Langage C

Cours 1

```c
/* fichiers en-tête qui contiennent les déclarations (prototypes)
* de fonctions utilisées dans le programme */
#include <stdio.h>
int main(void){
   int tab[]={-5, 8, 12, 9, -4, 21, -31};
   double s = 0;
   /* calculer le nombre d'elements de tab */
   int taille = sizeof(tab)/sizeof(tab[0]);
   /* calculer la somme de tous les éléments */
   for(int i=0 ; i <  taille ; i++){
     s += tab[i];
   }
   s/=taille; /* la moyenne s = s/taille */
   printf("somme=%8.3f\n", s); /*imprimer le resultat*/
   return 0; /*main doit retourner un int*/
}
```

Explications

- L'exécution de programme commence toujours dans la fonction main:
  - int main(void){ }
- main() doit retourner un entier, la valeur 0 indique terminaison correcte du programme, une valeur > 0 un terminaison incorrecte
- La fonction printf() est définie dans le fichier en-tête stdio.h sert à faire afficher les valeurs d'expressions

Compiler le programme C

compiler un programme qui se trouve dans fichier hello.c sur un terminal :

- gcc -Wall hello.c -o hello
- gcc - le compilateur C

-Wall une option de compilateur (obligatoire). Permet de voir tous les avertissements (warning). Compilation avec des avertissements signifie que le programme est presque sûrement incorrect (mais si la compilation réussie).

Exécuter le programme compilé depuis le terminal : ./hello

Types entiers

Types entiers signés :

- signed char
- short (short int)
- int
- long (long int)
- long long (long long int)

Types entiers non-signés :

- unsigned char
- unsigned short
- unsigned int
- unsigned long
- unsigned long long

Types pour les nombres réels

- float (à ne pas utiliser)
- double
- long double

Comment C calcule la valeur d'une expression arithmétique ?

Si on mélange des réels et des entiers alors tous les arguments sont transformés en réels appropriés et le résultat est réel.

Avertissement : Les règles de calcule sont peu intuitives si une expression mélange les entiers signés et non-signés.

La règle de bon sens fortement recommandée:

- ne mélangez jamais les entiers signés et non-signés ni dans les expressions ni dans les relations (ou faites un cast sur les entiers non-signés)
- utilisez les non-signés uniquement pour les opérations bit à bit et pour rien d'autre

Expressions arithmétiques comme conditions logiques

C traite une expression arithmétique dont la valeur est différente de 0 comme la valeur logique VRAI et une expression arithmétique dont la valeur est 0 comme FAUX.

Relations (renvoie 1 ou 0, pas un booléen)

- a < b
- a <= b
- a > b
- a >= b
- a == b
- a != b

if/switch

```c
if( condition1 ){
  instructions1
}
else if( condition2 ){
  instructions2
}
...
else{
  instructions_n
 }
```

```c
switch( expression ) {
  case expr-const : instructions ;
  case expr-const : instructions ;
  default : instructions ;
}
```

Boucle

```c
while( condition ){
  instructions
}
```

Remarque : si le corps de la boucle est vide alors on écrit

```c
while( tab[i++] )       ; /* boucle vide, il faut un élément qui contient 0 pour l'arrêter */
```

```c
do {
instructions
 }
while( condition );
```

(1) on exécute les instructions

(2) on vérifie la condition, si satisfaite alors on revient à (1)

sinon on termine la boucle. La condition vérifiée après chaque exécution de la boucle.La boucle exécutée au moins une fois.

Langage C

```
for( initialisation ; condition ; incrémentation ){ }
```

- `initialisation` exécutée une fois, avant l'entrée dans la boucle
- `condition` est vérifiée au début de la boucle et si vraie alors la boucle est exécutée
- `incrémentation` évaluée à la fin de chaque boucle et on revient au début, à la vérification de la condition

`break` provoque la sortie de la boucle, le saut vers la première instruction après la boucle

`continue` termine l'itération courante de la boucle, on passe à l'itération suivante (on revient au début de la boucle et on refait le test de la condition de terminaison)

Affection

- a *= b -> a = a * b;
- a /= b; -> a = a / b;
- a += b; -> a = a + b;
- a -= b; -> a = a - b;

a = expr;
est une expression dont la valeur est égale à la valeur de expr. Cette expression a un effet de bord qui consiste à affecter une nouvelle valeur à la variable a.

- a = b = c*d;

Tableau/vecteur

- il faut initialiser la taille des tableaux : int tab[5] (de 0 à 4)
- pour obtenir le nombre d'élément : sizeof(tab)/sizeof(tab[0])
  - sizeof() renvoie le nombre d'octet
  - ça ne marche pas quand le tableau est en paramètre d'une fonction, on doit ajouter la taille en paramètre sinon)

Fonctions

- il faut déclarer les fonctions (structure la fonction)
  - fonction (type parametre, ...)
    - il faut au moins le type, le nom de paramètre est facultatif

```
double somme(int nb, double tab[]);

int main(void){
   double tab[] = {-4.8, 6.1, 57.0, 23.99, -11.32, 4.5};
   int n = sizeof(tab)/sizeof(tab[i]);
   double s = somme(n, tab);
   printf("somme = %f\n",s);
}

double somme(int nb_elem, double t[]){
   double s=0;
   for(int i=0 ; i < nb_elem; i++)
      s += t[i];   return s;
}
```

Langage C

Cours 2

Eléments basiques

Vecteur

Les changements de valeur d'un tableau pris en paramètre d'une fonction, s'applique dessus directement sans return (on ne pas return un tableau)

Formatage du code

- Une déclaration par ligne
- Une instruction élémentaire par ligne
- les accolades ouvrante termine la ligne
- Une seule accolade fermante sur une ligne
- l'étiquette est seule sur une ligne
- if et else sur la même colonne
- Une ";" sur la ligne suivant une boucle vide

Opération à effet de bord

avec int x = 7

- y = ++x
  - y = 8, x = 8
- y = x++
  - y = 7, x = 8

Expression ternaire

condition ? val1 : val2 => si la condition est vraie alors val1 sinon val2

Exécution du code :

Quand on lance gcc il y a en fait trois programmes différents qui s'exécutent à tour de rôle :

- Le préprocesseurs transforme le texte du programme suivant les directives commençant par #. Le résultat est un fichier texte.
- Le compilateur traduit le texte du programme vers un code binaire qui n'est pas encore exécutable.
- Le linker rassemble les différentes parties du code binaire, ajoute les références vers les fonctions des bibliothèques. Le résultat est un code binaire exécutable sur la machine.

Structure (plus ou moins équivalent class java)

exemple

```
struct personne {
      char sex;
      unsigned int année;
      char nom[20];
      char prenom[20];
}; // ne pas oublier le point-virgule
```

```
struct personne bill {
      sex = 'm';
      année = 1996;
      nom = "clinton";
      prenom = "bill";
}
```

```
struct personne beta; //déclare une variable de type personne
/* On initialise les champs un par un
* beta.sex = 'm';
* beta.année = 1996;
```

- On compare des structures par leur champ et non directement (bill.année == beta.année)
- Pour éviter réécrire "struct" à chaque création d'instance on utilise :

```
typedef struct personne personne
// ou
```

```
typedef struct {
      char sex;
      unsigned int année;
      char nom[20];
      char prenom[20];
} personne; // structure anonyme, et
alias à la fin
```

- vecteur (tableau) de struct : personnage tab[10]
- une structure peut avoir une autre structure en champ;
- Bill = beta est une affectation possible (contrairement aux vecteurs)
- contrairement aux vecteurs, les champs modifiés dans une fonctions ne changent pas celle de la structure initiale, sauf si c'est retourné dans la variable : p1 = mirror(p1);
- on peut retourner une structure qui contient un vecteur mais pas retourner un vecteur

Enum

exemple

```
enum color {BLUE, RED, GREEN};
//énumère des constantes numériques
//typedef est aussi utilisé pour les alias
// on peut préciser la valeur
typedef enum color {BLUE = 1, RED = 4} color;
```

Goto (go to)

- sert à sortir d'une boucle intriquée

exemple

```
for(...){
      for(...){
            if(...){
                  goto et;
            }
      }
}
et:
      //suite code
}
```

digression : tableau à plusieurs dimensions

- int t[ ][4] = { {1, 2, 3, 4}, {5, 2, 3, 4}, {1, 1, 1, 8} };
- et non int t[ ] [ ];

Langage C

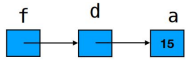
Langage C

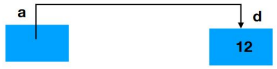
Cours 3

Pointeurs - arithmétique de pointeurs

Opérateur &

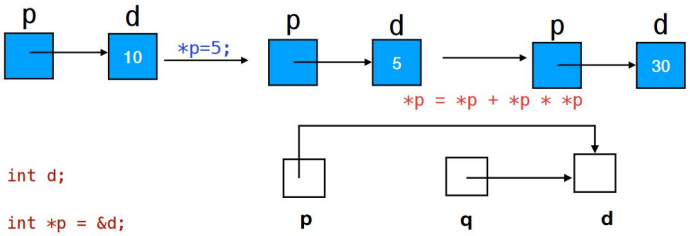
- variable de type pointeur pour mémoriser les adresses :
  - int b = 7;
  - int *ps = &b; (pointeur de l'adresse de b)
- En C le nom de tableau dans une expression est évalué comme l'adresse du premier élément du tableau
  - int *pt = tab[0];
  - int *pq = tab;
  - Les variables pt et pq contiennent l'adresse du premier élément de tab
- int *a, *b =/= int *c, d
- int a = 15; int *d = &a; int **f = &d;
- int d = 8; int *a = &d; *a = 12;





exemple

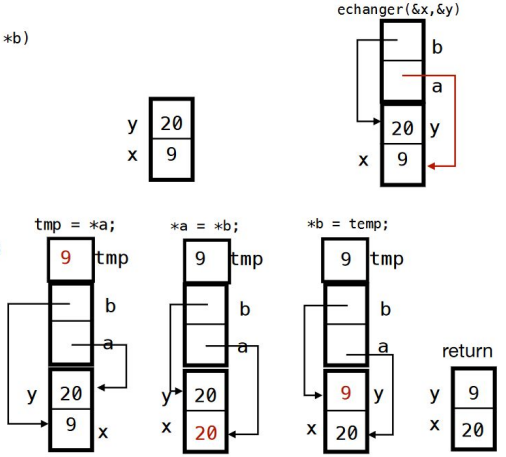
```
int *p;  int d; d = 10;
p = &d;
*p = 5;  /* mettre  la valeur 5 à l'adresse stockée dans p */
printf( "d=%d\n", d);   -->  d=5  d change la valeur
*p = *p + *p * *p ;
printf( "d=%d\n", d);            -->  x=30    x change la valeur
```




```
int d;

int *p = &d;

d = 10;

*p = (*p) * 2 + 5;      /* d prend la valeur 2*10 + 5 = 25 */


*p += 3;     /* incrémenter de 3 la valeur stockée à l'adresse p;
              * d reçoit 28 */

++(*p) ; /*   incrémenter un int qui se trouve à l'adresse donnée
          *   par p, d == 29   ++ s'applique à la valeur qui se trouve
          *   à l'adresse p     */

int *q = p ;  /* les pointeurs p et q contiennent l'adresse de d */

(*q)++; /* (*q)++ augmente la valeur int à l'adresse q,
         *  d == 30 */
```

exemple dans une fonction

```
void
echanger( int *a, int *b)
  {
    int tmp = *a;
    *a = *b;
    *b = temp;
    return;
}


int main(void){
    int x=9, y=20;
    ......
    echanger( &x, &y);


}
```



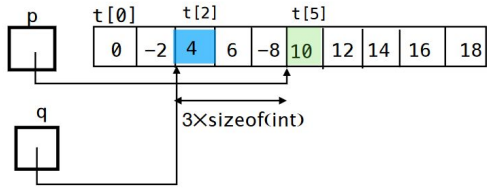
valeur NULL

- NULL défini dans : stdio.h stddef.h
- NULL une valeur spéciale pour les pointeurs, différente de toutes les adresses réelles.
  - pd == NULL
  - *pd = 5; provoque l'envoie d'un signal qui termine l'exécution de programme

arithmétique de pointeurs

- Si p est un pointeur de type t : t *p;
- n une expression de type int alors les valeurs des expressions
- p + n et p - n dépendent de type t du pointeur.

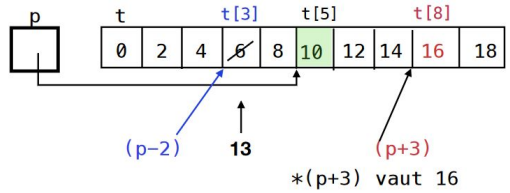
```
int t[]={0,-2,4,6,-8,10,12,14,16,18};
int *p = &t[5];

int *q = p - 3;

  p = &t[5];


  q = p - 3;
```



- Le décalage de l'adresse calculé en nombre d'octets est de n * sizeof(t)
- pour print un pointeur c'est le format : %p

exemple

```
int t[]={0,2,4,6,8,10,12,14,16,18};
int *p = &t[5];
*(p - 2) = *(p + 3) - 3;  /*
p[-2]=p[3]-3;*/
```



Nouvelle valeur de t[3] est *(p+3) - 3 = 16 - 3 = 13

Langage C

- Le compilateur C traduit **p[k]** et **p[-k]** automatiquement en : ***(p-k)** et ***(p+k)**
- Le compilateur duC ne fait (presque) aucune vérification si les adresses calculées à l'aide de pointeurs sont "correctes".
- Le résultat de la différence de deux pointeurs (du même type) est de typeptrdiff_t. Le type ptrdiff_t défini dans stddef.h
- ptrdiff_tun type entier signé qui dépend de l'implémentation.
- Lors de la traduction, le paramètre **int t[]** est traduit en **int *t**, avec comme valeur &tab[0]
- a = moy(4, &tab[3]); **/* moyenne sur les éléments tab[3]...tab[6] */**
- Il existe aussi les vrais pointeurs de tableau
  - int *p_t = &tab[0];
  - int *p_q = tab;
  - int (*p_tab)[4] = &tab;
  - Le type de la variable p_tab est différent des types des variables p_t et p_q (mais les trois variables contiennent la même adresse après les trois affectations).

exemple

```
int *somme( int n, int tab[]){
    int k;
    for( int i = 0; i < n; i++ ){
       k += tab[i];
    }
    return &k;
}
int main(void){
    int t[]={ 8, 9, 12, -15, -8};
    int *s = somme( 5, t );
    printf("somme = %d\n", *s );
    return 0;
}
```

```
Qu'est-ce qui se passe avec k quand on fait return de la
fonction somme() ?
Quand on fait return de somme(), les variables locales n,
tab, k de la fonction somme() sont dépilées donc somme()
retourne l'adresse qui n'est plus valable.
Ce programme n'est pas correct.
```

- A quoi sert le pointeur générique?
  - Nous pouvons faire une affectation entre un pointeur générique et un autre pointeur sans retypage "cast". C garantie que la valeur du pointeur est préservée.
  - Arithmétique de pointeurs ne s'applique pas aux pointeurs génériques
    - (t + 1) et (t - 1) => aucun sens
  - L'application de l'opérateur * n'a pas de sens pour le pointeurs génériques
    - int k = *t + 2; => aucun sens

Scanf(), lecture sur le terminal

- scanf() lit depuis les terminal et met les valeurs lues dans des variables, le premier paramètre de scanf() le format, tous les paramètres suivant sont des pointeurs donnant les adresses de variables où scant place les valeurs lues.
- scanf() retourne les nombres d'éléments lus
  - int a,b; int l = scanf("%i %i\n", &a, &b); **/* l sera 0 si la lecture échoue, 1 si la lecture de a réussie mais pas de b, 2 si la lecture de a et b réussie */**

- Un caractère blanc c'est un caractère espace ou le caractère de nouvelle ligne.
- Le caractère à l'entrée qui n'est pas "matché" par le format reste dans le tampon de l'entre, scanf() retourne sans lire la suite.

format	le type de paramètre	remarques
%10s	char * lire jusqu'à 10 caractères, ou jusqu'à l'espace	%s la plus longue chaîne sans espaces (attention danger !)
%lf %lg %le	double *	
%f %g %e	float *	
%d %i	int *	%d accepte uniquement la notation décimal %i accepte décimal, octal, hexadécimal
%ld %li	long int *	
%c	un caractère	%5c 5 caractères
%x %o %u	unsigned int *	
%lo %lu %lx	unsigned long int *	
%hd %fi	short *	
%ho %hu %hx	unsigned short *	

exemple

```
int t[4];  unsigned int u;   double d[4];  char c[30];

int k = scanf("%d %i %x", &t[0], &t[1], &t[2]);

for(int i=0; i<k; i++)
    printf("t[%d]=%d\n", i , t[i]);
```



```
-89   010   0x100   999.99  (Enter)
t[0]=-89
t[1]=8
t[2]=256
```

```
k = scanf("%le %lg %lf",  &d[0], &d[1], &d[2]);

for(int i=0; i<k; i++)
    printf("d[%d]=%4.2e\n", i , d[i]);
```

```
88e7 -77.777e3          (Enter)
d[0]=1.00e+03
d[1]=8.80e+08
d[2]=-7.78e+04
```

notez que 999.9 sera lu quand on envoie la deuxième ligne sur le termir et cette valeur sera arrondi à 1000