'un programme en le découpant en modules
  - Maximise la réutilisation de code
- Une fonction peut :
  - ne retourner aucune valeur (procédure)
  - modifier les valeurs de certains de ses arguments (ou paramètres)
  - avoir des arguments en nombre variable

Exemple

```c
int factoriel(int nombre)
{
    int i, resultat = 1;
    for (i = 2; i <= nombre; i++)
    {
        resultat *= i;
    }
    return resultat;
}

int main()
{
    int factoriel1, factoriel2, nombre1, nombre2;

    printf("Veuillez renseigner un nombre\n");
    scanf("%d", &nombre1);
    printf("Veuillez renseigner un second nombre\n");
    scanf("%d", &nombre2);
    factoriel1 = factoriel(nombre1);
    factoriel2 = factoriel(nombre2);
    printf("%d! = %d et ", nombre1, factoriel1);
    printf("%d! = %d\n", nombre2, factoriel2);
    return 0;
}
```

Déclaration

- La déclaration d'une fonction est composée de deux parties
  - L'entête (ou interface) de la fonction
    - Constitué de la classe de mémorisation (optionnelle), du type de la valeur renvoyée, du nom de la fonction, d'une parenthèse ouvrante, de la liste (optionnelle) des paramètres (avec leurs noms et leurs types) et d'une parenthèse fermante
  - Le corps de la fonction est constitué de :
    - Une accolade ouvrante
    - Définitions et déclarations éventuelles de variables
    - Instructions que doit réaliser la fonction
    - Une accolade fermante

Déclaration

- Exemple de déclarations

```
float carre (float nombre)
{
    return nombre * nombre;
}

static int myFonction(void)
{
    …
}
```

```
void calcul(int u, float f)
{
        ....
}

static int myFonction()
{
    ....
}
```

Déclaration

- La classe de mémorisation d'une fonction peut prendre deux valeurs
  - static : elle n'est utilisable que dans le module ou elle est définie
  - extern (ou vide) : elle devient globale et est donc accessible à l'extérieur du module où elle est définie
- L'instruction « return » d'une fonction
  - Termine le programme et rend la main (ainsi que la valeur de retour) à l'instance appelante
  - N'est pas nécessaire dans une fonction qui ne retourne rien (procédure)

Déclaration

- La déclaration d'une fonction doit se faire en dehors de toute autre fonction
  - Elle peut se faire avant la fonction main
  - Elle peut se faire après la fonction main
    - Dans ce cas, il faudra déclarer le prototype de la fonction au préalable
- Le prototype d'une fonction n'est rien d'autre que son entête
  - Il faudra le faire suivre d'un point-virgule (;)
  - Exemples
    - float carre (float nombre);
    - float carre (float); // le nom du paramètre n'est pas obligatoire

Déclaration

- Si la fonction « main » utilise « factoriel », la déclaration doit se faire en utilisant l'une ou l'autre des approches suivantes :

```
int factoriel(int nombre)
{
    // ...
}

int main()
{
    // ...
}
```

```
int factoriel(int nombre);

int main()
{
    // ...
}

int factoriel(int nombre)
{
    // ...
}
```

Portée d'une variable

- Portée d'une variable et d'une fonction
  - La manière de déclarer une variable ou une fonction joue sur sa portée, c'est-à-dire la zone dans laquelle elle est visible et utilisable
- Variables globales et locales
  - Une variable locale est définie à l'intérieur d'une fonction ou d'un bloc
    - Elle n'est accessible que là où elle est définie (dans la fonction / le bloc)
  - Une variable globale est définie hors de toutes fonctions ou blocs
    - Elle est accessible dans toutes les fonctions et blocs du programme, à moins qu'elle n'y soit redéfinie

```c
#include <stdio.h>

int variable_globale = 8;

int main(int argc, char const *argv[])
{
    int variable_locale = 5;
    printf("Dans main, variable_globale vaut %d\n", variable_globale);
    if (variable_globale == 8)
    {
        // isOk est uniquement accessible dans ce bloc
        int isOk = 1;
    }
    // L'instruction suivante provoque une erreur
    printf("IsOk : %d\n", isOk);
    return 0;
}

void fonction1()
{
    int variable_globale = 6;
    // affiche 6 et non pas 8
    printf("Dans F1, variable_globale vaut %d\n", variable_globale);
    // L'instruction suivante provoque une erreur
    printf("La variable locale vaut %d\n", variable_locale);
}
```

Portée d'une fonction

- Les fonctions adoptent, pour leur déclaration, les mêmes règles que les variables

```c
#include<stdio.h>
int carre(int); //Fonction globale
int main()
{
    int i=4;
    printf("Le carré de %d vaut: %d\n", i, carre(i));
    if(i == 4)
    {
        int triple(int); //Fonction locale
        printf("Le triple de %d vaut: %d\n", i, triple(i));
    }
    return 0;
}
int carre(int x){ return (x * x);}
int triple(int x){ return (3 * x);}
```

Valeur de retour d'une fonction

- La valeur retournée par une fonction est celle renvoyée par cette dernière une fois son exécution terminée
  - Elle peut être de type quelconque (sauf tableau)
  - Le renvoi d'une valeur se fait à travers l'instruction « return »

```
int factoriel(int nombre)
{
    int i, resultat = 1;

    if (nombre == 0)
    {
        return 1;
    }

    for (i = 2; i <= nombre; ++i)
    {
        resultat *= nombre;
    }
    return resultat;
}
```

Arguments d'une fonction

- Les arguments d'une fonction peuvent être de type quelconque
- Pour les arguments de type tableau
  - Spécifier la taille d'un tableau passé en paramètre ne sert à rien
    - Le compilateur ne le prend pas en compte
    - Exemple : void showTab(int tab[]) et void showTab(int tab[10]) sont équivalentes
  - Il n'y a aucun moyen de connaitre la taille du tableau dans la fonction
    - Elle doit être définie comme variable globale ou être spécifiée comme paramètre à la fonction
    - Exemple : void showTab(int tab[], int taille)

Le type void

- Ce type est utilisé lorsqu'une fonction :
  - ne doit pas renvoyer de valeur
    - La fonction peut être considérée dans ce cas comme une procédure
    - Exemple : void afficherTableau(float moyenne[]);
  - Ne prend pas de paramètres
    - Devient optionnel dans ce cas
    - Exemple : void effacerEcran(void)
      - void effacerEcran(void) et void effacerEcran() sont équivalentes

Passage de paramètres

- Le passage des paramètres par valeur
  - Ce n’est pas la variable en elle-même qui est passée en paramètre, mais plutôt sa valeur
    - La fonction ne peut donc pas modifier la variable passée en paramètre

Passage de paramètres

- Le passage des paramètres par valeur : exemple

Passage par valeur

```
void f(int n) {
    n=n+1;
}
int main(void) {
    int i=2;
    printf("1: i=%d",i);
    f(i);
    printf("2: i=%d",i);
}
```

Donne le résultat suivant :

```
1: i=2
2: i=2
```

Comment faire si on veut que les modifications réalisées dans la fonction ne soient pas perdues ?

Passage de paramètres

- Le passage des paramètres par adresse
  - Pour que la valeur de la variable passée en paramètre puisse être modifiée par la fonction, l'argument devra être passé par adresse
    - Ce n'est pas la variable en elle-même qui est passée en paramètre, mais plutôt son adresse
    - On utilise pour celà l'opérateur d'adressage « & »
  - Connaissant son adresse, la fonction peut aller directement modifier l'emplacement mémoire réservé à la variable fournie en paramètre
    - On utilise pour celà l'opérateur d'indirection « * »

Passage de paramètres

- Le passage des paramètres par adresse : exemple

```
void f(int *n) {
   *n=*n+1;
}

int main(void) {
   int i=2;
   printf("1: i=%d",i);
   f(&i);
   printf("2: i=%d",i);
}
```

Donne le résultat suivant :

```
1: i=2
2: i=3
```

On obtient bien le comportement attendu !

Passage de paramètres

- Passage de fonctions en paramètres
  - Il est possible en C de passer une fonction comme paramètre d'une autre fonction
    - Pour cela, il faut définir un pointeur vers la fonction à passer comme paramètre
  - Pointeur de fonction
    - Exemple : int (*afficher)() = printf;
      - mettrait les mêmes fonctionnalités de printf dans la fonction afficher
  - Règles de construction
    - Donner le même type de retour que la fonction, les arguments sont inutiles et nous permettent d'indexer plusieurs fonctions de même types de retour

Passage de paramètres

- Passage de fonctions en paramètres
  - Il faut juste passer le pointeur de la fonction en paramètre

```
#include <stdio.h>
#include <math.h>
float tangente( double (*f)(), double (*p)(), double x)
{
    return (*f)(x)/(*p)(x);
}
int main() {
    printf("la tangente de 3.14 est %f:", tangente(sin, cos, 3.14) );
}
```

Appel d'une fonction

- Une fois qu'une fonction est définie (et éventuellement déclarée), il est possible de l'appeler
  - Comme une variable, une fonction est uniquement visible dans sa zone de déclaration (cf. portée des variables et fonctions)
    - C'est dans cette zone qu'elle peut être appelée
  - Syntaxe
    - nom_fonction(liste_éventuelle_des_paramètres)
    - Exemple : factoriel(12);

Appel d'une fonction

- Une fois qu'une fonction est appelée
  - Les paramètres effectifs (s'ils existent) sont passés aux paramètres formels
  - Le contrôle est passé à la fonction appelée
  - La fonction appelée se termine à la rencontre d'un « return » ou à l'exécution de sa dernière instruction
  - La main est retournée à la fonction appelante qui continue son exécution

Fonction récursive

- Il est possible d'appeler une fonction dans sa propre définition
  - On obtient ainsi une fonction récursive
- L'usage de fonctions récursives peut aboutir à des récursions infinies
  - si la condition de base n'est pas correctement spécifiée
    - Elle est l'équivalente de la condition d'arrêt au niveau des boucles
- Exemple d'utilisation
  - Calcul de factoriel
  - Vérification de palindromes
  - Manipulation de suites numériques

Passage d'arguments depuis la ligne de commande

- Il peut arriver que l'on ait besoin de passer des paramètres avant l'exécution du programme
  - Il faut alors donner des arguments la fonction main
  - Ses arguments sont des paramètres passés en ligne de commande.
- Avec ces paramètres, on aura deux éléments :
  - Le nombre de paramètres (y compris le nom du programme) dans argc
  - La liste des paramètres sous forme d'un tableau de chaînes de caractères stockée dans argv[]
- La signature de la fonction principale devient alors :
  - int main(int argc,char* argv[ ])

Variables locales statiques

- Ce sont des variables
  - locales pour leur visibilité;
  - statiques (cad permanentes) pour leur durée de vie
- Exemple:

```
void f() {
    static int compteur = 1000;
    printf("%d ", compteur );
    compteur ++;
}
```

Variables locales statiques

- Lorsque la déclaration d'une telle variable comporte une initialisation, il s'agit de l'initialisation d'une variable statique :
  - elle est effectuée une seule fois avant l'activation du programme
- Une variable locale statique conserve sa valeur entre deux exécutions consécutives de la fonction
  - Ainsi, des appels successifs de la fonction précédentes produisent l‘affichage des valeurs 1000, 1001, 1002, etc.

MERCI DE VOTRE ATTENTION

Université Cheikh Anta Diop
Ecole Supérieure Polytechnique
Département Génie Informatique

PROGRAMMATION EN C

COURS INTRODUCTIF

Formateur

M. Mouhamed DIOP

mouhamed.diop@esp.sn

DUT2 / ESP

LICENCE 3 / EC2LT