

Programmation structurée et Langage C



L'informatique à Phelma



- 1ère année : les objectifs
 - Comprendre et maîtriser le développement de logiciels
 - Apprendre la programmation structurée
 - Connaître quelques structures de données et algorithmes
- Semestre 1 : 16 séances de 2h
 - 14 séances : Environ 1 cours/TD et 1 sujet de TP pour 2 séances
 - TP préparés à l'avance, individuellement
 - Contrôle des préparations papier à chaque TP
 - Terminer les TP en dehors des séances encadrées si nécessaire
 - Conseil : taper le code à l'avance
 - 1 mini projet de 2x2h en fin de semestre
- Semestre 2:
 - PET : 18 séances de 2h : algorithme et structures de données, Cours, TP, Projet
 - PMP : 7 séances de 4h : projet
- Tous les documents sur le site tdinfo.phelma.grenoble-inp.fr/
- 1 examen écrit + contrôle continu (20%)

Première séance



■ Environnement Unix

- Le système de fichiers
- Les commandes de base

■ Développement logiciel

- Connexion
- Mon premier programme
- Un autre programme

■ Infos sur les TDs d'informatique (sujet, docs, etc...)

- <http://tdinfo.phelma.grenoble-inp.fr>

■ Pour tout problème lié à l'informatique

- Help@phelma.grenoble-inp.fr

■ Les autres sites :

- <http://webmail.minatec.grenoble-inp.fr>

- Accès à votre messagerie via un navigateur web

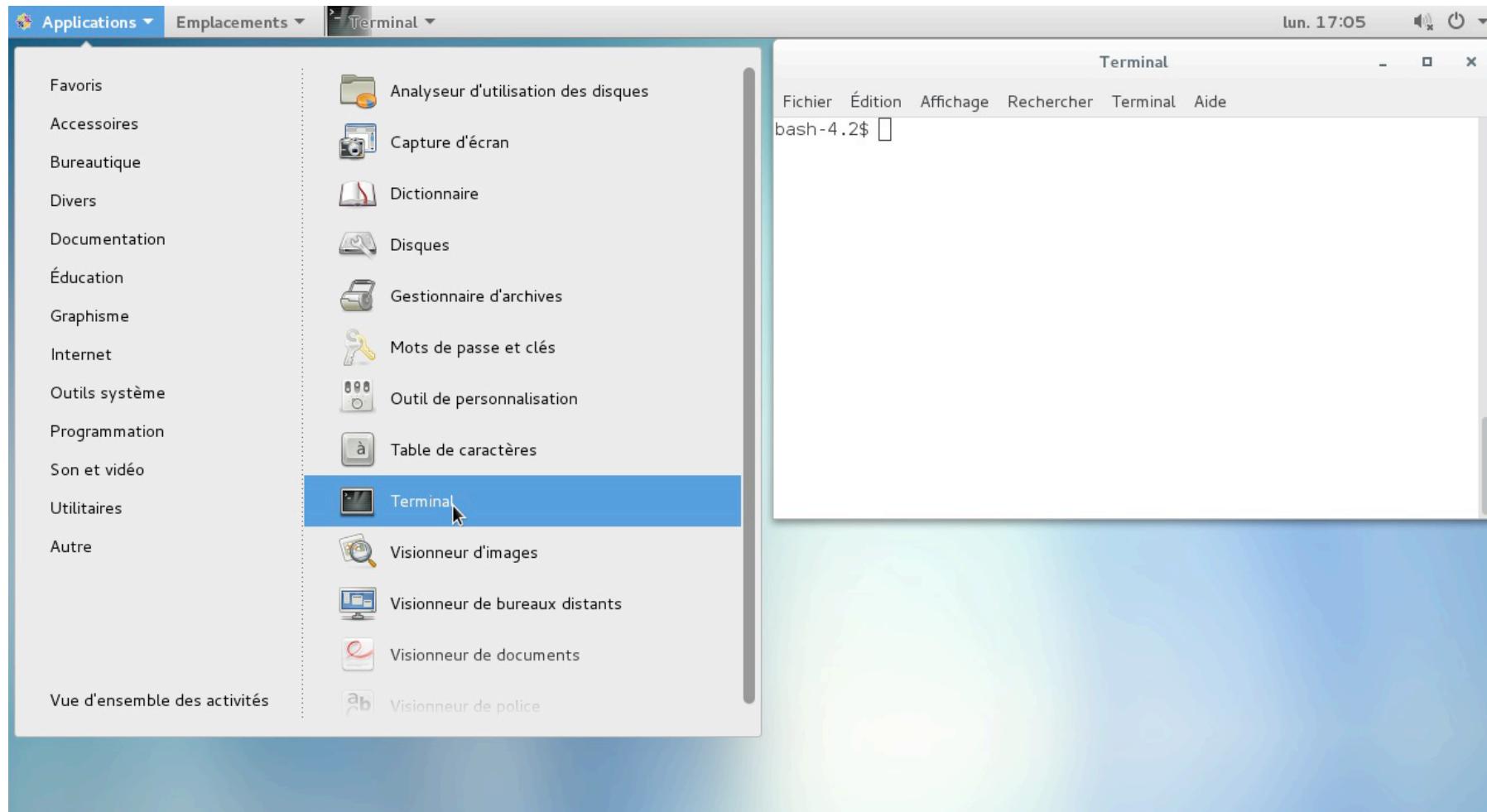
- <http://login.minatec.grenoble-inp.fr>

- Informations sur vos quota mail et impression, changement de votre mot de passe

- <http://prism.minatec.grenoble-inp.fr>

- Site d'informations et de support du service informatique

Lancer un terminal



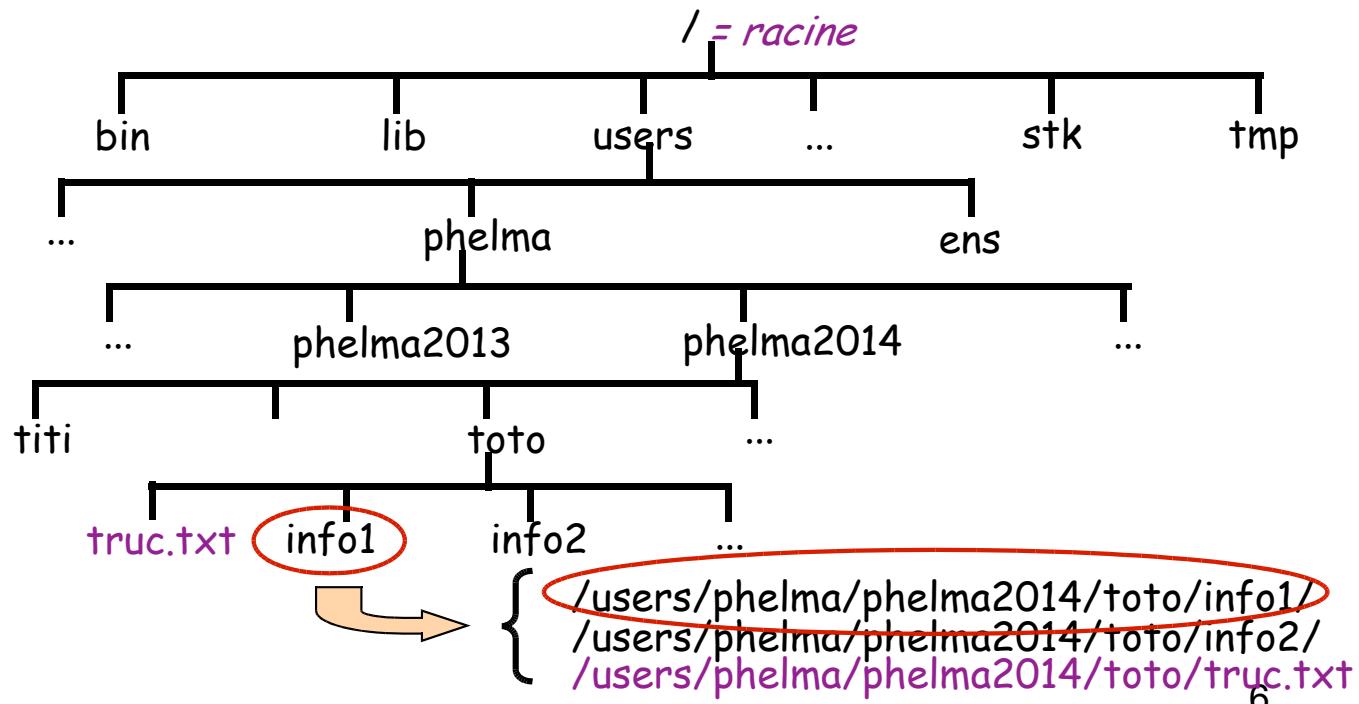
Le shell



- Le « shell » est l'interpréteur de vos commandes.
- Le « shell » dispose d'astuces vous facilitant la vie :
 - Le rappel de commande :
 - Les touches « flèche vers le haut » et « flèche vers le bas » permettent de se déplacer dans les dernières commandes déjà tapées
 - La compléction automatique
 - Lorsque vous saisissez des chemins, l'appui sur la touche « tabulation » permet de compléter automatiquement votre chemin avec le nom du fichier ou du répertoire disponible.
 - En cas d'incertitudes (plusieurs chemins possibles), appuyez deux fois sur la touche « tabulation », la liste des chemins possibles s'affiche.
- Le copier/coller : Sous Xwindows, le copier coller est possible via la souris
 - Le copier se fait automatiquement via une sélection du texte (clic gauche)
 - Le coller se fait grâce au bouton du milieu de la souris

Système de fichiers

- Le « système de fichiers » est une arborescence de dossiers où vous stockez tous vos fichiers et dossiers
- Le dossier racine est le dossier root : « / »
- Dans chaque dossier on peut créer d'autres dossiers ou des fichiers
- Les dossiers sont protégés contre les autres utilisateurs



Répertoires

- Chaque utilisateur dispose d'un espace de travail ou home directory
 - Grâce à des protocoles de partage de fichiers réseaux, ce répertoire est accessible de tous les postes clients.
 - Par défaut à la connexion on « tombe » dans sa « home dir »
 - Pour retourner dans sa home dir, on peut utiliser la commande : cd ~
- Tout répertoire ou fichier peut être désigné par rapport à sa « home dir » :
 - ~/tdinfo/td0
 - ~/..../users1/projet/td0
- Tout répertoire ou fichier peut être désigné par rapport à la racine: c'est le chemin absolu
 - /users/phelma/phelma2015/monlogin/tdinfo/td0
- Tout répertoire ou fichier peut être désigné par rapport à l'endroit où l'on est : c'est le chemin relatif
 - Si on est dans son répertoire HOME : tdinfo/td0

Unix : commandes

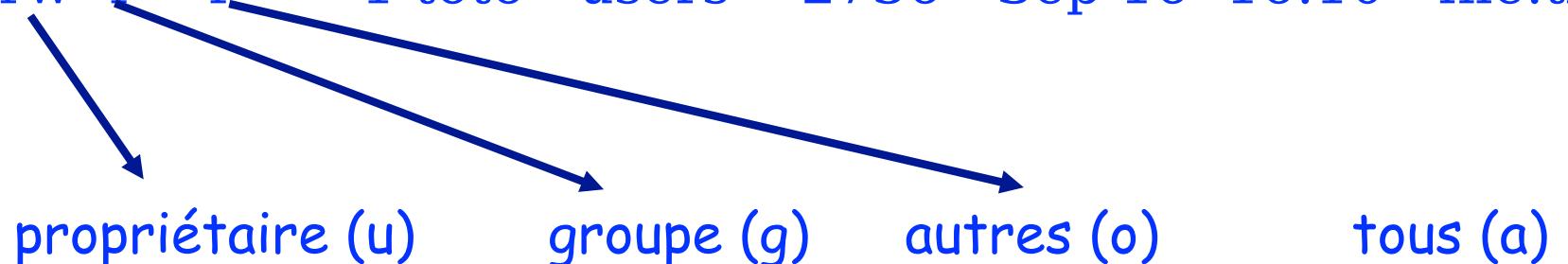
- man : permet de comprendre le fonctionnement d'une commande
 - Donne accès également aux informations des fonctions C standard. Appuyer sur 'q' pour quitter man
- pwd : permet de savoir dans quel répertoire on se situe
 - (pas d'argument)
- cd : permet de se déplacer dans un autre répertoire
 - (ex : de tdinfo, cd ../../tdinfo/user1)
- ls : permet de lister le contenu d'un répertoire
 - (ex., ls, ls -l, ls -la, pour les options voir le man)
- mkdir : permet de créer un répertoire
 - (ex., mkdir td0, mkdir ~/tdinfo)
- rmdir : permet de supprimer un répertoire (il doit être vide)
 - (ex., rmdir td0, rmdir /users/phelma/phelma2008/users1/tdinfo/td0)
- cp : permet de copier un ou plusieurs fichiers
 - (ex., cp file1.txt ../../test, cp ..//tdinfo/td0/file1.txt .)
- mv : permet de déplacer ou renommer un ou plusieurs fichiers
 - (ex., mv file1.txt file1Old.txt, mv file1.txt ../../tdinfo/td1)
- rm : permet d'effacer un ou plusieurs fichiers ou répertoires
 - (ex., rm file1.txt, rm -r dir1, rm -rf dir1)
- quota : permet de connaître son utilisation d'espace disque

Permissions/sécurité

- Trois cibles de permissions
 - u=user : utilisateur propriétaire
 - g=group : groupe propriétaire
 - o=other : tous les autres
 - (a=all : tout le monde)
- Trois types de droits sur fichier et sur répertoire
 - r=read : lecture listage
 - w=write : écriture ajout/suppression fichier
 - x=exec : exécution ou traversée
- ls -l pour lister, chmod pour modifier les permissions

```
ls -l file.txt
```

```
- rw- r- - r- - 1 toto users 2736 Sep 16 16:10 file.txt
```





Développer un programme

Comment créer un programme



- Un programme
 - indique à la machine ce qu'elle doit faire
 - succession d'octets (8 bits) qui sont des commandes reconnues par le processeur
- Il faut
 - Un cahier des charges
 - Analyse du problème
 - Cycle de développement
 - Saisie du code à l'aide d'un éditeur de texte
 - Compilation : correction des erreurs de syntaxe
 - Exécution, tests unitaires et débogage : correction des erreurs de logique en modifiant le code
 - Tests de validation

Exemple

- Cahier des charges
 - ✓ Faire un programme qui calcule l'inverse d'un nombre rentré au clavier

- Analyse : il faut 2 nombres réels

Rentrer le **nombre** dont on veut l'**inverse**

Calculer l'**inverse** du **nombre** et le **mettre** dans un autre **nombre**

Afficher à l'écran ce **nombre**

- créer le fichier contenant le programme C :

■ bash-3.00\$ atom prog1.c &

```
#include <stdio.h>

main(){ double x,y;      /* Créer deux nombres réels appelés x et y */
    printf("Entrer un nombre au clavier\n"); /*Affiche un message à l'écran*/
    scanf("%lf", &x);           /* Entrer un nombre au clavier, le mettre dans x */
    y = 1/x;                  /* Calculer l'inverse, le mettre dans y */
    printf("L'inverse de %lf est : %lf \n ",x,y); /* Afficher le résultat */
}
```

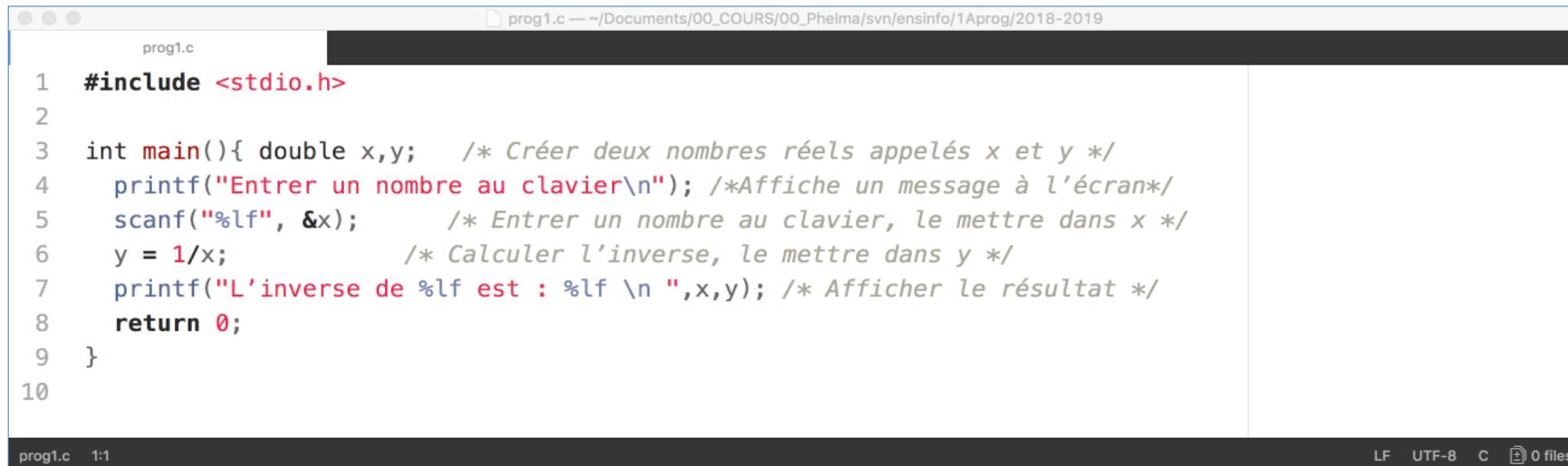
Exemple



```
bash$ atom prog1.c &
[1] 26329
bash$
```

atom est l'éditeur de texte que nous utiliserons cette année.

Il permet d'écrire le code du programme en langage C.



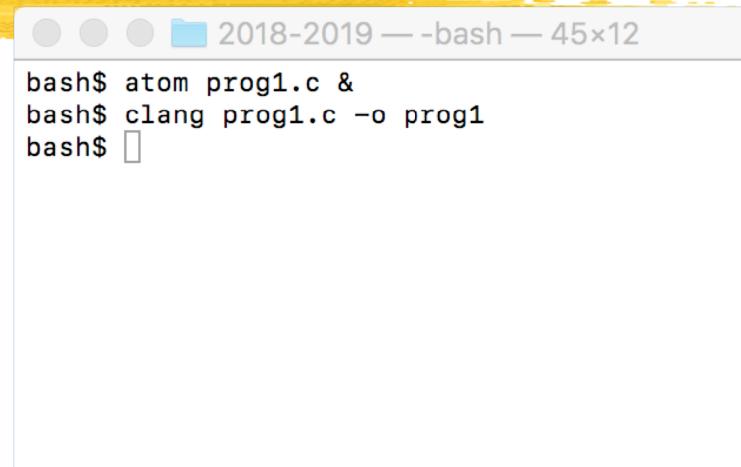
```
prog1.c
1 #include <stdio.h>
2
3 int main(){ double x,y; /* Créer deux nombres réels appelés x et y */
4     printf("Entrer un nombre au clavier\n"); /*Affiche un message à l'écran*/
5     scanf("%lf", &x); /* Entrer un nombre au clavier, le mettre dans x */
6     y = 1/x; /* Calculer l'inverse, le mettre dans y */
7     printf("L'inverse de %lf est : %lf \n ",x,y); /* Afficher le résultat */
8     return 0;
9 }
10
```

prog1.c 1:1 LF UTF-8 C 0 files

Exemple

- Créer le fichier exécutable contenant le programme. On utilisera cette année le compilateur *clang* :

- bash-3.00\$ clang prog1.c -o prog1



```
bash$ atom prog1.c &
bash$ clang prog1.c -o prog1
bash$ 
```

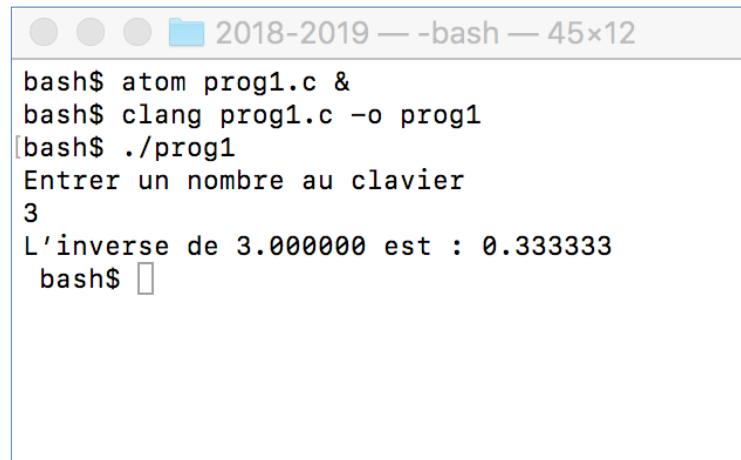
- Exécuter le programme

- bash-3.00\$./prog1

Entrer un nombre au clavier

3

L'inverse de 3 est : 0.333333



```
bash$ atom prog1.c &
bash$ clang prog1.c -o prog1
[bash$ ./prog1
Entrer un nombre au clavier
3
L'inverse de 3.000000 est : 0.333333
bash$ ]
```

Un éditeur de texte



- Un éditeur est un programme qui sert à taper, modifier, sauver du texte dans un fichier
- De nombreux éditeurs existent : vim , gedit, atom, xemacs.
 - Le plus simple : gedit
 - Le plus facile : atom
 - Le préféré des programmeurs libres : xemacs
 - Le plus vieux : vim

pouilly:~/Documents/ens/1AERG/2008/ex_desvignes\$ od -A n -t x
p1

Pourquoi compiler

- Voici le code binaire de l'addition de deux nombres.

- C'est un programme qui met 5 dans i, 8 dans j et i+j dans k.
- Taille du programme binaire : 16904 octets
- Facile, non !!!

- On utilise des langages plus proches de nous

feedface	00000012	00000000	00000002
0000000b	00000530	00000085	00000001
00000038	5f5f5041	47455a45	524f0000
00000000	00000000	00001000	00000000
00000000	00000000	00000000	00000000
00000004	00000001	0000018c	5f5f5445
58540000	00000000	00000000	00001000
00002000	00000000	00002000	00000007
00000005	00000005	00000000	5f5f7465
78740000	00000000	00000000	5f5f5445
58540000	00000000	00000000	000023ac
00000988	000013ac	00000002	00000000
00000000	80000400	00000000	00000000
5f5f7069	6373796d	626f6c5f	73747562
5f5f5445	58540000	00000000	00000000
00002d34	00000000	00001d34	00000002
00000000	00000000	80000008	00000000
00000024	5f5f7379	6d626f6c	5f737475
62000000	5f5f5445	58540000	00000000
00000000	00002d34	00000000	00001d34
00000002	00000000	00000000	80000008
00000000	00000014	5f5f7069	6373796d
626f6c73	74756231	5f5f5445	58540000
00000000	00000000	00002d40	00000160
00001d40	00000005	00000000	00000000
80000408	00000000	00000020	5f5f6373
7472696e	67000000	00000000	5f5f5445
58540000	00000000	00000000	00002ea0
00000130	00001ea0	00000002	00000000
00000000	00000002	00000000	00000000
00000001	0000018c	5f5f4441	54410000
00000000	00000000	00003000	00001000
00002000	00001000	00000007	00000003
00000005	00000000	5f5f6461	74610000
00000000	00000000	5f5f4441	54410000
00000000	00000000	00003000	00000014
00002000	00000002	00000000	00000000

Assembleur

```
.section __TEXT,__text,regular,pure_instructions
.section __TEXT,__picsymbolstub1,symbol_stubs,pure_instructions,32
.machine ppc
.text
.align 2
.globl _main

_main:
    stmw r30,-8(r1)
    stwu r1,-64(r1)
    mr r30,r1
    li r0,5          // Mettre 5 dans le registre r0
    stw r0,32(r30)   // stocker r0 dans i (adresse r30+32)
    li r0,8          // Mettre 8 dans le registre r0
    stw r0,28(r30)   // stocker r0 dans j (adresse r30+28)
    lwz r2,32(r30)   // Charger i (adresse r30+32) dans r2
    lwz r0,28(r30)   // Charger j (adresse r30+28) dans r0
    add r0,r2,r0     // Additionner r0 et r2 dans r0
    stw r0,24(r30)   // stocker r0 dans k (adresse r30+24)
    lwz r1,0(r1)
    lmw r30,-8(r1)
    blr
.subsections_via_symbols
```

En C

```
#include <stdio.h>

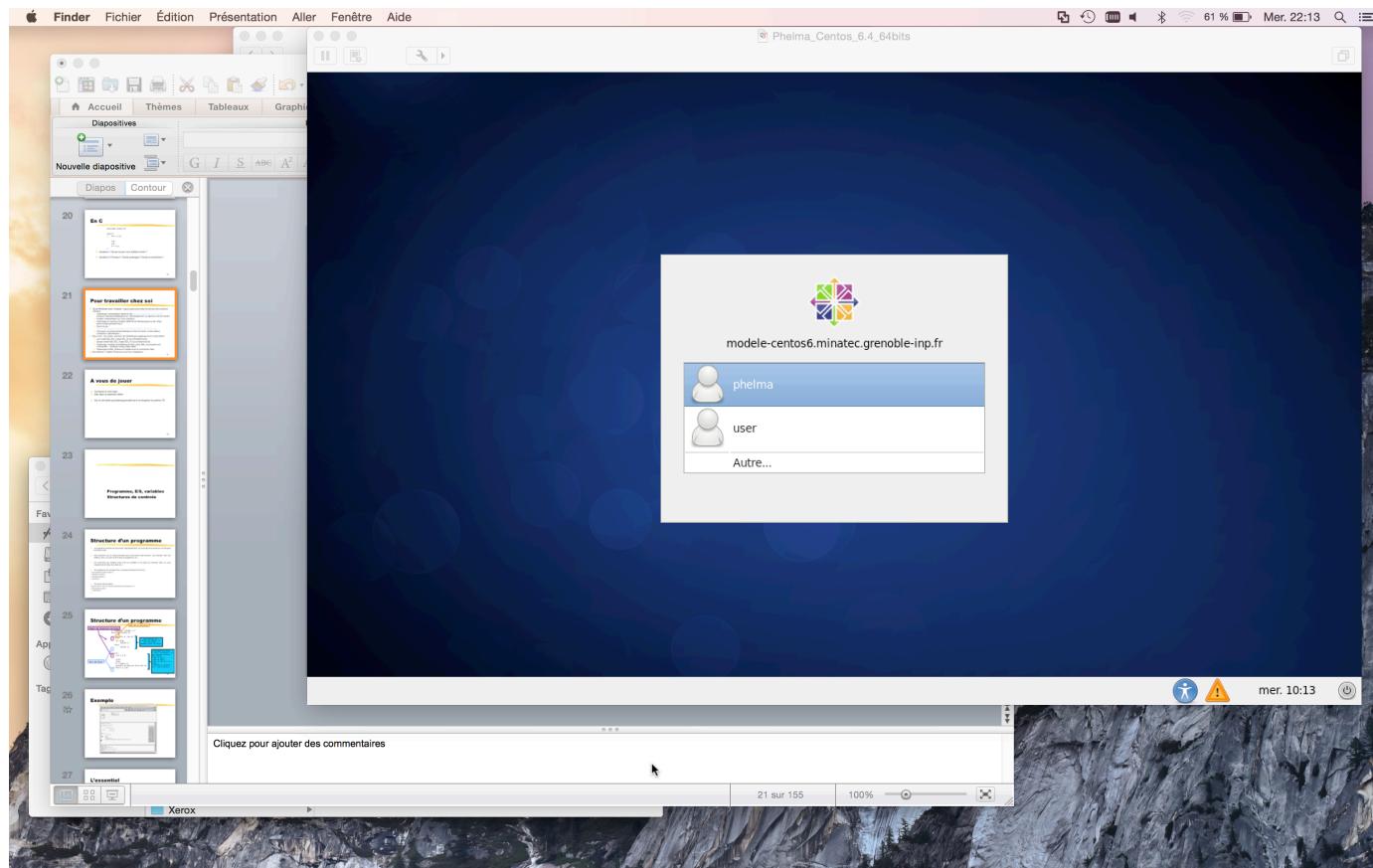
main()
{    int i,j,k;

    i=5;
    j=8;
    k = i+j;
}
```

- Question 1: Qu'est ce que vous préférez écrire ?
- Question 2: Pourquoi ? Quels avantages ? Quels inconvénients ?

Pour travailler chez soi

- Sous Windows avec Vmware : logiciel gratuit permettant d'exécuter des machines virtuelles



Pour travailler chez soi

■ Sous Windows avec VirtualBox : logiciel gratuit permettant d'exécuter des machines virtuelles

- Installer VirtualBox sur votre ordinateur
- Télécharger la machine virtuelle CENTOS de Phelma depuis le site <http://tdinfo.phelma.grenoble-inp.fr>
- Vous avez un environnement identique à celui de l'école : même éditeur, compilateur, bibliothèques....

■ Utiliser le Bash Ubuntu sous Windows 10

■ Sous Linux : rien à faire, sauf pour les bibliothèques graphiques (voir le site tdinfo)

- yum install SDL SDL_image SDL_ttf sous RedHat/Centos
- apt-get install SDL SDL_image SDL_ttf sous Debian/Ubuntu
- Télécharger, Installer les bibliothèques SDL_draw, SDL_sound grâce aux commandes : ./configure; make; make install
- Télécharger la SDL_phelma et l'installer avec la commande make

■ Sous Macosx : installer Xcode pour avoir les compilateurs, puis procéder comme pour linux

A vous de jouer



- Connexion à votre login
- Aller dans le répertoire tdinfo

- Voir le site tdinfo.phelma.grenoble-inp.fr et récupérer le premier TD

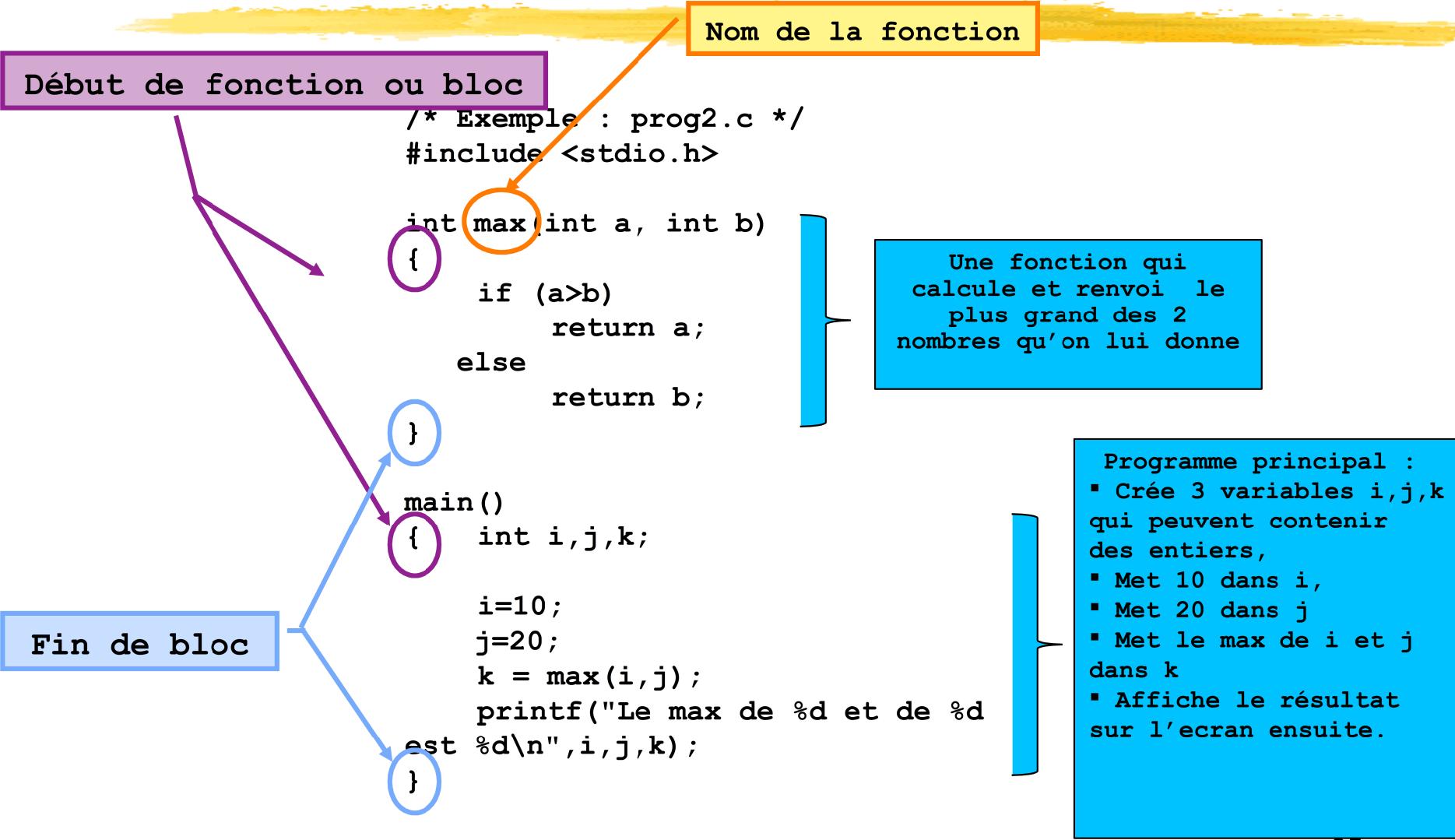


Programme, E/S, variables Structures de contrôle

Structure d'un programme

- Le programme exécute les instructions séquentiellement, les unes après les autres en commençant par la fonction main
- Une instruction est un ordre/commande que le processeur sait exécuter : par exemple, faire une addition, aller à un autre endroit dans le programme, etc...
- On commence par déclarer quels sont les variables et les types de données utiles: on peut manipuler des entiers, des réels, etc...
- Un programme est composé d'un ou plusieurs fichiers de la forme :
< Commandes du préprocesseur >
< Définition de types >
< Variables globales >
< Fonctions >
- Structures des fonctions :
<type de retour> nom_de_fonction(<Déclaration des arguments >)
{ <Déclarations locales>
 <instructions>
}

Structure d'un programme



Exemple

The screenshot shows the DDD (Debian Debug) interface running a C program named `prog2.c`. The program code is as follows:

```
#include <stdio.h>

int max(int a, int b)
{
    if (a>b)
        return a;
    else
        return b;
}

main()
{
    int i,j,k;
    i=10;
    j=20;
    k = max(i,j);
    printf("Le max de %d et de %d est %d\n",i,j,k);
}
```

The `Locals` window shows variable values: `i = 10`, `j = 20`, and `k = 20`. The `Args` window shows "No arguments.". The bottom window displays the GDB command history and output:

```
(gdb) step
main () at prog2.c:18
(gdb) step
Le max de 10 et de 20 est 20
(gdb) I
```

A status message at the bottom says: `Updating status displays...done.`

L'essentiel



- Qu'est ce qu'un programme informatique manipule ?
 - Des nombres, des mots
- Pour manipuler des nombres, il faut pouvoir les stocker dans des variables.
- Il faut pouvoir introduire les nombres dans le programme et le programme doit pouvoir les afficher
- Le programme doit exécuter les ordres : instructions.
 - Une instruction se termine par un point virgule
 - Les instructions sont incluses dans des blocs { }
 - Les blocs sont inclus dans les fonctions

Types de nombre

Entier

- **char** : Octet : 1 octet compris entre -128 et +127
- **int** : Entier (1 mot machine)
 - **short** : entier Court
 - **long** : entier Long
 - **long long** : entier Long
 - **unsigned int** : entier Non Signe
 - **unsigned long** : entier Long Non Signe
- **short int <= int <= long int**

Réels

- **float** : Réel flottant Simple Précision (4 octets) :
- **double** : Réel flottant Double Précision (8 octets)
- **long double** : Réel flottant quadruple Précision (16 octets)

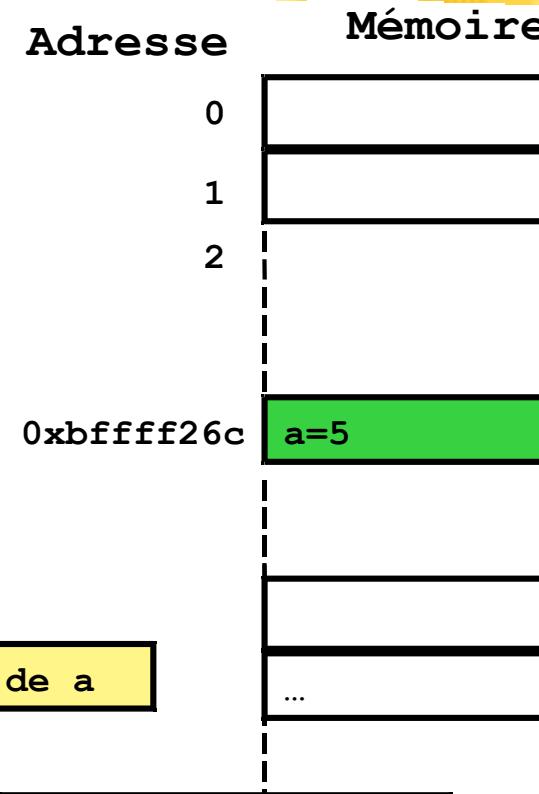
PAS de booleens

- FAUX = 0 VRAI = NON ZERO

Notion de variable

- Objet informatique permettant de conserver et de modifier sa valeur
 - Un type : entier, réel, etc...
 - Un nom
 - Une adresse : où est elle ? un entier qui est le numéro de la case mémoire où elle se trouve
 - Sa durée de vie (ou portée) : de l'endroit où elle est déclarée à la fin du bloc : }
- Exemple : var1.c
 - La variable a est créée à l'adresse 3221221990 en décimal, soit 0xbffff26c en hexadécimal

```
#include <stdio.h>
main() {
    int a;
    a=5;
    printf("Bonjour\n");
    printf("La valeur de a est %d\n",a);
    printf("L'adresse de a est %p\n",&a);
}
```



Création de a

&a: adresse de a

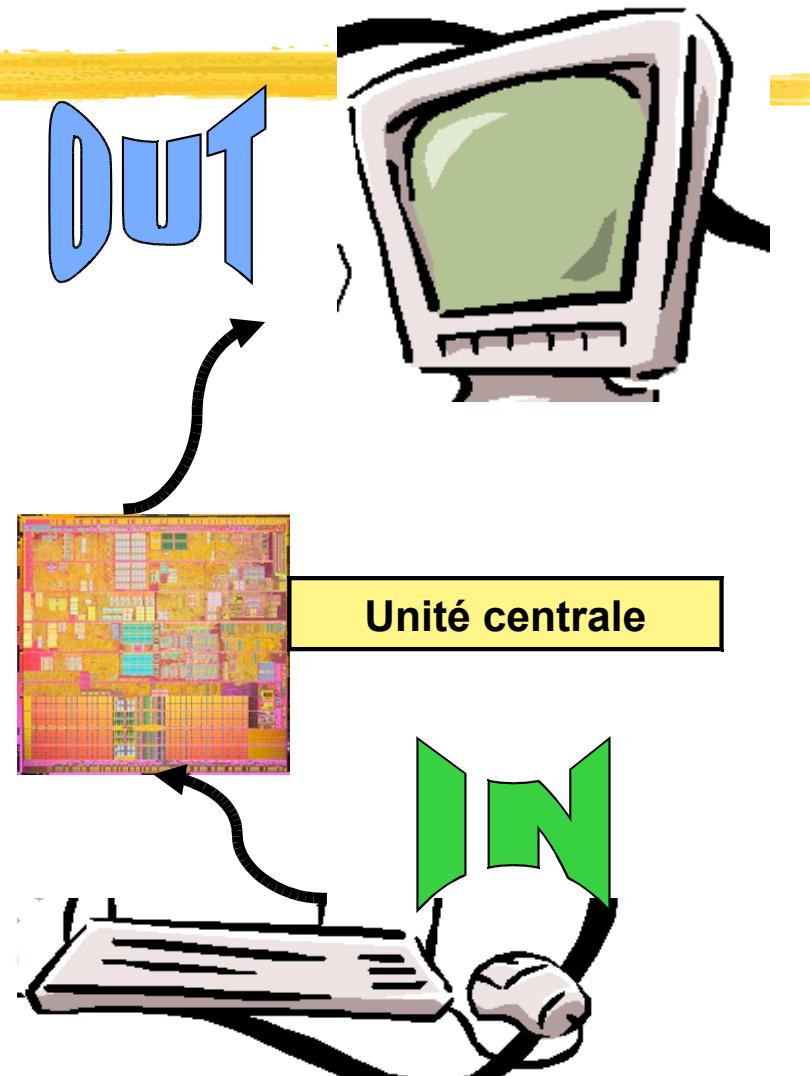
Destruction de a



Entrées/Sorties

Entrées/sorties

- Opérations permettant la communication entre le programme et l'extérieur
- Unité centrale : egocentrique
- Sortie
 - Exportation d'une valeur vers l'extérieur (écriture) à travers un périphérique de sortie : écran, réseau , fichier, port série ou parallèle, pipe.....
- Entrée
 - introduction d'une valeur à l'intérieur du programme (lecture) à travers un périphérique d'entrée : clavier, réseau, fichier, port série ou parallèle, pipe.....



Comment afficher : printf

- On utilise des fonctions déjà écrites : il faut inclure un fichier d'entête au début du fichier

#include <stdio.h>

Aller à la ligne

- Afficher un message :

printf("Mon message\n")

- Afficher un nombre :

printf("Mon entier i : %d\n", i);

Un message optionel

%d : le symbole % indique que la lettre suivante est le type du nombre à afficher

Le nombre, la variable ou l'expression à afficher

printf : syntaxe

- | Syntaxe : Affichage d'un type de base

```
int printf(const char * format, variable1, variable2, ...);
```

format : %[*][largeur] [.precision] [modif type] type_carac

- | spécifier le type du nombre à afficher (type_carac) par
 - | %d : nombre entier
 - | %c : un caractère
 - | %lf : un réel double précision
 - | %f : un réel simple précision
- | En option, on peut préciser le nombre de chiffres que l'on souhaite pour l'affichage par le nombre largeur et le nombre de chiffre après la virgule pour un réel par le nombre précision

printf : exemples

Exemple : affiche0.c

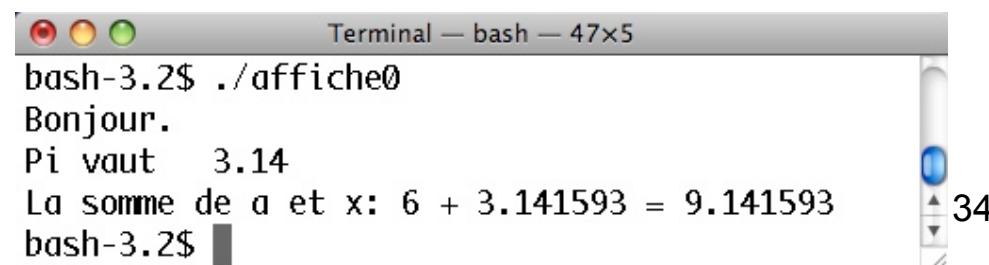
```
#include <stdio.h>
main() { double x; int a;
    x = 3.1415927;
    a=6;
        /* Afficher un message */
    printf("Bonjour.\n");
        /* Afficher un reel avec 6 chiffres dont 2 apres la virgule */
    printf("Pi vaut %6.2lf\n",x);
        /* Afficher plusieurs nombres entier ou reels */
    printf("La somme de a et x: %d + %lf vaut %lf\n",a,x,a+x); }
```

Un message facultatif

Le premier nombre affiché est l'entier a

Le deuxième nombre affiché est le réel x

Le troisième nombre affiché est la somme a+x



printf : le piège

Exemple : affiche0a.c

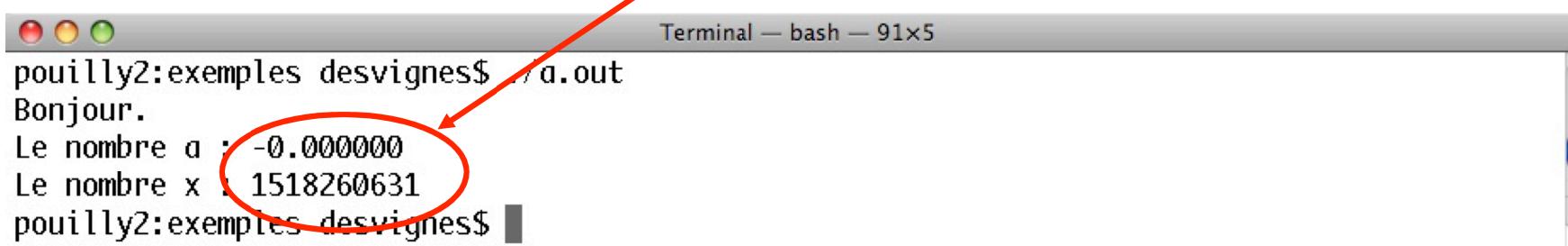
```
#include <stdio.h>
main() { double x; int a;
  x = 3.1415927;
  a=6;
  /* Afficher un message */
  printf("Bonjour.\n");
  printf("Le nombre a : %lf\n", a);
  printf("Le nombre x : %d\n", x);
}
```

Erreur sur le type des nombres à afficher

- `a` est un entier, le format demandé (`%lf`) est celui des réels
- idem pour `x`

==>

le programme est syntaxiquement correct, la compilation est correcte, mais à l'exécution du programme, l'AFFICHAGE sera incohérent



```
pouilly2:exemples desvignes$ ./a.out
Bonjour.
Le nombre a : -0.000000
Le nombre x : 1518260631
pouilly2:exemples desvignes$
```

Lire au clavier: scanf

- On utilise des fonctions déjà écrites : il faut inclure un fichier d'entête au début du fichier

- #include <stdio.h>

ATTENTION au symbole &

- Lire un nombre au clavier :

```
scanf("%d", &i);
```

Aucun message

L'adresse de la variable à lire

- spécifier le type du nombre à lire dans le format

- %d : nombre entier
- %c : un caractère
- %lf : un réel double précision
- %f : un réel simple précision

%d : le type de i

- Spécifier ensuite le nombre à lire

- Par l'adresse d'une variable

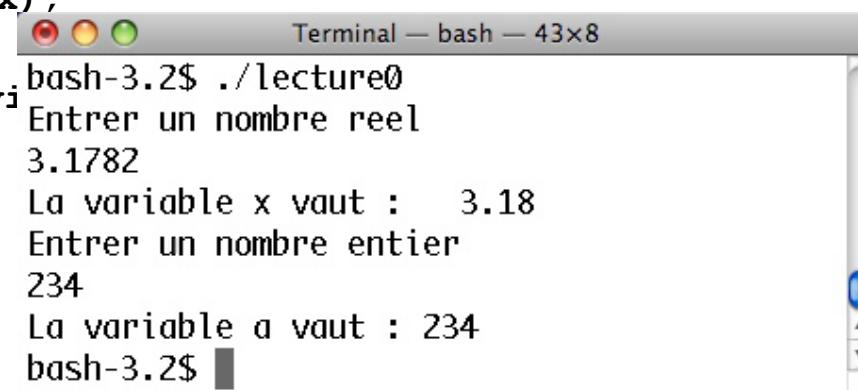
scanf : exemple

- Syntaxe : lecture d 'un type de base

```
int scanf(const char * format, adresse_variable1, ...);  
format : %[*][largeur] [.precision] [modif type] type_carac
```

- Exemple : lecture0.c

```
#include <stdio.h>  
main() { double x; int a;  
    /* Afficher un message pour demander les nombres a l'utilisateur*/  
    printf("Entrer un nombre reel \n");  
    /* Il faut taper un reel au clavier */  
    scanf("%lf", &x);  
    /* Afficher un reel avec 6 chiffres dont 2 apres la virgule */  
    printf("La variable x vaut : %6.2lf\n",x);  
    printf("Entrer un nombre entier \n");  
    /* Il faut taper un entier au clavier */  
    scanf("%d", &a);  
    /* Afficher un entier */  
    printf("La variable a vaut : %d\n",a);  
}
```



scanf : lire plusieurs nombres en une fois

Exemple : lecture1.c

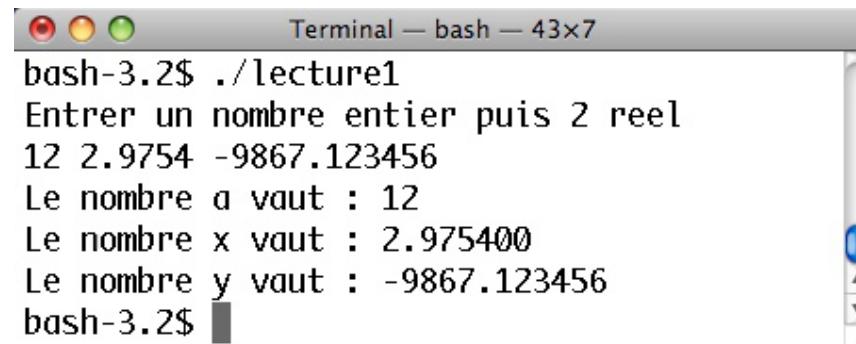
```
#include <stdio.h>
main() { double x,y; int a;
    /* Afficher un message pour demander les nombres a l'utilisateur*/
printf("Entrer un nombre entier puis 2 reel\n");
    /* lire un nombre entier puis 2 reels */
```

scanf('%d %lf %lf', &a, &x, &y);

deuxième nombre : le réel x

Le premier nombre tapé au clavier sera stocké dans la variable entière a

```
printf("Le nombre a vaut : %d\n",a);
printf("Le nombre x vaut : %lf\n",x);
printf("Le nombre y vaut : %lf\n",y);
}
```



scanf: les pièges

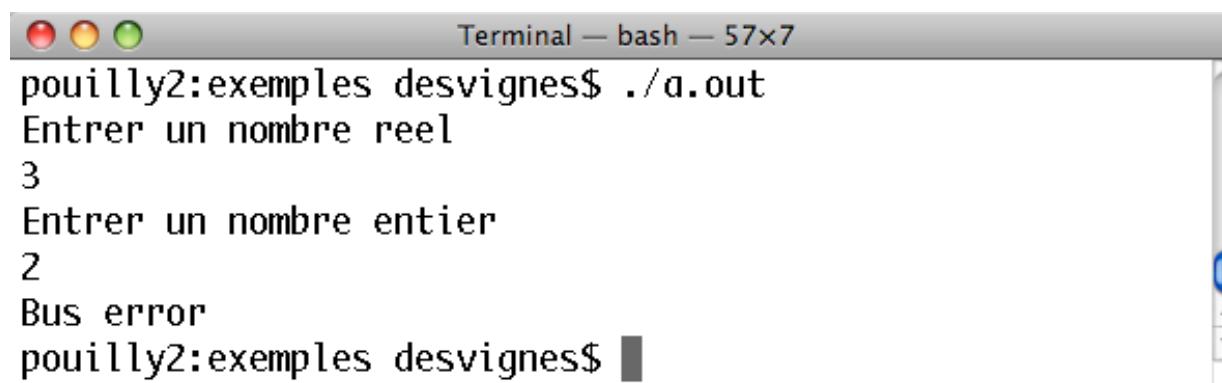
Exemple : lecture0a.c

```
#include <stdio.h>
main() { double x; int a;
    printf("Entrer un nombre reel \n");
    scanf("%d", &x);

    printf("Entrer un nombre entier \n");
    scanf("%d", a);
    /* Afficher un entier */
    printf("La variable a vaut : %d\n",a);
}
```

Erreur sur le type des nombres à lire ==> Valeur incohérente

Oubli de l'adresse ==>
Valeur incohérente ou arret du programme



The screenshot shows a terminal window titled "Terminal — bash — 57x7". The command "pouilly2:exemples desvignes\$./a.out" is run. The user is prompted to "Entrer un nombre reel" and enters "3". Then, the user is prompted to "Entrer un nombre entier" and enters "2". The program then outputs "Bus error".

```
pouilly2:exemples desvignes$ ./a.out
Entrer un nombre reel
3
Entrer un nombre entier
2
Bus error
pouilly2:exemples desvignes$
```

Exercices :



- Exo 1 : Calculer et afficher la somme de 3 nombres réels lus au clavier
- Exo 2 : Calculer et afficher la moyenne de 3 nombres entiers lus au clavier
- Exo 3 : Calculer et afficher le quotient de 2 nombres réels lus au clavier. Pourquoi, en utilisant uniquement ce qui vient d'être vu, ce programme n'est-il pas correct ?

Les structures de contrôles

Les structures de contrôles



■ Structures de contrôle

- Ruptures au sein du déroulement séquentiel du programme
- Deux types

- Conditionnelles : faire une action si un test est vérifié
 - Répétitives : répéter une action un certain nombre de fois

- Mots clés :

`if, else, switch, case, do..while, while, for`

Conditionnelle : if

- Une instruction conditionnelle simple permet de réaliser une ou plusieurs actions selon **un** test sur **une** expression

Syntaxe

```
if ( expression ){   instruction1; }
[else if (expression) {instruction2;} ]
[else {instruction2;} ]
```

Signification

- SI expression est vraie (`!= 0`) , instruction1 est exécutée
- sinon instruction2 est exécutée

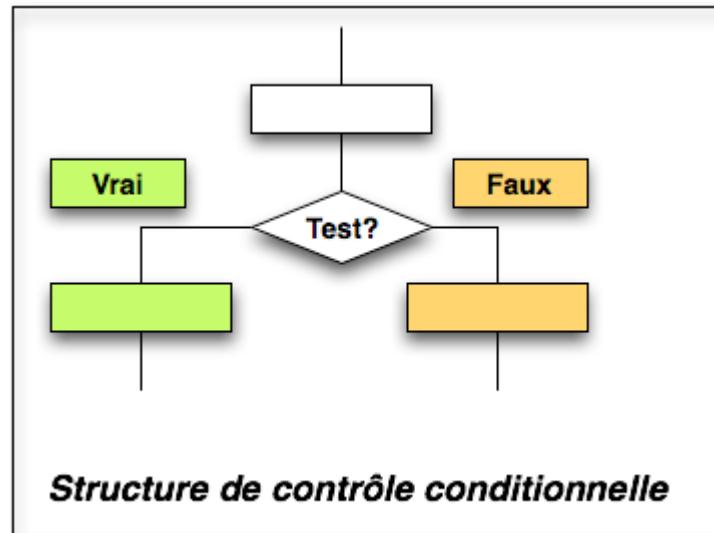
La clause else instructions2; est optionnelle

Clauses if et else

- Une clause comporte une seule instruction ou plusieurs instructions encadrées par `{.....}` qui définissent un bloc

Imbrication de plusieurs if ... else

- clause else se rapporte au if le plus proche



Exemple

- Exemple : if1.c : on affiche le plus grand des 2 nombres
- Les 2 clauses if et else comportent chacune une seule instruction

```
#include <stdio.h>
main() { int a,b;
printf("Entrer 2 entiers\n");
scanf("%d %d",&a,&b);
if (a>b)
    printf("a:%d est plus grand que b%d\n",a,b);
else
    printf("b:%d est plus grand que a%d\n",b,a);
printf( "Fin du programme\n");
}
```

Exemple 2

- Exemple : if2.c : on calcule en plus la différence absolue entre les 2 nombres
- Les clauses if et else comportent plusieurs instructions DANS un bloc

```
#include <math.h>
#include <stdio.h>
main() { int a,b,c;
    printf("Entrer 2 entiers\n"); scanf("%d %d", &a, &b);
    if (a>b) {
        c=a-b;
        printf("a:%d est plus grand que b:%d\n",a,b);
    }
    else {
        c=b-a;
        printf("b:%d est plus grand que a:%d\n",b,a);
    }
    printf("la racine carree de la difference absolue vaut %lf\n",sqrt(c));
}
```

```
Entrer 2 entiers
2 89
b:89 est plus grand que a:2
la racine carree de la difference absolue vaut 9.327379
bash-3.2$ ./a.out
Entrer 2 entiers
67 4
a:67 est plus grand que b:4
la racine carree de la difference absolue vaut 7.937254
bash-3.2$
```

Plusieurs instructions :
il faut un bloc entre { }

Un piège

- Exemple : if3.c : on teste si 2 nombres sont égaux
- Attention :
 - l'opérateur = affecte la valeur de droite dans la variable de gauche
 - l'opérateur == teste si les 2 expressions sont égales

```
#include <stdio.h>
main() { int a,b;
    printf("Entrer 2 entiers\n");
    scanf("%d %d",&a,&b);
    if (a=b) ←
        printf("a:%d est egal a b%d\n",a,b);
    else
        printf("b:%d n'est pas egal a%d\n",b,a);
    printf( "Fin du programme\n");
}
```

Ici, on met la valeur de b dans a, puis on teste si ce résultat est vrai (non nul) ou faux (nul)

Si on veut simplement tester si a et b sont égaux, il faut utiliser

`if (a == b)`

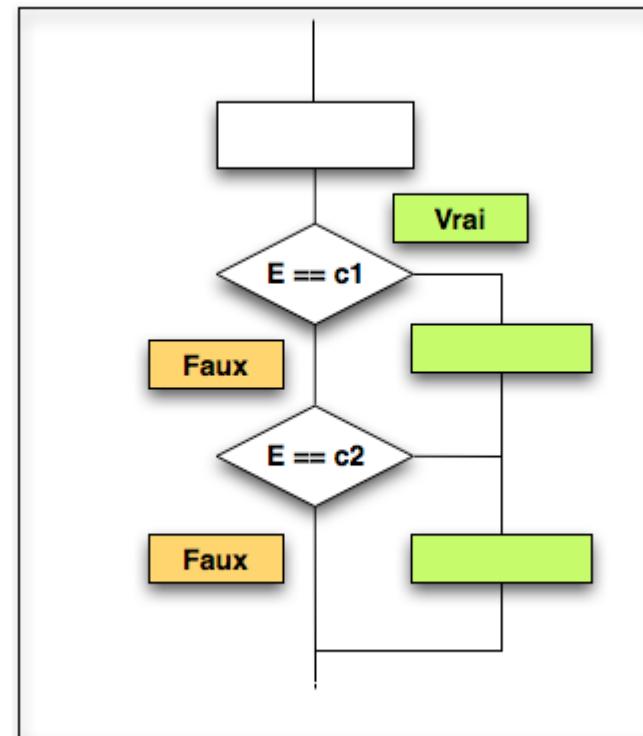
Structure conditionnelle multiple

■ Une instruction conditionnelle multiple permet de réaliser une ou plusieurs actions selon **un ou plusieurs tests séquentiels** sur **une expression**

- → positionner d'abord les cas les plus courants (réalisés le plus rapidement)

■ Syntaxe

```
switch ( expression ) {  
    case constante1 : instructionS1;  
    case constante2 : instructionS2;  
    ....  
    [ default : instructionS; ]  
}  
/* Fin du switch */
```



Menu

Exemple 2 : switch2.c

```
main() { char i; int cout;
printf ( "Au menu : \n") ;
printf ( "\t 1 : Vin rouge\n" ) ;
printf ( "\t 2 : Vin blanc \n" ) ;
printf ( "\t 3 : Biere \n" ) ;
printf ( "\t 4 : Eau\n" ) ;
printf ( "\t 5 : Rien \n" ) ;
printf ( "Tapez votre choix " );
i= getchar(); /* Lecture clavier de la reponse = scanf("%c",&i); */
printf("Vous avez choisi :");
switch ( i ) {
    case '1' : printf("un rouge"); cout=100; break;
    case '2' : printf("un blanc"); cout=60; break;
    case '3' : printf("une biere"); cout=80; break;
    case '4' : printf("de l'eau"); cout=10; break;
    case '5' : printf("pas soif"); cout=0; break;
    default : printf("Choix non valide"); cout=0; break;
}
printf(" pour un cout de %d\n",cout);
```

```
Terminal — bash — 64x19
macbook-pro-de-administrateur:exemples desvignes$ ./a.out
Au menu :
1 : Vin rouge
2 : Vin blanc
3 : Biere
4 : Eau
5 : Rien
Tapez votre choix 2
Vous avez choisi :un blanc pour un cout de 60
macbook-pro-de-administrateur:exemples desvignes$ ./a.out
Au menu :
1 : Vin rouge
2 : Vin blanc
3 : Biere
4 : Eau
5 : Rien
Tapez votre choix 8
Vous avez choisi :Choix non valide pour un cout de 0
macbook-pro-de-administrateur:exemples desvignes$
```

Quitte le cas '1'
et passe à la fin
du switch

Structure répétitive



- Une instruction répétitive est une construction permettant de répéter en séquence une ou plusieurs actions selon un test d'arrêt donné.
- 2 instructions
 - **tant que test_vrai répéter instructions;**
 - On vérifie d'abord qu'on a le droit de faire les instructions, et on execute instructions tant que le test est vrai.
 - **répéter instructions tant que test_vrai ;**
 - On fait d'abord les instructions puis on vérifie la condition. Si elle est vraie, on recommence, sinon, on arrete.

Structure répétitive : do .. while

■ Faire au moins une fois qqchose

■ Syntaxe

```
do {instructions;} while (expression);  
    instruction est exécutée tant que l'expression est vraie ( $\neq 0$ ),
```

■ Exemple1 : do1.c : vérifie les valeurs d'un utilisateur

```
main() { int i;  
    printf ( "Au menu : " );  
    do {  
        printf ( "1 : Vin rouge\n" );  
        printf ( "2 : Vin blanc\n" );  
        printf ( "3 : Biere \n" );  
        printf ( "4 : Eau\n" );  
        printf ( "5 : Rien \n" );  
        printf ( "Tapez votre choix " );  
        scanf("%d",&i); /* Lecture au clavier */  
    } while (i<=0 || i>=6);  
    printf("Mon choix est: %d \n",i);  
}
```

OU
logique

Réponses de
l'utilisateur

```
bash-3.2$ gcc do1.c  
bash-3.2$ ./a.out  
Au menu :  
1 : Vin rouge  
2 : Vin blanc  
3 : Biere  
4 : Eau  
5 : Rien  
Tapez votre choix 0  
Au menu :  
1 : Vin rouge  
2 : Vin blanc  
3 : Biere  
4 : Eau  
5 : Rien  
Tapez votre choix 9  
Au menu :  
1 : Vin rouge  
2 : Vin blanc  
3 : Biere  
4 : Eau  
5 : Rien  
Tapez votre choix 1  
Mon choix est : 1  
bash-3.2$
```

Structure répétitive : while

■ Faire plusieurs fois qqchose

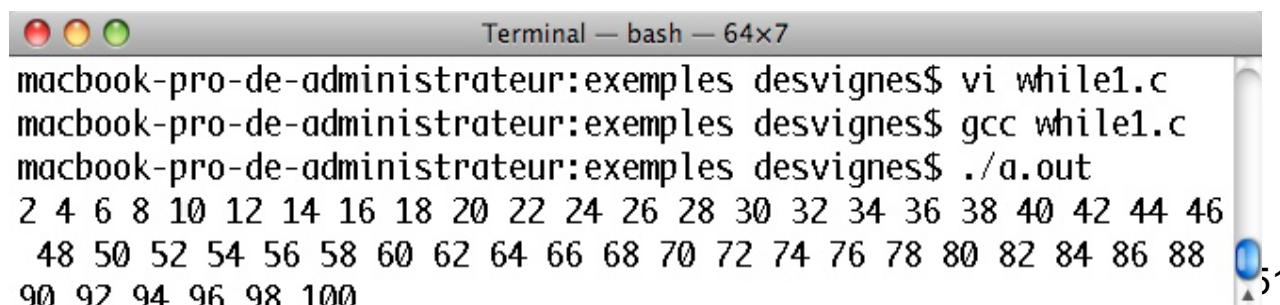
■ Syntaxe

```
while (expression) {instructions;}  
    | tant que expression est vraie ( != 0 ), instruction est exécutée
```

■ Exemple1 : while1.c : Compte de 2 en 2 jusqu'à 100 et affiche le nombre

```
main() { int i=0;  
    while (i<100) {  
        i = i + 2; /* On peut écrire i += 2; en C */  
        printf("%d ",i); /* On affiche i */  
    }  
    puts("");  
}
```

■ Quels sont le premier et le dernier nombre affichés ?



```
macbook-pro-de-administrateur:exemples desvignes$ vi while1.c  
macbook-pro-de-administrateur:exemples desvignes$ gcc while1.c  
macbook-pro-de-administrateur:exemples desvignes$ ./a.out  
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46  
48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88  
90 92 94 96 98 100
```

Structure répétitive : for

Boucle FOR

- instruction répétitive la plus courante
- localise les parties en les dissociant du reste du code :
 - initialisation (expr1),
 - test d'arrêt de la boucle (expr2)
 - passage à l'itération suivante (expr3)
- ne nécessite pas de connaître à l'avance le nombre d'itérations

Syntaxe

```
for(exp1; exp2; exp3;) instruction;
```

Identique à

```
expr1;  
while ( expr2 ) {  
    instruction;  
    (expr3);  
}
```

Que font ces deux exemples ?

Exemple1 : for1.c

```
main() {  
    int i;  
    for (i=0; i < 100; i++)  
        printf("%d ",i);  
    printf("\n");  
}
```

Exemple2 : for2.c

```
main() { int i; char c='a';  
puts("taper votre texte, finir par q");  
for (i=0; c!= 'q'; i++) {  
/* Lecture d'un caractère */  
c=getchar();  
printf("caractere lu : %c\n",c);  
}  
printf("La valeur de i est %d\n",i);  
}
```

Rupture de séquence



- Instruction **continue**
 - Passe à l'itération suivante d'une boucle
- Instruction **break**
 - Quitte une boucle ou un switch
- Instruction **return**
 - Quitte une fonction et retourne la valeur passée en paramètre, sort du programme si utilisée dans un **main**.
- Instruction **exit (n)**
 - Sortie d'un programme et retour au système d'exploitation. La valeur de retour est n

Exercices :

- Exo 1: Faire un programme qui affiche la valeur absolue d'un réel lu au clavier.
- Exo 2: Calculer la somme des N premiers entiers et vérifier qu'elle fait $N*(N+1)/2$. N est lu au clavier.
- Exo 3: Calculer et afficher les 100 premiers termes de la suite $U_n = U_{n-1} + 2$
- Exo 4 : Calculer Pi par la limite de
$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \dots$$

Les nombres



Représentation des nombres entiers

Notion de bit, octet, mot machine

- bit : 1 unité d'information : 1 ou 0
- octet : 8 bits

$2^7 \quad 2^6 \quad 2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$
b7 b6 b5 b4 b3 b2 b1 b0

$$A = b_7 * 2^7 + b_6 * 2^6 + b_5 * 2^5 + b_4 * 2^4 + b_3 * 2^3 + b_2 * 2^2 + b_1 * 2^1 + b_0 * 2^0$$

Nombres positifs : décomposition en puissance de 2

Nombres négatifs : complément à 2

- Complémenter le nombre positif (0 => 1 et 1 => 0) et Ajouter 1

Nombre	Représentation binaire
0	00000000
1	00000001
2	00000010
3	00000011
127	01111111

Nombre	Représentation binaire
0	00000000
-1	11111111
-2	11111110
-3	11111101

Nombres entiers

- 1 octet : 8 bits
 - | **unsigned char** : Nombres non signés : 0.. $255=2^8-1$
 - | **char** : Nombres signés : bit 7 = bit de signe : $-128=-2^7..127=2^7-1$
- 2 octets : 16 bits
 - | **unsigned short** : Nombres non signés : 0.. $65535=2^{16}-1$
 - | **short** : Nombres signés : bit 15 = bit de signe : $-32768=-2^{15}..32767=2^{15}-1$
- 4 octets : 32 bits
 - | **unsigned long** : Nombres non signés : 0.. $4294967296=2^{32}-1$
 - | **long** : Nombres signés : bit 31 = bit de signe
 - | : $-2147483648=-2^{31}.21474836487=2^{31}-1$
- 8 octets : 64 bits
 - | **unsigned long long** : Nombres non signés : 0.. $18446744073709551615 =2^{64}-1$
 - | **long** : Nombres signés : bit 63 = bit de signe
 - | : $-9223372036854775808=-2^{63}..9223372036854775808 =2^{63}-1$
- Conséquences sur l'Arithmétique entière
 - | Pas d'erreur de calcul : $i+1$ est toujours exact sauf quand on dépasse les limites
 - | Domaine de validité limité :
 - | Exemple en **char** : $127 + 1 = -128; 127+2=-127; 127+3=-126....$

Nombres réels

- Norme IEEE 754 : Spécifie un modèle arithmétique complet pour les réels
- Format général

Signe : s Exposant : e Mantise : m

Valeur = s x m x $2^{e-\text{décalage}}$

- Représentation binaire finie :

- Nombres de réels FINI !!!
- L'infini n'existe pas
- L'écart entre 2 réels consécutifs est relatif

X	plus proche suivant	Différence
1.0	1.0000001	1.19 10 ⁻⁷
2.0	2.0000002	2.38 10 ⁻⁷
16.0	16.000002	1.9 10 ⁻⁶
128.0	128.00002	1,52 10 ⁻⁵

- Conséquences : la Troncature : il existe toujours une erreur de calcul

- L'addition et la multiplication ne sont plus exactement associatives
 - $(3.11 * 2.30) * 1.50 = 10.729500000000000$
 - $3.11 * (2.30 * 1.50) = 10.729499999999998$
- La multiplication n'est plus distributive
- Attention : les erreurs se cumulent

Précision et limites des réels



- Réels simple précision : 32 bits : float
 - Mantisse sur 23 bits, exposant sur 8 bits
 - Max : +/- 2^{128} soit +/- 3×10^{38} Min : +/- 2^{-150} soit +/- 1×10^{-45}
 - Précision : 2^{-23} soit +/- 1×10^{-7}
- Réels double précision : 64 bits : double
 - Mantisse sur 52 bits, exposant sur 11 bits
 - Max : +/- 2^{1024} soit +/- 1×10^{308} Min : +/- 2^{-1075} soit +/- 1×10^{-324}
 - Précision : 2^{-52} soit +/- 1×10^{-16}
- Réels quadruple précision : 128 bits (80 bits sur intel) : long double
 - Mantisse sur 112 bits (64 bits sur intel), exposant sur 15 bits
 - Max : +/- 2^{16383} soit +/- 3×10^{4932} Min : +/- 1×10^{-4966}
 - Précision : 2^{-112} soit +/- 1×10^{-34} (2^{-64} soit +/- 1×10^{-19} sur intel)
- Conclusion sur l'arithmétique réelle
 - Les réels ont un domaine de variation limités (mais plus grand que les entiers)
 - Les calculs en réel ont une précision très limitée, mais suffisante dans la plupart des cas

Calcul flottant : exemple

Exemple : /* Fichier cunu2.c */

Ajouter un réel à 1 pour visualiser l'erreur de calcul

```
#include <stdio.h>
main() { float x,y;
    printf("Entrer le nombre réel a ajouter a 1 "); scanf("%f", &x);
    y = 1/ ( (1+x) - 1 );
    printf("Valeur de x:%.15f et de 1+x:%.15f\n",x,1+x);
    printf("Valeur de 1+x-1:%.15f et de y:%.15f\n", (1+x)-1,y);
}
```

Valeur de y exacte à 10-5 pres

Plus proche réel de
0.0001. Ce n'est
pas exactement 10-4

Troncature car x
est trop petit

```
Terminal — bash — 64x9
bash-3.2$ ./a.out
Entrer le nombre réel a ajouter a 1 : 0.0001
Valeur de x:0.000099999997474 et de 1+x:1.000100016593933
Valeur de 1+x-1:0.000100016593933 et de y:9998.340820312500000
bash-3.2$ ./a.out
Entrer le nombre réel a ajouter a 1 : 0.00000001
Valeur de x:0.000000010000000 et de 1+x:1.000000000000000
Valeur de 1+x-1:0.000000000000000 et de y:inf
```

Valeur de y inexacte

Calcul flottant : exemple

- Cumul des erreurs et non associativité
- Exemple : /* Fichier cunu2c.c et cunu2d.c */
- Calcul des 80 premiers termes de $X_n = (R+1)X_{n-1} - R*X_{n-1}^2 X_{n-1}$
 - par $X = (R+1)*X - (R*X)*X;$ dans cunu2c avec $R=3$ et $X_0=0,5$
 - Par $X = (R+1)*X - R*(X*X);$ dans cunu2d
- Affichage des termes 0, 10, 20 ,30 40, 50,60,70 et 80, calcul en double

The screenshot shows a terminal window titled "Terminal — bash — 80x5". The window contains the following text:

```
pouilly2:exemples desvignes$ ./cunu2d
0.500000 0.384631 0.418895 0.046399 0.320177 0.067567 0.001145 1.296775 0.553038
pouilly2:exemples desvignes$ ./cunu2c
0.500000 0.384631 0.418895 0.046399 0.320184 0.063747 0.271115 1.328462 0.817163
pouilly2:exemples desvignes$
```

Représentation des Caractères : code à skis



Code ascii

unités

Dizaines

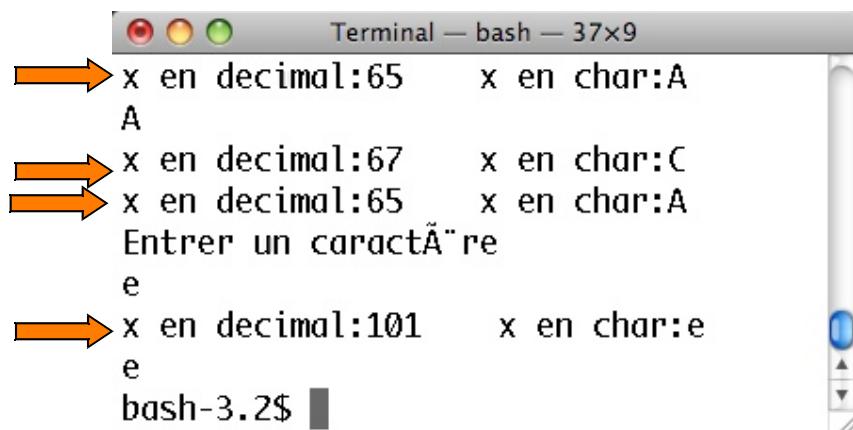
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STH	ETH	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	CD2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	:	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	-
6	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	}	~	DEL	
8	€	□	,	f	"	„	†	‡	^	%o	Š	<	Œ	□	Ž	□
9	□	'	'	"	"	•	-	—	“	™	š	>	œ	□	ž	ÿ
A	í	¢	£	¤	¥	ƒ	§	“	”	©	ª	«	¬	-	®	-
B	°	±	²	³	‘	µ	¶	·	,	¹	º	»	¼	½	¾	½
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Ø	Ù	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	ø	ù	ý	þ	ÿ

Caractères : code ascii

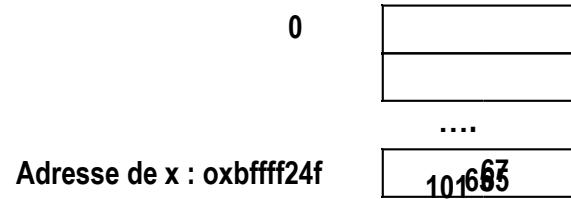
■ char : 1 octet

- La valeur stockée est toujours 8 bits
- Seule la manière dont le programmeur regarde cet octet modifie la perception dont nous la voyons

```
main() { char x;  
         x = 65;  
         printf("decimal: %d ",x);printf("char %c \n",x);  
         putchar(x); puts("");  
  
         x = x + 2;  
         printf("decimal: %d ",x);printf("char %c \n",x);  
  
         x = 'A';  
         printf("decimal: %d ",x);printf("char %c \n",x);  
         puts("Entrer un caractère");  
  
         x = getchar(); /* ou scanf("%c",&x); */  
         printf("decimal: %d ",x);printf("char %c \n",x);  
         putchar(x); puts("");  
     }
```



```
Terminal — bash — 37x9  
x en decimal:65      x en char:A  
A  
x en decimal:67      x en char:C  
x en decimal:65      x en char:A  
Entrer un caractère  
e  
x en decimal:101     x en char:e  
e  
bash-3.2$
```



Opérateurs

Opérateurs

() [] -> .
! ~ ++ -- + - * & (type) sizeof
* / % (modulo)
+ -
<< >> : décalage de bits
< <= > >=
== !=
& : ET bit à bit
^ : Et exclusif bit à bit
| : OU bit à bit
&& : ET logique
|| : OU logique
?:
= += -= *= /= %= ^= |= <<= >>=
,

Associativité

gauche droite
droite gauche
gauche droite
droite gauche
droite gauche
gauche droite

Tableaux



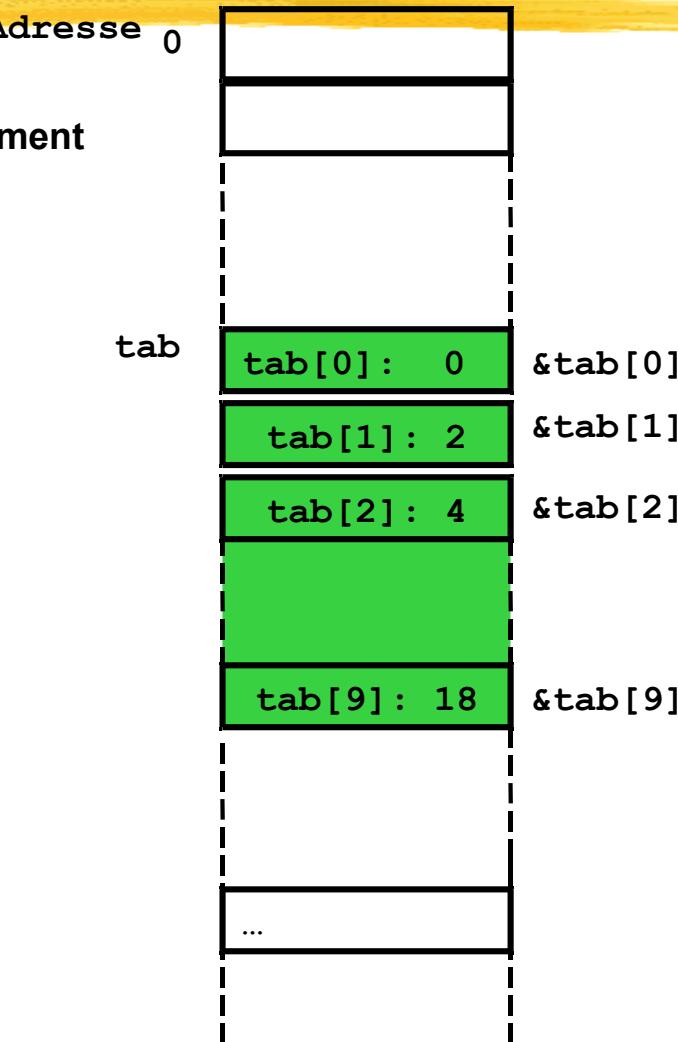
Tableaux

Tableaux

- Collection de variables de **même** type, rangées **continûment** en mémoire
- Déclaration : spécifier
 - | le **type** des éléments
 - | le **nom** du tableau
 - | le **nombre** des éléments
- Exemples :
 - | float c[100]; /* tableau de 100 réels */
 - | int tab[10]; /* tableau de 10 entiers */

Exemple : tab1.c

```
main() { int i;
    int tab[10]; /* Tableau de 10 entiers */
    float c[20]; /* Tableau de 20 réels */
    for (i=0; i<10; i++) tab[i]= 2*i;
        /*Mettre 0,2,4,6..dans les elements*/
    for (i=0; i<10; i++) printf("%d ",tab[i]);
        /* afficher les éléments de t */
}
```



Tableaux 2

Remarques

- Nombre d'éléments **constant**, non modifiable
- Le nom du tableau est son adresse (ie l'endroit où il se trouve en mémoire)

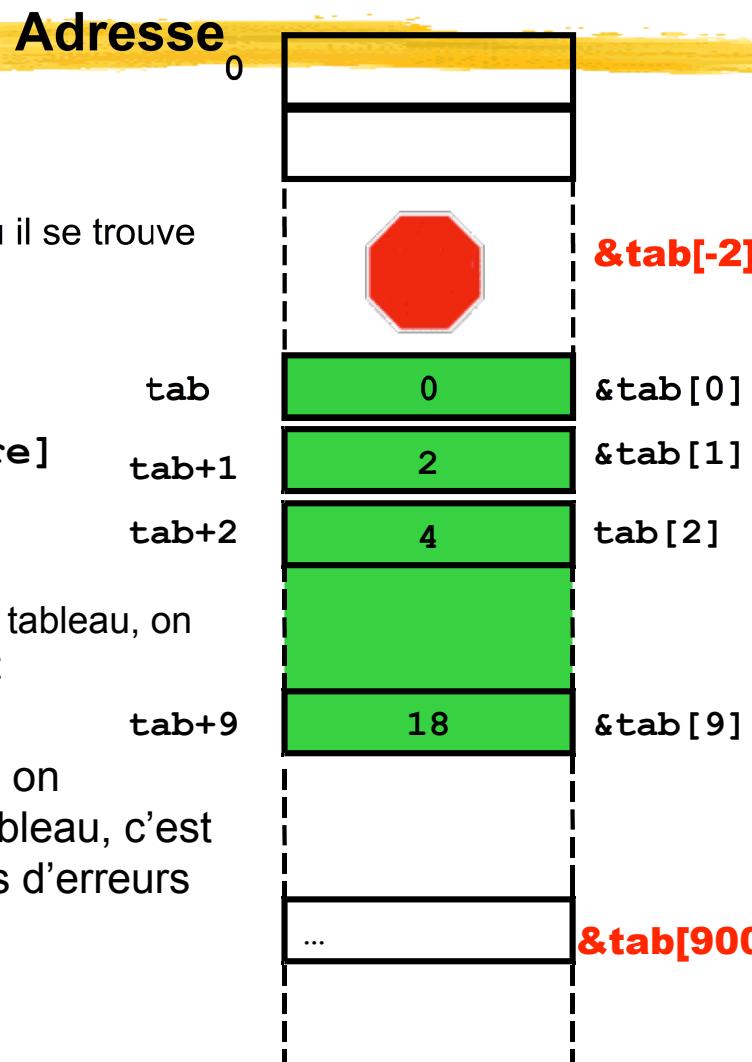
Accès à un élément du tableau :

`nom_du_tableau[expression entiere]`

Exemple : `tab[i]`

Comment fait le compilateur : on part du début du tableau, on ajoute i : c'est l'endroit où se trouve notre élément

Attention : Pas de vérification sur les indices. Si on demande un élément avant ou après la fin du tableau, c'est une erreur à l'exécution du programme mais pas d'erreurs à la compilation



Tableaux 3

Exemple 2 : tab2.c

```
main() { int i;  int tab[10];
  for (i=0; i<10; i++) tab[i]= 2*i;
  puts("Voici les elements : ");
  /* afficher les éléments de t */
  for (i=0; i<5; i++) printf("%d ",tab[i]);
  puts(""); puts("Voici les adresses : ");
  /* afficher les adresses */
  for (i=0; i<5; i++) printf("%p ",tab+i);
  puts("");
  tab[-2]= 18952;
  printf("%d ",tab[900]);
}
```

macbook-pro-de-administrateur:exemples desvignes\$./a.out

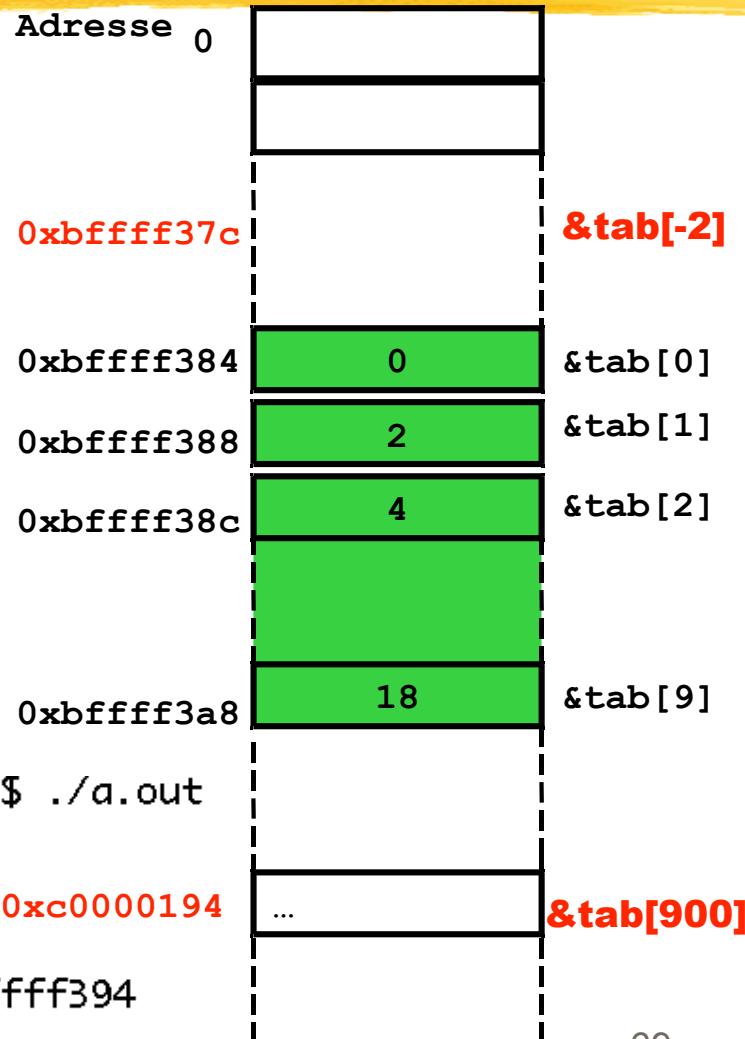
Voici les elements :

0 2 4 6 8

Voici les adresses des elements :

0xbffff384 0xbffff388 0xbffff38c 0xbffff390 0xbffff394

Segmentation fault



Tableaux 4

- Attention : AUCUNE opération globale sur un tableau
 - les opérations et les E/S doivent se faire élément par élément
- En particulier
 - $T1==T2$ ne teste pas l'égalité de 2 tableaux
 - $T1=T2$ ne recopie pas les éléments de $T1$ dans $T2$
- Exemples : tab3.c

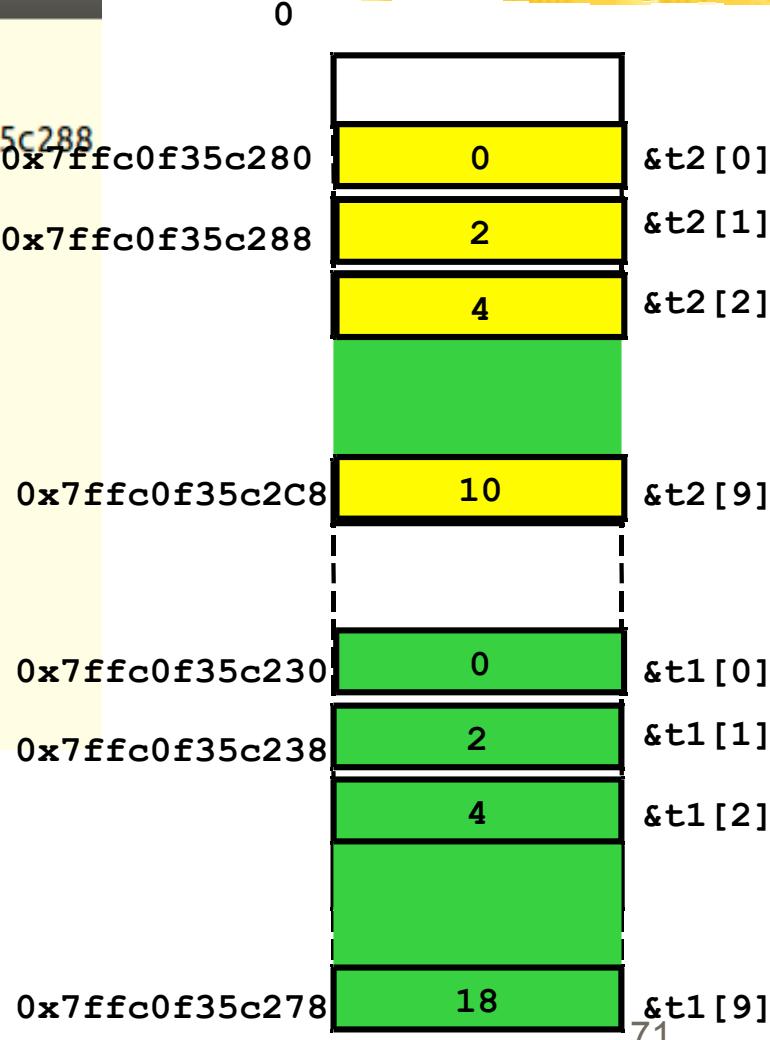
```
main() {int i; double t1[10], t2[10];
         /* t1 et t2 sont 2 tableaux de 10 réels */
    for (i=0; i<10; i++) {t1[i]=2*i; t2[i]=log(100*i+1); }
    printf("Emplacement de t1:%p de t2:%p\n",t1,t2);
    printf("Emplacement de t1[1]:%p de t2[1]:%p\n",t1+1,t2+1);
    printf("Valeur de t1[0]:%lf de t2[0]:%lf\n",t1[0],t2[0]);
    if (t1==t2) printf("t1 et t2 sont au même endroit\n");
    else printf("t1 et t2 ne sont pas au même endroit\n");
    for (i=0; i<10; i++) t2[i]= t1[i];
/*Copie de t2 dans t1: on peut remplacer cette ligne par:
   memcpy(t2,t1,sizeof(t1));
   Mais on ne peut pas utiliser t2=t1;  */
    for (i=0; i<10; i++) printf("Valeur de t2[%d] : %lf\n",i,t2[i]);
}
```

Tableaux 5

```
portet@dracon:~/temp$
```

```
portet@dracon:~/temp$ ./a.out
Emplacement de t1:0x7ffc0f35c230 et de t2:0x7ffc0f35c280
Emplacement de t1[1]:0x7ffc0f35c238 et de t2[1]:0x7ffc0f35c288
Valeur de t1[0]:0.000000 de t2[0]:0.000000
t1 et t2 ne sont pas au même endroit
Valeur de t2[0] : 0.000000
Valeur de t2[1] : 2.000000
Valeur de t2[2] : 4.000000
Valeur de t2[3] : 6.000000
Valeur de t2[4] : 8.000000
Valeur de t2[5] : 10.000000
Valeur de t2[6] : 12.000000
Valeur de t2[7] : 14.000000
Valeur de t2[8] : 16.000000
Valeur de t2[9] : 18.000000
portet@dracon:~/temp$
```

Adresse



Tableaux Attention !

■ Taille de tableau : plusieurs types de déclarion

- 1) `int tab[Expr];` → `Expr` est calculable comme une constante par le compilateur.
- 2) `int tab[] = {1,2,3};` → taille déduite à partir de la *liste d'initialisation*.
- 3) `int tab[]` → sans taille, valable uniquement dans les **prototypes** de fonction.
- 4) `int n=5; int tab[n];` → tableau automatique dont la taille est calculée à *l'exécution*. !!! on **bannit les tableaux automatiques dans ce cours !!!**

■ Tableau automatique :

- Uniquement norme C99 (pas de compatibilité descendante), pas compatible C++<C14.
- Nécessité de contrôler la valeur de `n` à l'exécution (*stack smashing* si incorrect).
- Inefficacité mémoire.

```
main() {
    int n;
    scanf("%d", &n); /* attention au scanf, aucun contrôle !*/
    while (n-- > 0) {
        int tab[n]; /*réalloué n fois avec une taille différente à chaque fois*/
        printf("le tableau tab a %u éléments\n", sizeof(tab)/sizeof (*tab));
    } /* on quitte le bloc, tab est libéré*/
}
```

Tableaux 6

- Les fonctions travaillant sur les zones mémoires : comme un tableau est une zone mémoire continue, on peut utiliser ces fonctions avec les tableaux
- `#include <string.h>`
- `void * memmove(void *s1, const void *s2, size_t n);`
 - Copie n octets de la zone de mémoire src vers la zone dest. Les deux zones peuvent se chevaucher.
- `void *memcpy (void *dest, const void *src, size_t n);`
 - idem, mais les zones ne peuvent pas se chevaucher
- `int memcmp (const void *s1, const void *s2, size_t n);`
 - compare les n premiers octets des zones mémoire s1 et s2.
- `void *memset (void *s, int c, size_t n);`
 - remplit les n premiers octets de la zone mémoire pointée par s avec l'octet c.
- `void swab (const void * from, void * to, ssize_t n);`
 - copie n octets de la zone from dans la zone to, en échangeant les octets adjacents

Exos



- Faire un programme qui effectue la saisie des éléments d'un tableau de 5 réels à travers le clavier
- Faire un programme qui crée 3 tableaux de 10 réels, ajoute les 2 premiers tableaux dans le troisième et affiche ce dernier tableau
- Faire un programme qui recherche la plus petite valeur d'un tableau et qui l'affiche
- Faire un programme qui fait la somme de tous les éléments d'un tableau

Fonctions



Fonctions

Fonctions



- Une fonction est une unité de traitement dans un programme
- Une fonction est utilisée pour :
 - Structurer le programme
 - Décomposer un problème en sous problèmes
 - Spécifier les dépendances entre ces sous problèmes
 - Meilleure lisibilité des étapes de résolution
 - Réutiliser le code
 - une même fonction peut être utilisée plusieurs fois
 - Séparer et rendre étanche différentes parties d'un programme.
 - limite les propagations d'erreurs
 - Offrir une maintenance du code plus aisée
- Une fonction prend en entrée des données et renvoie des résultats après avoir réalisé différentes actions

Utiliser une fonction

- Déclarer le prototype
 - Le prototype d'une fonction indique comment doit être utilisée une fonction
 - Il comporte le nom, la liste des paramètres et la valeur de retour suivi d'un ";"
- L'appel ou l'exécution de la fonction se fait avec les paramètres effectifs par

```
nom_fonction(paramètres);
```
- Exemple

```
int max( int a, int b);
/* Prototype d'une fonction appelée max, qui prend 2 entiers en paramètres et
qui retourne un entier. */

main() {int i,j,k,l,m,n;          /* 6 entiers */
i=10; j=20; k=-8; l=12;
m=max(i,j); /* On appelle max avec i et j, le résultat est mis dans m*/
printf("Le max de %d et de %d est %d\n",i,j,m);
n=max(k,l); /* On appelle max avec k et l, le résultat est mis dans n*/
printf("Le max de %d et de %d est %d\n",k,l,n);
}
```



Ecrire une fonction

- Une fonction est constituée de
 - Son nom
 - Sa liste de paramètres s'ils existent
 - Son type de valeur de retour
 - Des instructions indiquant ce qu'elle doit faire
- Le code de la fonction s'écrit en utilisant les paramètres "formels"
- La valeur de retour
 - Est unique et scalaire : par exemple, une fonction ne peut pas renvoyer 2 réels
 - est renvoyée à l'aide de l'instruction " return " : on quitte la fonction IMMEDIATEMENT, on revient à la fonction qui a appelée
- Exemples : f2.c

```
int max(int a, int b) { int c;  
    if (a>b) c=a;  
    else c = b;  
    return c;  
    printf("Ce point n'est jamais atteint");  
}
```

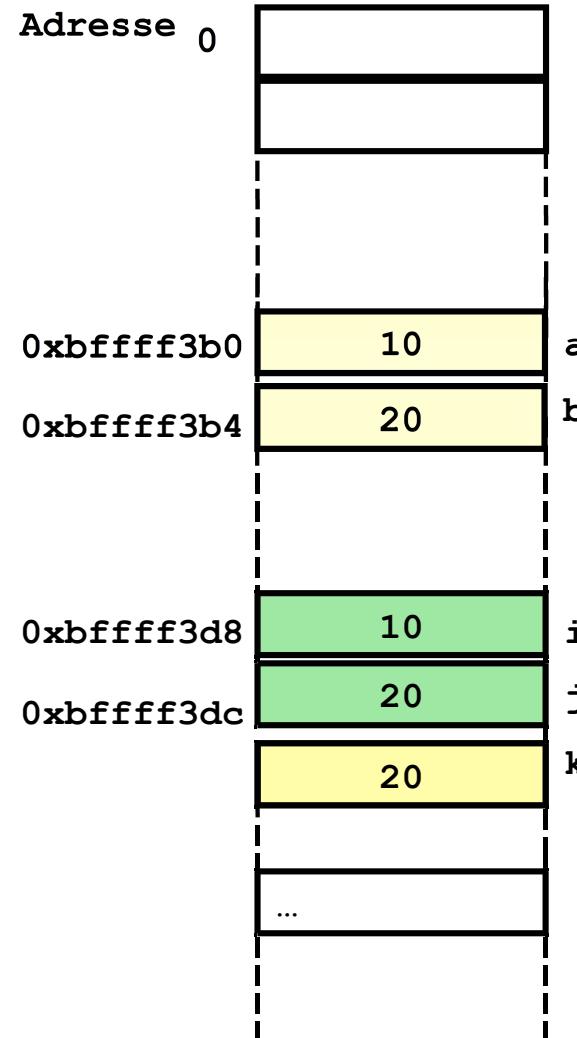
a et b sont les paramètres formels

c est la valeur renournée à la fonction appelant max

Executer une fonction

- Lors de l'appel max(i,j) : Les paramètres sont copiés sur la pile (zone mémoire particulière) en commençant par le plus à droite, soit j en premier.
- On execute alors max
 - j et i sont recopiés dans b et a
 - La fonction max utilise alors a et b et fait son travail
 - a et b sont détruites quand la fonction se termine en executant return
- On revient ensuite à la fonction appelante (ici main)

```
int max(int a, int b) {  
    if (a>b) return a;  
    else return b;  
}  
  
main() {int i,j,k; i=10; j=20; k=0;  
printf("i:%d j:%d\n",i,j);  
k=max(i,j);  
printf("i:%d j:%d \n",i,j,);  
}
```



Passage par valeur

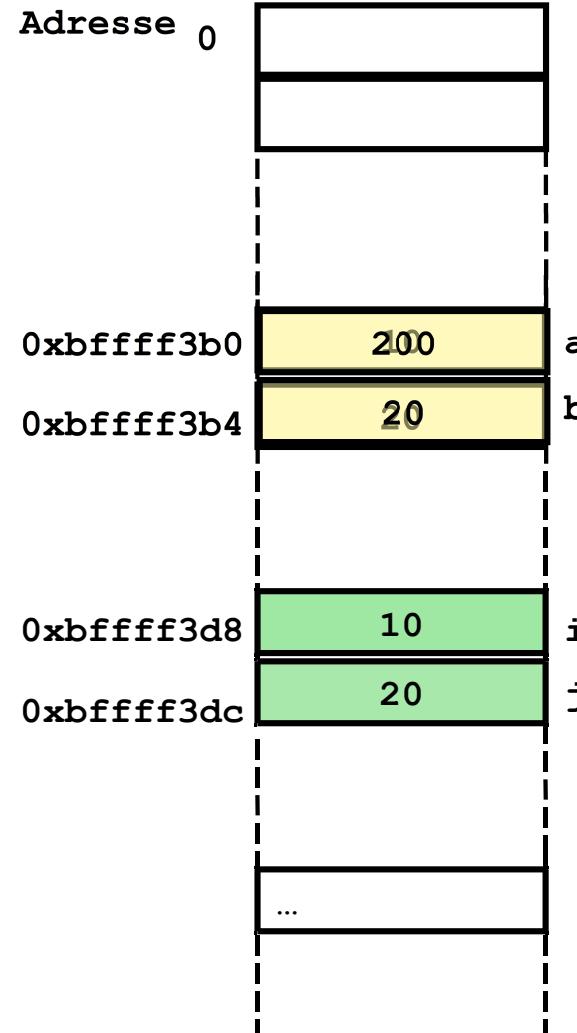
- Lorsqu'une fonction est appelée, elle travaille avec une **copie des paramètres effectifs** : **une fonction ne peut jamais modifier les paramètres qu'on lui donne**
- Exemples : f3.c : fonction qui calcule le produit de 2 nombres

```
void mult1(int a, int b) {  
    printf("Valeur de a:%d et b:%d au debut de la fonction\n",a,b);  
    printf("Adresse de a:%p et b:%p au debut de la fonction\n",&a,&b);  
    a = a * b;  
    printf("Valeur de a:%d et b:%d en fin de fonction\n",a,b);  
} /* Erreur, car a ne sera pas modifié */  
  
main() {int i,j,k,l,m,n,c; /* 7 entiers */  
    i=10; j=20; k=-8; l=12; m=n=c=0;  
    printf("Valeur de i:%d j:%d m:%d avant l'appel de mult1\n",i,j,m);  
    printf("Adresse de i:%p et j:%p avant l'appel de mult1\n",&i,&j);  
    mult1(i,j); /* On appelle max avec i et j, mais i et j ne peuvent pas être  
    modifiées */  
    printf("Valeur de i:%d j:%d m:%d apres l'appel de mult1\n",i,j,m);  
    printf("Adresse de i:%p et j:%p apres l'appel de mult1\n",&i,&j);  
    /* ICI, i vaut toujours 10 */  
}
```

Paramètres des fonctions

- Lors de l'appel mult1(i,j) : Les paramètres sont copiés sur la pile (zone mémoire particulière) en commençant par le plus à droite, soit j en premier.
- On execute alors mult1
 - j et i sont recopiés dans b et a
 - La fonction mult utilise alors a et b
 - a et b sont détruites quand la fonction se termine
- On revient ensuite à la fonction appelante (ici main)
- i et j ne sont pas modifiées, car ce sont les copies a et b qui ont été modifiées

```
→ void mult1(int a, int b) {  
→     a = a * b;  
→ }  
→  
→ main() {int i,j,k; i=10; j=20; k=-8;  
→     printf("i:%d j:%d\n",i,j);  
→     mult1(i,j);  
→     printf("i:%d j:%d k:%d\n",i,j,k);  
→ }
```



Variables locales



- Les variables locales à la fonction (la variable c) sont créées au moment où elles sont déclarées, et détruites à la fin du bloc, au moment où la fonction se termine
- Les paramètres de la fonction (les paramètres a et b) sont créées au moment où la fonction est exécutée, et détruites au moment où la fonction se termine
- Exemples : f4.c

```
int mult2(int a, int b) { int c;
    c = a * b;
    return c;
}

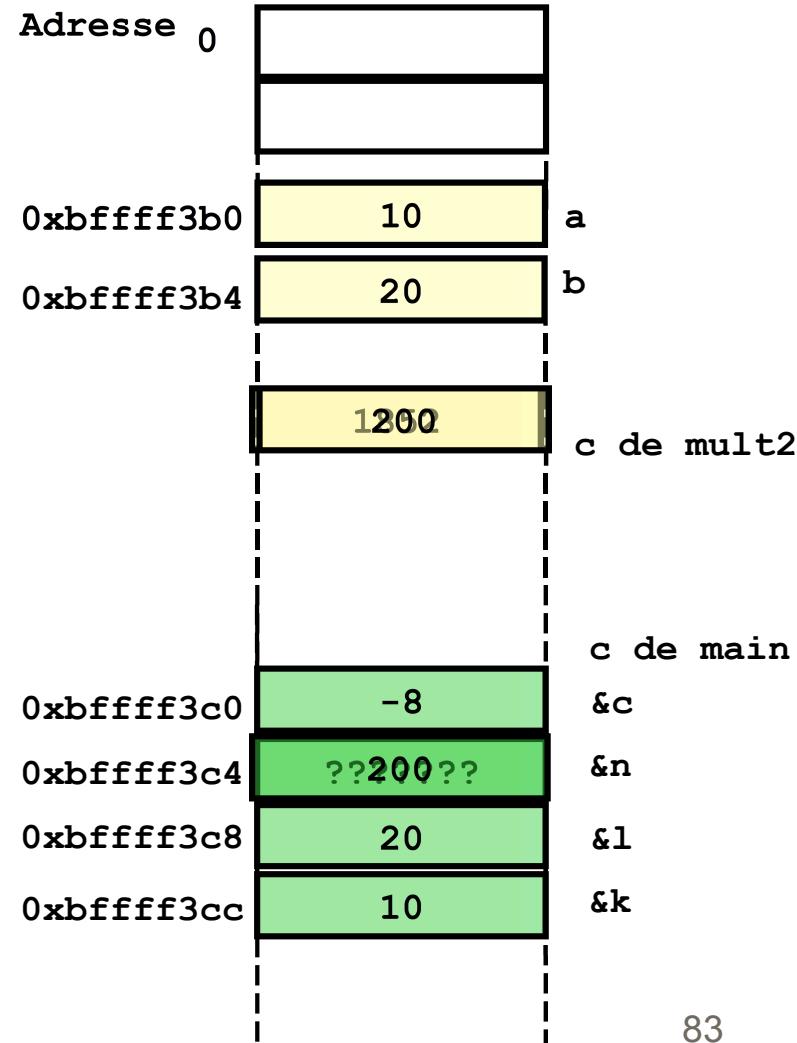
main() {int k,l,n,c;      /* 4 entiers */
    k=10; l=20; n=0; c=15;
    printf("Valeur de k:%d l:%d n:%d c:%d avant l'appel de mult2\n",k,l,n,c);
    n=mult2(k,l); /* On appelle mult2 avec k et l, le resultat est mis dans n*/
    printf("Valeur de k:%d l:%d n:%d c:%d apres l'appel de mult2\n",k,l,n,c);
    /* ICI, c vaut toujours 0, la variable c de mult2 n'existe plus */
}
```

Variable locales

- Lors de l'appel mult2(i,j) : Les paramètres sont copiés sur la pile (zone mémoire particulière) en commençant par le plus à droite, soit j en premier.
- On execute alors mult
 - a et b sont des copies de k et l
 - la variable c est créée sur la pile**
 - La fonction mult utilise alors a et b
 - la variable c est détruite lorsque la fonction se termine**
 - a et b sont détruites quand la fonction se termine
- On revient ensuite à la fonction appelante (ici main)
- i et j ne sont pas modifiées

```
int mult2(int a, int b) {int c=1852;
    c = a * b;
    return c;
}

main() {int k,l,n,c; k=10; l=20; c=-8;
    printf("k:%d l:%d\n",k,l);
    n=mult2(k,l);
    printf("k:%d l:%d n:%d c:%d \n",k,l,n,c);
}
```



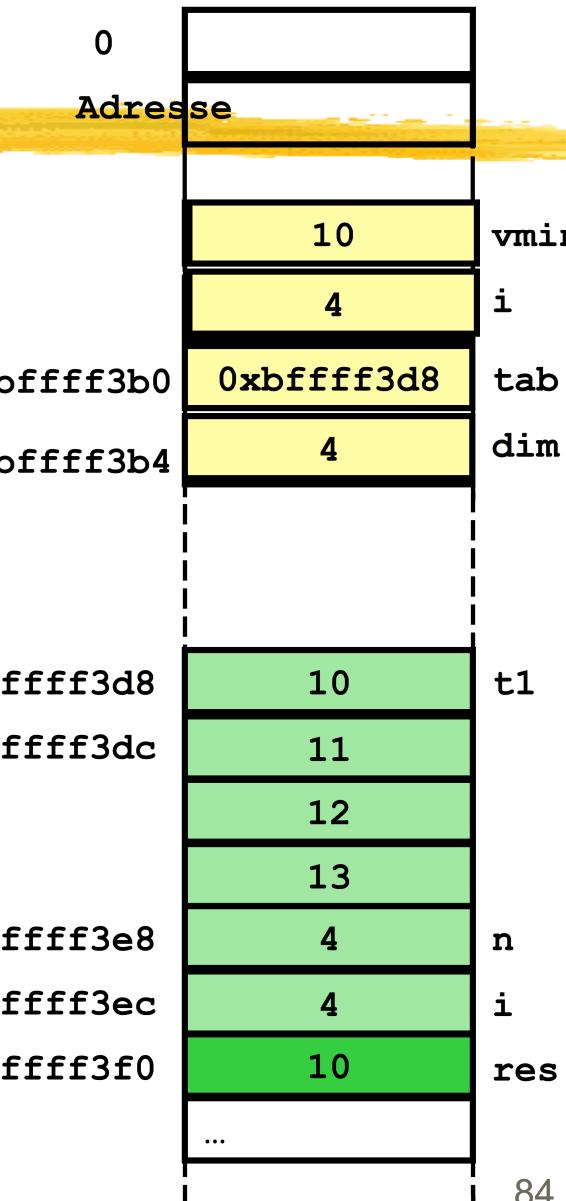
Passer un tableau

```

→ int min1(int tab[], int dim) {
    int i,vmin;
    for(vmin=tab[0], i=1; i<dim; i++)
        if(vmin>tab[i]) vmin=tab[i];
    return vmin;
}
→ main() {int res,i,n=4, t1[4];
    for(i=0;i<4;i++) t1[i]=10+i;
    res=min1(t1,n);
    printf("minimum:%d \n",res);
}

```

- Pour passer un tableau, on déclare un paramètre de type tableau
- Souvent, on ajoute un paramètre donnant le nombre d'éléments du tableau
- Lors de l'appel `min1(t1,n)` : Les paramètres sont copiés sur la pile
- **ATTENTION: tab est l'adresse du tableau !!!!**
- On execute alors `min1`
 - `n` et `t1` sont recopiés dans `dim` et `tab`
 - La fonction `min1` utilise alors `tab` et `dim`
 - `dim` et `tab` sont détruites quand la fonction se termine
- On revient ensuite à la fonction appelante (ici `main`)



En pratique

Pour écrire une fonction, vous devez

Définir son rôle, ses paramètres

- | Pour savoir quels sont les paramètres d'une fonction, définissez clairement le rôle de cette fonction
- | Par exemple :

- | Une fonction retourne la racine carrée d'un réel implique que la fonction a un seul paramètre réel; Son prototype est donc : `double sqrt(double);`
- | Une fonction retourne le plus petit élément d'un tableau de n réels implique que la fonction a 2 paramètres : un tableau de réel et un entier indiquant la taille du tableau. Son prototype est donc : `double min(double t[], int n);`

Faire Un fichier d'entête (extension .h : fonction.h) contenant le prototype de la fonction

Faire Un fichier source C (extension .c : fonction.c) contenant le code de la fonction

Editer le fichier f.h

```
int max(int a, int b);
```

Editer le fichier f.c

```
int max(int a, int b) { int c;  
    if (a>b) c=a; else c=b;  
    return c;  
}
```

En pratique

Pour utiliser une fonction, vous devez

- Inclure le fichier d'entête au début de votre code par `#include "fonction.h"`
- Utiliser votre fonction dans votre programme, écrit dans le fichier p.c
- Compiler le fichier contenant la fonction : clang –c f.c
- Compiler le fichier utilisant la fonction : clang –c p.c
- Créer un executable en réalisant l'édition de liens : clang p.o f.o –o p

Editer le fichier p.c

```
#include <stdio.h>
#include "f.h"
main() {int i,j,m;      /* 7 entiers */
    i=10; j=20;
    m=max(i,j);
    printf("Valeur de i:%d j:%d m:%d \n",i,j,m);
}
```

Exos



- Faire une fonction qui renvoie la valeur absolue d'un entier.
- Faire une fonction qui renvoie le produit de 3 nombres réels passés en paramètres
- Faire une fonction qui calcule la factorielle d'un nombre passé en un paramètre
- Faire une fonction qui prend un tableau en paramètre et remplace chacune des valeurs par leur valeur absolue
- Faire une fonction qui prend un tableau d'entiers en paramètre et vérifie que toutes les valeurs sont impaires

Pointeurs



Pointeurs

Pointeurs

- Variable contenant l'adresse d'un autre objet (variable ou fonction)
 - Adresse : numéro d'une case mémoire
- Déclaration : `type_pointé* identificateur;`
- Exemples :

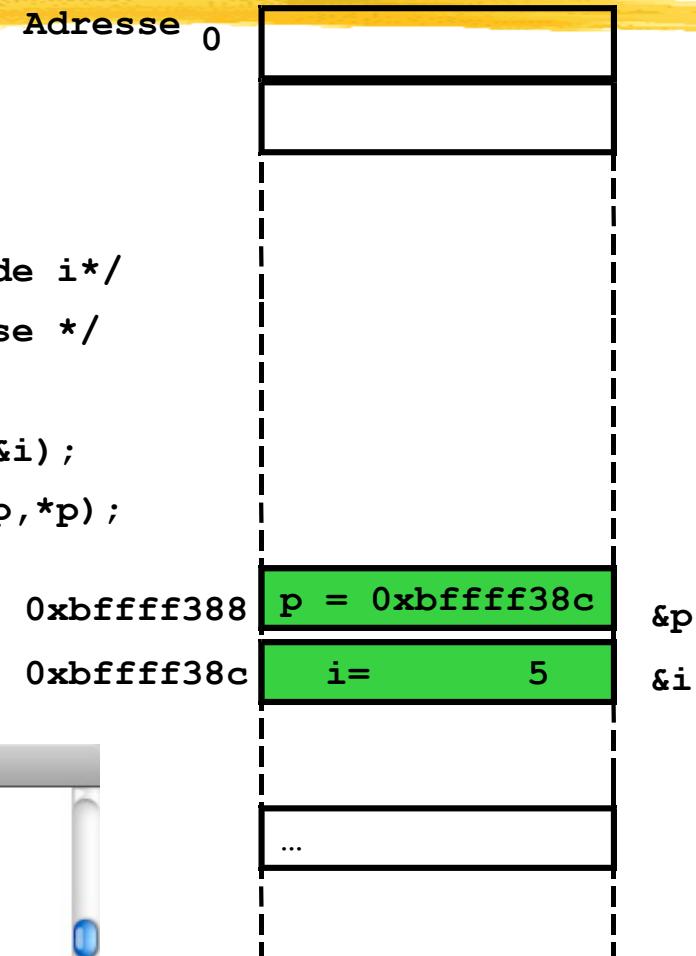
```
int* p1; /* p1 contient l'adresse d'un entier */  
double* p2; /* p2 contient l'adresse d'un réel */
```
- ATTENTION : un pointeur doit toujours être initialisé avant d'être utilisé = Il doit contenir une adresse légale :
 - soit celle d'un objet existant
 - soit celle obtenu par une demande d'allocation dynamique
 - soit NULL, qui est la valeur 0. Il est interdit de lire et écrire à l'adresse 0
- Remarque : on affiche la valeur d'un pointeur en hexadecimal par
 - `printf("Voici la valeur du pointeur %p\n",p);`

Pointeurs

Pointeurs

Exemple : p1.c

```
main() {  
    int i=0;  
    int* p=NULL; /* p1: pointeur sur un entier */  
    p = &i; /* p1 pointe i, ie contient l'adresse de i*/  
    /* ICI, i ou *p1 sont une seule et même chose */  
    *p = 5; /* identique à i=5; */  
    printf("Valeur de i:%d, Adresse de i:%p\n",i,&i);  
    printf("Valeur de p:%p, Valeur pointée:%d\n",p,*p);  
    printf("Adresse de p:%p\n", &p);  
}
```



```
Terminal — bash — 62x6  
macbook-pro-de-administrateur:exemples desvignes$ gcc p1.c  
macbook-pro-de-administrateur:exemples desvignes$ ./a.out  
Valeur de i:5, Adresse de i:0xbffff38c  
Valeur de p:0xbffff38c, Valeur pointee:5  
Adresse de p:0xbffff388  
macbook-pro-de-administrateur:exemples desvignes$
```

Opérations sur pointeurs

- Affectation : donner une valeur au pointeur, celle d'une adresse légitime.

```
p1=&i; /* &i : l'adresse de i */
```

- Indirection : trouver la valeur pointée par p.

```
j = *p1; /* *p1 : ce qu'il y a à l'adresse p1 */
```

- Comparaison :

- == et != : p1==p2; p1 et p2 regardent ils la même adresse ?
- <, >, <=, >= : par exemple, p1<p2 sur un même tableau, p1 est il avant p2

- Arithmétique

- Adresse + entier ==> adresse : p1 +1 est l'adresse de l'élément suivant p1
- Adresse - Adresse ==> entier : p2 -p1 est donc le nombre d'éléments entre les adresses contenues dans p2 et p1. Valide uniquement si p1 et p2 sont de même type.

- ATTENTION : les pointeurs étant typés, les opérations se font en **nombre** d'éléments et non en nombre d'octets

Adresse et tableaux

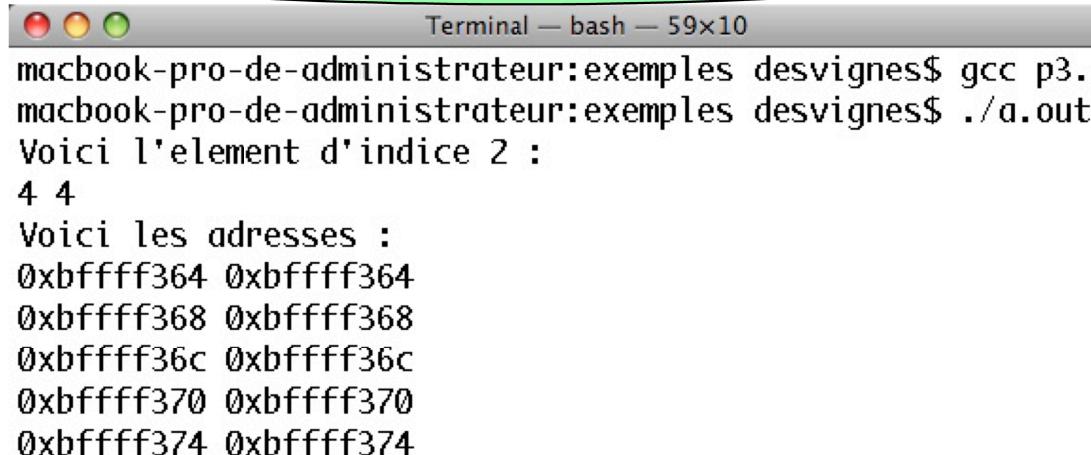
- Nom du tableau : adresse du tableau
- Accès à un élément $t[i]$:
 - on part de l'adresse de début du tableau, on ajoute i et on obtient l'adresse du i^e élément.
 - L'élément est obtenu par l'opérateur d'indirection *
- Conséquences
 - L'élément $t[i]$ s'écrit aussi $*(t+i)$
 - L'adresse de $t[i]$ s'écrit $\&t[i]$ ou bien $(t+i)$

Adresses et tableaux

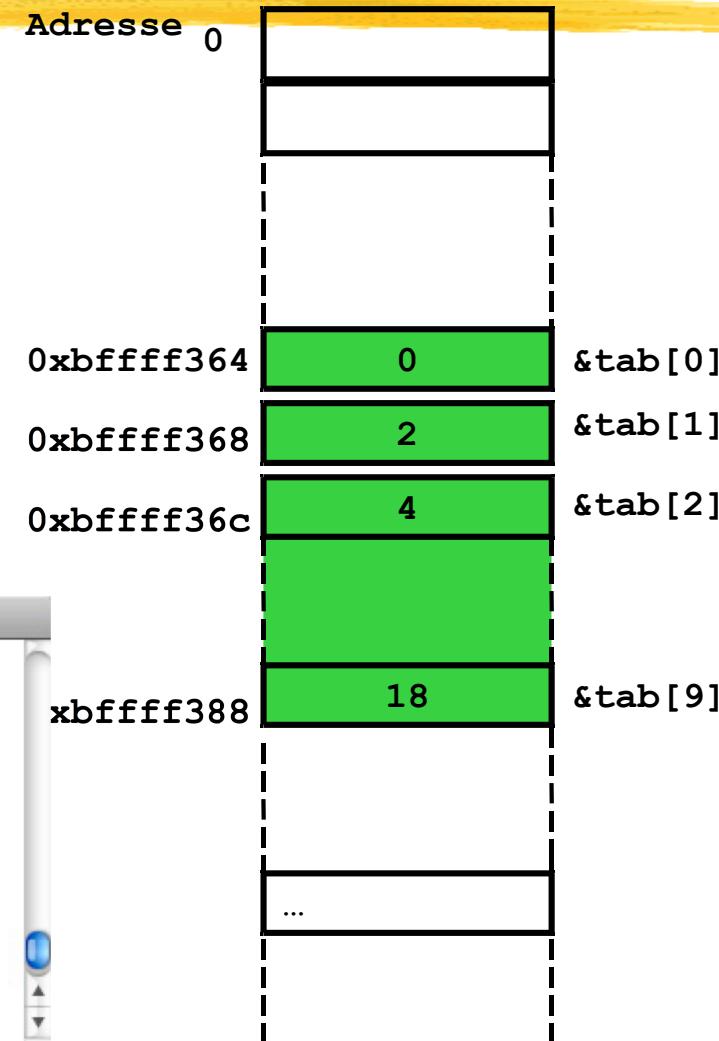
Exemple : p3.c

```
main() { int tab[10]; int i;
    for (i=0; i<10; i++) tab[i]= 2*i;
    puts("Voici l'élément d'indice 2 : ");
    printf("%d %d",tab[2],*(tab+2));
    puts(""); puts("Voici les adresses : ");
    for (i=0; i<5; i++)
        printf("%p %p",tab+i, &tab[i]);
}
```

2 manières différentes d'écrire l'adresse de `t[i]`



```
macbook-pro-de-administrateur:exemples desvignes$ gcc p3.c
macbook-pro-de-administrateur:exemples desvignes$ ./a.out
Voici l'element d'indice 2 :
4 4
Voici les adresses :
0xbfffff364 0xbfffff364
0xbfffff368 0xbfffff368
0xbfffff36c 0xbfffff36c
0xbfffff370 0xbfffff370
0xbfffff374 0xbfffff374
```



Parcourir un tableau avec un pointeur

Exemple : p4.c

```
main() {int* p=NULL; int i; int tab[5];
    for (i=0; i<5; i++) tab[i]=2*i+1;
    → p=tab;
    → while (p<tab+5) {
        printf(" Pointeur: %p ", ,p);
        printf(" Valeur %d", *p);
        → *p=234;
        → printf("Valeur modifiée %d\n", *p);
        → p++;
    }
}
```



A screenshot of a terminal window titled "Terminal — bash — 73x9". The window displays the output of the program p4.c. The output shows the initial state of the array, the modification of the first element, and the final state of the array after all iterations.

```
macbook-pro-de-administrateur:exemples_desvignes$ ./a.out
Adresse du tableau:0xbffff384, adresse de i=0xbffff398
adresse de p=0xbffff39c
Boucle 0 : Pointeur : 0xbffff384 Valeur avant 1 Valeur modifiée 234
Boucle 1 : Pointeur : 0xbffff388 Valeur avant 3 Valeur modifiée 234
Boucle 2 : Pointeur : 0xbffff38c Valeur avant 5 Valeur modifiée 234
Boucle 3 : Pointeur : 0xbffff390 Valeur avant 7 Valeur modifiée 234
Boucle 4 : Pointeur : 0xbffff394 Valeur avant 9 Valeur modifiée 234
macbook-pro-de-administrateur:exemples_desvignes$
```

Exos



- Ex1 : faire un programme qui lit les éléments d'un tableau au clavier en utilisant uniquement des indices pointeurs (et pas entiers)
- Ex2 : faire un programme qui calcule la somme des éléments d'un tableau en utilisant uniquement des indices pointeurs (et pas entiers)
- Ex3 : faire un programme qui copie un tableau de N entiers ($N=10$) dans un autre tableau en utilisant uniquement des indices pointeurs

Paramètres de fonctions

- En C, passage des variables par valeur
 - Il y a recopie de l'objet effectif (de x et y ci dessous) sur la pile
 - La fonction travaille sur une copie des objets effectifs (a et b sont des copies de x et de y)
 - Si la fonction modifie le paramètre formel (a et b), c'est en fait la copie de l'objet effectif qui est modifiée
 - L'objet initial n'est pas modifié (x et y ne change pas)

- Exemple : swap1.c

```
void swap(int a, int b) { int c;
    printf( "Debut de swap a:%d ; b:%d . Adresses de a:%p et b:%p\n",a,b,&a,&b);
    c=a; a=b; b=c; /* On echange a et b en utilisant c */
    printf( "Fin de swap a:%d ; b:%d . Adresses de a:%p et b:%p\n",a,b,&a,&b);
}

main() { int x,y;
    x=1; y=2;
    printf("Avant swap x:%d de y:%d et des adresses de x:%p et y:%p\n",x,y,&x,&y);
    swap(x,y);
    printf("Apres swap : x:%d et y:%d . Adresses de x:%p et y:%p\n",x,y,&x,&y);
}
```

- Que fait ce programme ?

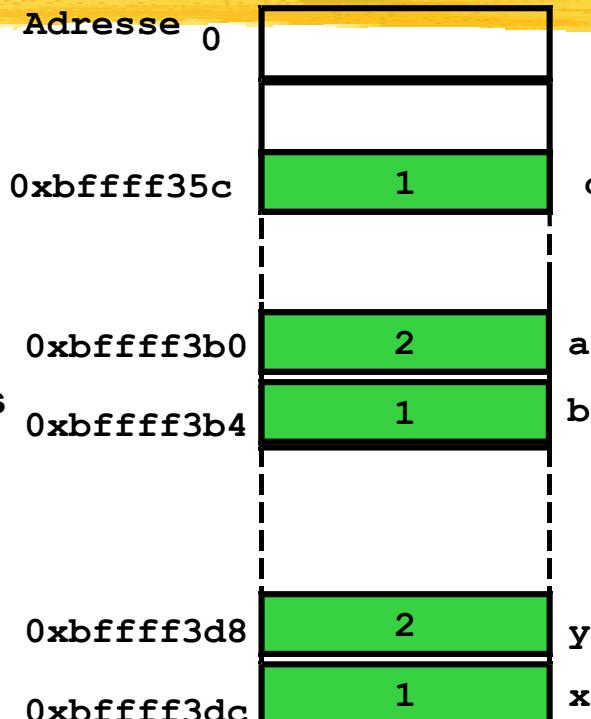
Paramètres de fonctions

Appel de la fonction swap : création des paramètres a et b et de la variable c

Execution de la fonction swap : echange de a et b

Fin de la fonction swap : suppression des paramètres a et b et de la variable c

Que valent x et y ?



```
macbook-pro-de-administrateur:exemples desvignes$ ./a.out
Avant swap : x=1 et y=2 . Adresses de x:0xbffff3dc et y:0xbffff3d8
Debut de swap : a=1 et b=2 . Adresses de a:0xbffff3b0 et b:0xbffff3b4
Fin de swap : a=2 et b=1 . Adresses de a:0xbffff3b0 et b:0xbffff3b4
Apres swap : x=1 et y=2 . Adresses de x:0xbffff3dc et y:0xbffff3d8
macbook-pro-de-administrateur:exemples desvignes$
```

Passage par adresse

- Comment une fonction peut modifier un paramètre ?

- En passant son adresse
- L'adresse n'est pas modifiée
- Mais le contenu (obtenu par l'opérateur *) peut être modifié

- Exemple : swap2.c

```
void swap2(int* pa, int* pb) { int c;
    printf("Debut de swap2: pa:%p ; pb:%p ", pa,pb);
    printf("Contenu de pa :%d et pb:%d\n",*pa,*pb);
    printf("Adresse de pa :%p et pb:%p\n",&pa,&pb);
    c=*pa; *pa=*pb; *pb=c; /* On échange a et b en utilisant c */
    printf("Fin de swap2: pa:%p ; pb:%p ", pa,pb);
    printf("Contenu de pa :%d et pb:%d\n",*pa,*pb);
}
main() { int x,y;
    x=1; y=2;
    printf("Avant swap2 x:%d de y:%d et des adresses de x:%p et y:%p\n",x,y,&x,&y);
    swap2(&x,&y);
    printf("Après swap2 : x:%d et y:%d Adresses de x:%p et y:%p\n",x,y,&x,&y);
}
```

pa et pb sont des adresses de variables existantes à échanger.
Ici, ce seront les adresses de x et y.
Donc, *pa est la même chose que x

On passe les adresses de x et y

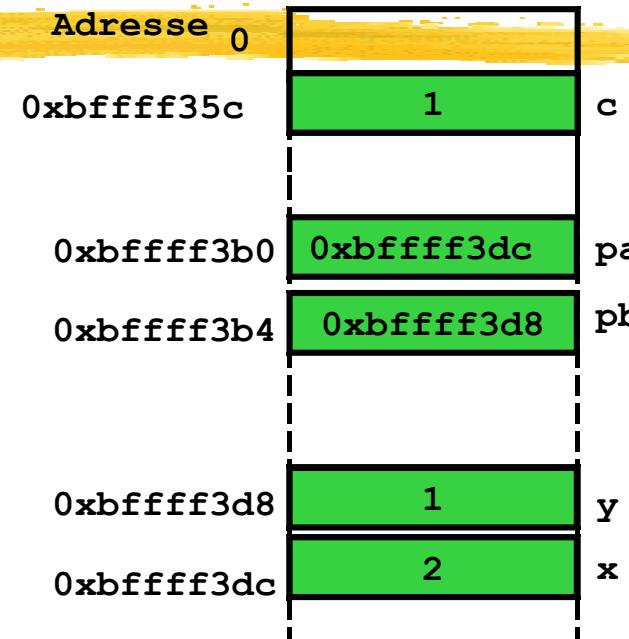
Passage par adresse

Appel de la fonction swap2 : création des paramètres pa et pb et de la variable c.
Que contiennent pa et pb ?

Execution de la fonction swap : echange de *pa et *pb

Fin de la fonction swap2 : suppression des paramètres pa et pb et de la variable c

Que valent x et y ?



```
[[[Amacbook-pro-de-administrateur:exemples desvignes$ ./a.out
Avant swap2 x:1 de y:2 et des adresses de x:0xbffff3dc et y:0xbffff3d8
Debut de swap2: pa:0xbffff3dc ; pb:0xbffff3d8
Contenu de pa :1 et pb:2
Adresse de pa :0xbffff3b0 et pb:0xbffff3b4

Fin de swap2: pa:0xbffff3dc ; pb:0xbffff3d8
Contenu de pa :2 et pb:1
Apres swap2 : x:2 et y:1 . Adresses de x:0xbffff3dc et y:0xbffff3d8
macbook-pro-de-administrateur:exemples desvignes$
```

En pratique

- Comment sait on qu'une fonction doit modifier ses paramètres ?
 - | Définissez et écrivez correctement le rôle de la fonction
 - | Exemple 1: la fonction échange les valeurs de 2 entiers
 - ==> la fonction a 2 paramètres et elle doit modifier la valeur de ces 2 paramètres
 - ==> ces 2 paramètres doivent être passés par adresse
- Si vous avez écrit une fonction avec des paramètres passés par valeur, par exemple le paramètre a, et que vous souhaitez la transformer pour passer le paramètre a par adresse
 - | Il suffit de
 - | remplacer dans la fonction a par *pa, en prenant garde à la priorité des opérateurs
 - | A l'appel de la fonction, remplacer la valeur du paramètre par son adresse à l'aide de l'opérateur &.

Exos



- Ex1 : faire une fonction qui prend un double en paramètre et met ce paramètre à la valeur 3.14159. Faire le programme principal qui appelle cette fonction.
- Ex2 : faire une fonction qui prend 2 entiers en paramètres, ajoute un au premier et soustrait un au second. Faire le programme qui appelle cette fonction
- Ex3 : Donner le prototype de la fonction qui résoudrait l'équation du second degré dans R
- Ex4 : faire une fonction qui prend un tableau de n éléments de type double en paramètre et met tous les éléments à 89.12.

Les chaînes de caractères



Les chaînes de caractères

- En C, les chaînes de caractères ne sont pas un type particulier mais sont :

- Des tableaux de char, exemple : char t[256];
- Terminés par '\0'.

- "Ceci est une chaine" est une chaîne de caractère constante

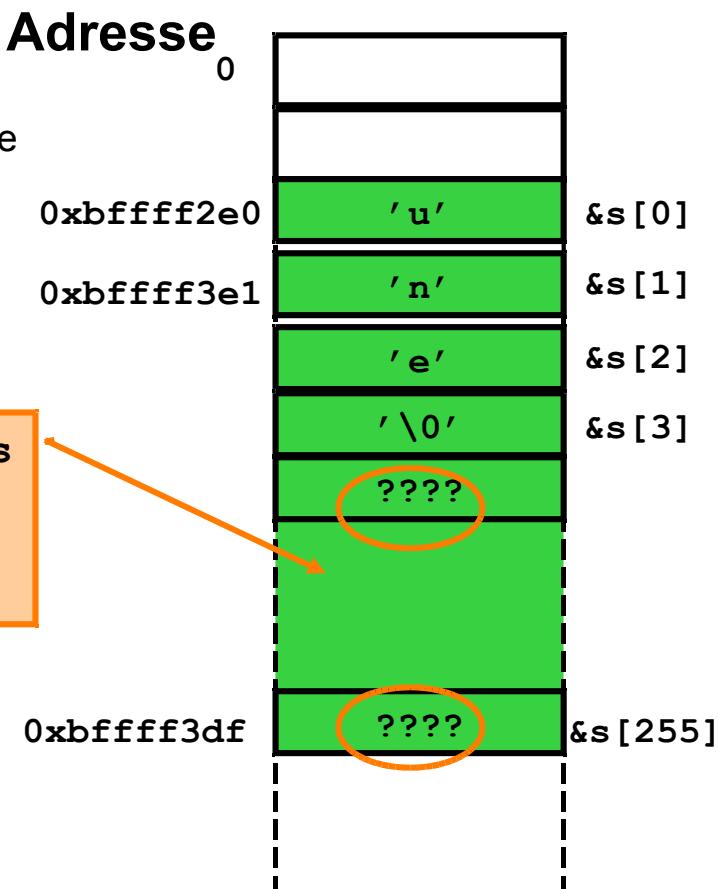
- Pour utiliser une chaîne de caractère, 2 solutions

- Créer un tableau et utiliser les fonctions d'E/S sur les chaînes (scanf, gets) qui ajoutent un '\0' en fin de chaîne.
- Créer un tableau et le remplir élément/élément, y compris le '\0' terminal.

- Exemple : chaîne0.c

```
main() {char s[256];
    s[0]='u';    s[1]='n';
    s[2]='e';    s[3]='\0';
    /* Affichage de s */
    puts(s); /* ou printf("%s\n",s);
}
```

les autres valeurs
sont aléatoires
car on ne les a
pas entrées



Les chaînes de caractères

- Exemple : chaine1.c : crée et lit 2 chaines au clavier avec 2 methodes

```
main() { int i;
    char s1[256], s2[256];
    puts("Entrer une chaine");

/* Lecture d'une chaine. Gets positionne la marque de fin de chaine*/
    gets(s1);          /* On peut aussi utiliser scanf("%s",s1); */
/* Affichage d'une chaine. Utilise la marque de fin de chaine pour
   afficher les éléments du tableau réellement utilisé*/
    puts(s1); /* ou printf("%s",s1); */

/* Lecture caractères/caractères de 10 caractères */
    for (i=0; i< 10; i++) s2[i]=getchar();
/* Ajout de la marque de fin de chaine */
    s2[i]='\0';
/* Affichage de s2 */
    puts(s2);
/* Affichage de s2 caractère/élément*/
    for (i=0; s2[i]!='\0'; i++) printf("%c", s2[i]);
}
```

Les chaînes de caractères



- Les chaînes de caractères sont des tableaux
- ==> AUCUNE opération globale n'est possible :
 - s1==s2 ne compare pas 2 chaînes
 - s1=s2 ne copie pas s2 dans s1
- Il faut utiliser les fonctions spécifiques
 - Concaténation : strcat
 - Copie : strcpy, strncpy
 - Comparaison : strcmp, strncmp, strcasecmp, strncasecmp, strcoll
 - Recherche : strstr, strchr
 - Longueur : strlen
 - Découpage de la chaîne en sous chaîne : strtok
- Toutes ces fonctions mettent le résultat dans une chaîne de caractères
- Attention : la chaîne recevant le résultat doit exister et doit avoir une taille suffisante

Chaînes de caractères

Copie d'une chaîne

```
char* strcpy (char* destin, char* source);

main() { char t1[256], t2[256];
    printf("tapez une chaine");  gets(t1);
    strcpy(t2,t1); puts(t2);

strcpy("chaine", t1);
```



Concaténation de 2 chaînes :

```
char* strcat (char* destin, char * source);
main() { char t1[256], t2[256];
    printf("tapez 2 chaines");
    gets(t1); gets(t2);
    strcat(t1,t2); puts(t1);

strcat ("ceci est une chaine", t2);
```



Duplication (allocation et copie) d'une chaîne

Chaine4.c

```
char* strdup (char* source);

main() { char t1[256], t2[256];
    char* s1; /* Pas de taille: pointeur */
    gets(t1);
    s1 = strdup(t1); puts(s1);

t2=strdup(t1);
```



Autre exemple

```
main() { char t1[256], t2[256], t3[256];
    gets(t1);
    strcat(strcpy(t2,t1),".obj");
    strcat(strcpy(t3,t1),".exe");
    puts(t1); puts(t2); puts(t3);
}
```

Chaînes de caractères

Comparaison de 2 chaînes

```
int strcmp (char* chain1, char* chain2)
```

Retourne :

un entier négatif si chain1 inférieure à chain2
0 si chain1 identique à chain2
un entier positif si chain1 supérieure à chain2



ordre lexicographique.

```
int strcasecmp (char* chain1, char* chain2)
```



majuscules et minuscules indifférentes

Comparaison de 2 chaînes sur n octets

```
int strncmp(char* chain1, char* chain2, int n) }  
/*n= Nombre de caractères à comparer */
```

Formation/extraction dans des chaînes de caractères

```
int sprintf(char* s, char* format, valeurs)  
int sscanf(char* s, char* format, adresses)
```

```
main() { char t1[256], t2[256];  
    int i; float x; double y;  
  
    sprintf(t1,"Valeurs %d et %f et %lf", i,x,y);  
    puts(t1);
```

```
strcpy(t1,"1000 10.3 truc 3.14259");  
sscanf(t1,"%d %f %s %lf ",&i,&x,t2,&y);
```

```
sscanf(t1, "%d %f %lf ",i,x,y) ;
```



Chaînes de caractères



■ Longueur d'une chaîne

```
unsigned strlen ( char* chaine)
```

■ Recherche d'une chaîne

```
char* strstr (char* chain1, char* chain2)
```

- Recherche de la chaîne chain2 dans chaine1.
- Retourne l'adresse de la première occurrence de chaine2 dans chaine1

■ Recherche d'un caractère

```
char* strchr (char* chain1, int n)
```

- Recherche du caractère n dans chaine1.
- Retourne l'adresse de la première occurrence du caractère n dans chaine1

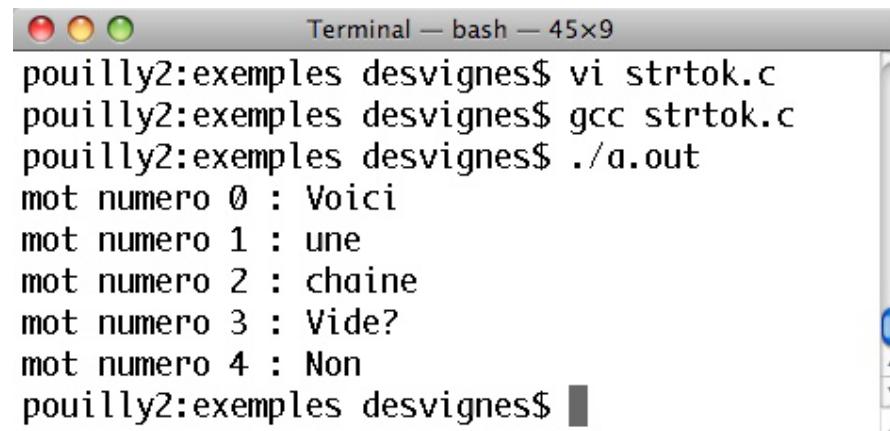
Chaînes de caractères

Découpage d'une chaîne en mots

```
Char* strtok(char* str, char* séparateur)
```

- Découpe la chaîne str en mots.
- Les séparateurs sont définis par les caractères de la chaîne séparateur.

```
#include <stdio.h>
#include <string.h>
main() { char t1[256], t2[256];
    char* p; int i =0;
/* Les séparateurs sont espace, virgule, point et point virgule et d'interrogation */
    strcpy(t1,"Voici une chaîne. Vide? Non");
    /*Initialisation du découpage */
    p=strtok(t1, " .,");
    while (p!=NULL) {
        printf("mot numero %d : %s \n",i,p);
        i++;
        /* On cherche le mot suivant */
        p= strtok(NULL, " .,");
    }
}
```



The terminal window shows the following session:

```
pouilly2:exemples desvignes$ vi strtok.c
pouilly2:exemples desvignes$ gcc strtok.c
pouilly2:exemples desvignes$ ./a.out
mot numero 0 : Voici
mot numero 1 : une
mot numero 2 : chaîne
mot numero 3 : Vide?
mot numero 4 : Non
pouilly2:exemples desvignes$
```

Exercices

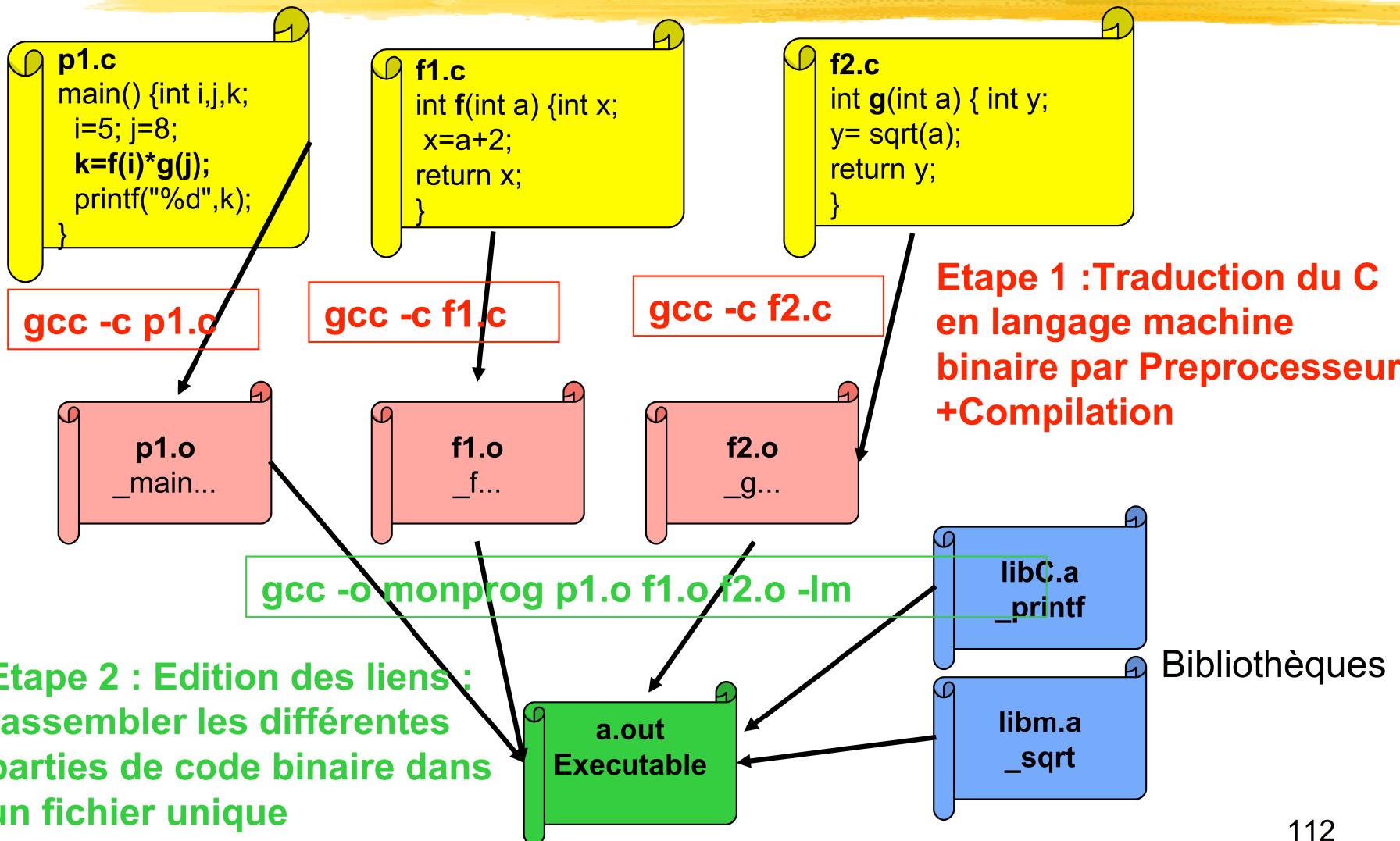


- Ex1 : faire une fonction **taille** qui prend en entrée une chaîne de caractères et retourne le nombre de caractères qu'elle contient.
- Ex2 : faire une fonction **MAJUSCULE** qui prend en entrée une chaîne de caractères et met en majuscule les caractères qu'elle contient. Vous pouvez vous aider de la fonction int toupper(int c);
- Ex3 : Écrivez la fonction **egal** qui prend deux chaînes de caractères en paramètre et retourne 0 si elles sont différentes et un autre nombre sinon.

La chaîne de compilation

- Dans la pratique, un programme peut être constitué de plusieurs millions de lignes de code et de plusieurs centaines de fichiers
- Comment construit-on un programme dans ce cas ?
- Exemple : un programme réalisant le produit de $(i+2)$ par la racine carré de j .
 - La somme $i+2$ est réalisée par une fonction f , qui est écrite dans le fichier $f1.c$
 - La racine carré de j est réalisée par une fonction g , qui est écrite dans le fichier $f2.c$
 - Le programme principal, qui calcule $f(i)*g(j)$ se trouve dans le fichier $p1.c$
 - On veut construire le programme qui affiche le résultat.
- 2 étapes : compilation puis édition des liens

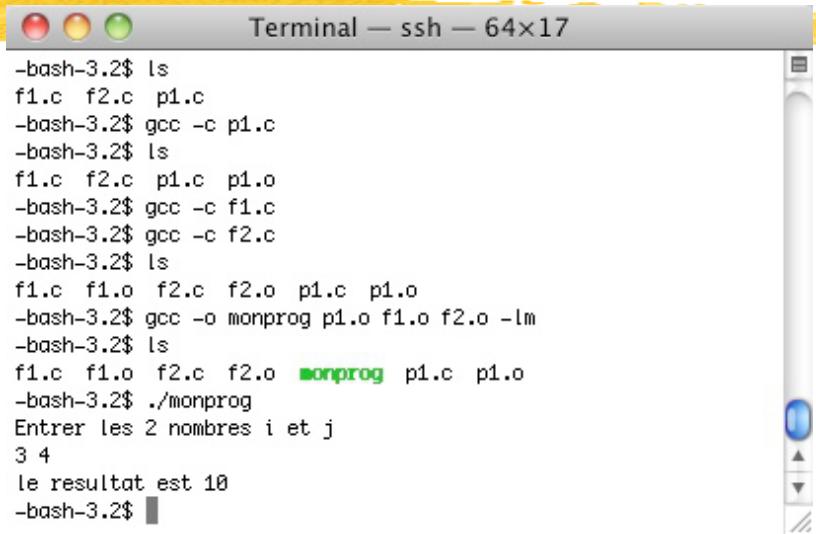
La chaîne de compilation



Compilation

- Pour créer le programme, il faut faire toutes les commandes suivantes :

- gcc -c p1.c
- gcc -c f1.c
- gcc -c f2.c
- gcc -o monprog p1.o f1.o f2.o -lm



```
-bash-3.2$ ls
f1.c f2.c p1.c
-bash-3.2$ gcc -c p1.c
-bash-3.2$ ls
f1.c f2.c p1.c p1.o
-bash-3.2$ gcc -c f1.c
-bash-3.2$ gcc -c f2.c
-bash-3.2$ ls
f1.c f1.o f2.c f2.o p1.c p1.o
-bash-3.2$ gcc -o monprog p1.o f1.o f2.o -lm
-bash-3.2$ ls
f1.c f1.o f2.c f2.o monprog p1.c p1.o
-bash-3.2$ ./monprog
Entrer les 2 nombres i et j
3 4
Le resultat est 10
-bash-3.2$
```

- L'utilitaire make et le fichier Makefile permettent de simplifier cette tâche. Vous l'utiliserez plus tard chaque fois que possible, nous fournissons la plupart du temps le fichier Makefile utile

Structures

**Structures, définition de
types**

Type énuméré

Pourquoi ?

- Restreindre le domaine de valeurs d'un entier (vérification à la compilation)
- Clarifier l'écriture d'un programme

Exemple :

- enum jour {LUNDI,MARDI,MERCREDI,JEUDI,VENDREDI,SAMEDI,DIMANCHE}; /* déclaration du type enum jour*/
- enum jour variable = JEUDI; /*déclaration et initialisation d'une variable de type enum jour*/

On peut donc écrire :

```
int main ( int argc , char * argv[] ) {
    enum jour jourSemaine = LUNDI ;
        // jourSemaine est une variable de type enum jour
    if(jourSemaine == DIMANCHE) {
        printf("c'est dimanche. Grasse mat' ! \n") ;
    }
    ...
    return 0 ;
}
```

il n'est plus possible d'écrire :

```
enum jour jourSemaine = 1 ; // INTERDIT
// => on est forcé d'utiliser les étiquettes
// Et c'est une bonne chose pour la clarté du code !
```

Et encore moins :

```
enum jour jourSemaine = 255 ; // INTERDIT (heureusement...)
```

Créer ses propres types



- Pourquoi ?
 - Clarifier l'écriture d'un programme
- Exemple :
 - j'aime pas les int*
 - je veux indiquer clairement qu'une variable est un octet et pas un caractère
- **Définir un nouveau type : instruction `typedef`**
 - `typedef ancien_type nouveau_type;`

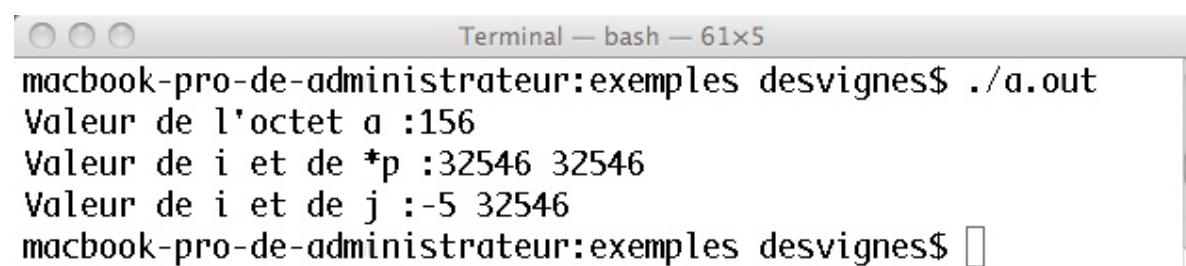
Créer ses propres types

```
typedef unsigned char OCTET;
typedef int* POINTEUR;

void swap(POINTEUR p1, POINTEUR p2) {
    int c=*p1,
        *p1=*p2;
    *p2=c;
}

main() { OCTET a; int i,j;
    POINTEUR p,
    a=156, i=32546; p=&i; j=-5;
    printf("Valeur de l'octet a :%d\n",a);
    printf("Valeur de i et de *p :%d %d\n",i,*p);
    swap(&i,&j);
}
```

POINTEUR peut remplacer
int* partout



Les structures

- Regroupement d'informations de types identiques ou différents

- Relatif à une même entité abstraite.
- permet de manipuler sous un même nom plusieurs éléments d'informations

- Exemple :

- Un point : une structure comportant les deux coordonnées X et Y du point
- Un complexe : les parties réelle et imaginaire du complexe.
- un état civil : une structure regroupant le nom, prénom, n°SS, age....

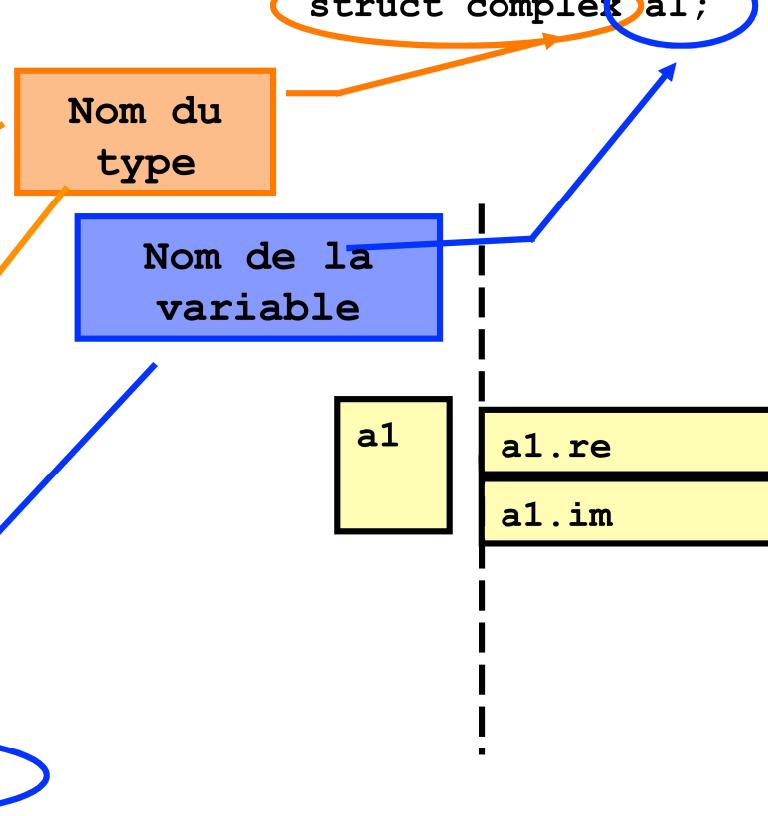
- Déclaration de **type**

```
struct ident1 {  
    type nom_du_champ;  
    ....  
} /* fin de structure */
```

- Définition de variable :

```
struct nomdestructure identif_var;
```

```
struct complex {  
    double im,re;  
};  
struct complex a1;
```

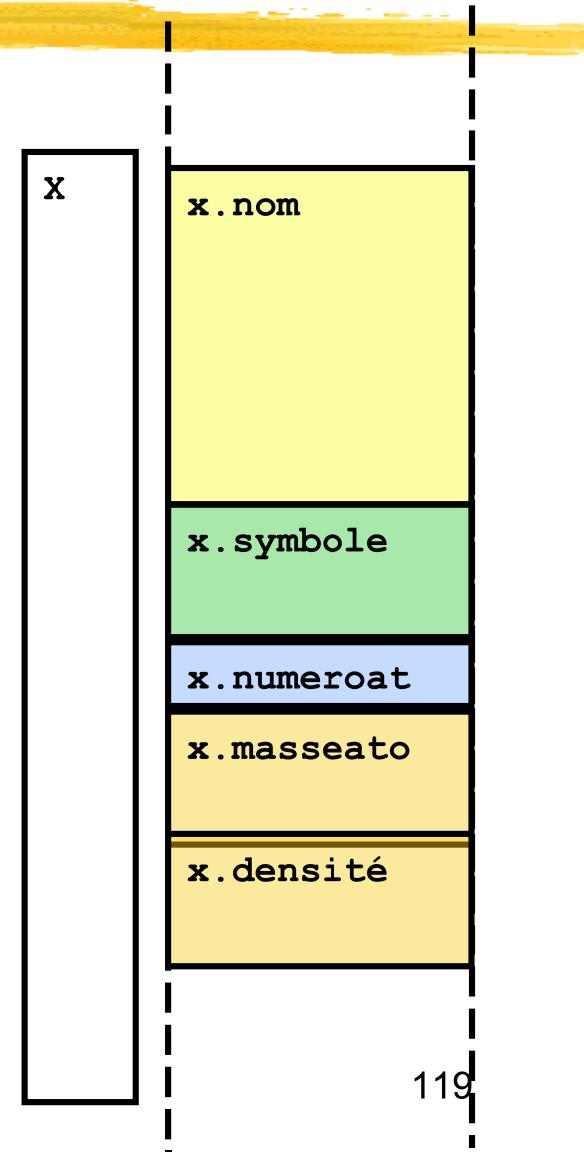


Structures (2)

- Un élément chimique avec les informations de type différent:

```
struct element_atomique {  
    char nom[20] ;  
    char symbole[4] ;  
    int numeroatomique;  
    double masseatomique;  
    double densite ;  
    double fusion ; /* Temperature de fusion en ° */  
    double vap; /* Temperature de vaporisation en ° */  
    double rayon; /* rayon atomique en A */  
    double rayon_cov; /* rayon de covalence en A */  
    char rayonionique[24] ; /* Rayon ionique */  
} x;
```

Le type s'appelle
struct element_atomique



Structure (3)

- Accès à un champ
 - Pour une variable structurée : opérateur .
 - Pour une structure pointée : opérateur ->
- Opérations sur les structures : aucune sauf
 - Affectation de structures même type
 - Passage et retour de structures par valeur dans les fonctions
 - Les éléments de la structure sont des variables à part entière : ils ont une adresse

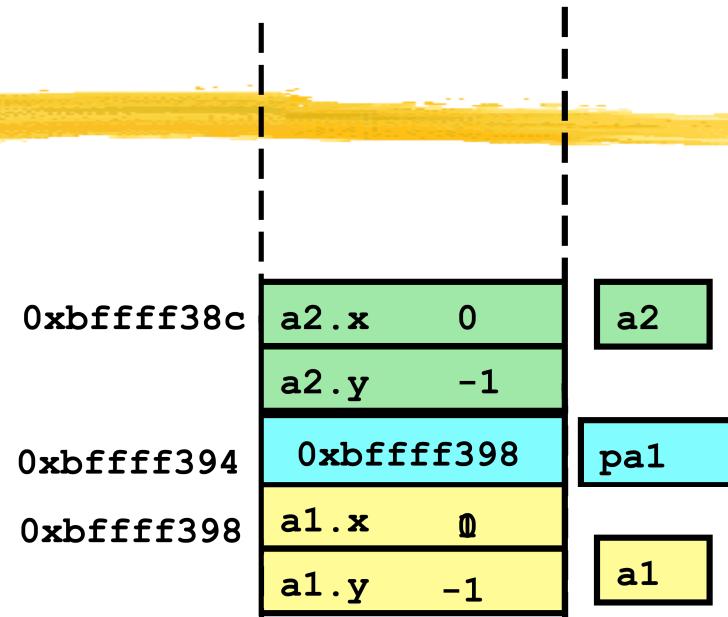
```
typedef struct { int x,y} T_POINT;
```

```
void aff(T_POINT a) {
    printf("%d %d\n",a.x,a.y);
}
main() { T_POINT a1, *pa1, a2;
    a1.x=1; a1.y=-1;
    pa1=&a1; pa1->x=0;
    a2 = a1;
    aff(a2);
    aff(*pa1);
}
```

On définit une structure anonyme et un nouveau type

aff est une fonction qui utilise un T_POINT : elle a accès aux 2 coordonnées de a, a.x et a.y

pa1 est un pointeur. Donc, pa1->x est la partie x de a1



Structures (4)

- Aucune autre action globale, sauf la copie
- en particulier



- Comparaison de structure `a1==a2`
- Lecture/ecriture de structure : `printf("%lf",a1);`

■ Il faut tout faire champ par champ

- Exemple : struct3.c

```
main() { T_POINT a3,a1,a2={1,-1};

    scanf("%lf %lf",&a1.x, &a1.y);
    a3 = a2; /* Copie a2 dans a3 : a3.x=1 et a3.y=-1 */
    puts("Point a1"); aff(a1); /* affiche un point */
    puts("Point a2"); aff(a2); /* affiche un point */
```

```
/* Ce code est impossible
```

```
if (a1==a2)
    puts("a1 et a2 identiques");
else puts("a1 et a2 different");
```



Lecture clavier des
2 champs

```
}
```

Structure (5)

Une structure est un type utilisable pour construire d'autres types

On peut faire

- des tableaux de structures
- des structures de structures

```
struct complex {  
    double im,re;  
};  
typedef struct complex CPMPLEX;  
main() { int I;  
    COMPLEX t1[25],t2[25], t3[25] ; // tableau de 256 complexes  
    for (i=0; i<25; i++) {  
        t1[i].re=t2[i].re+t3[i].re;  
        t1[i].im=t2[i].im+t3[i].im;  
    }  
    for (i=0; i<25; i++)  
        printf("Element %d : %lf +i %lf ",t1[i].re,t1[i].im);  
}
```

Champ de bits

- Découper les octets en tranches de bits
- Gestion des bits sur un octet ou un mot machine
 - utilisé en particulier en SE, les tables de drapeaux (flag), etc...
 - chaque groupe de bits a un rôle différent
 - alignement et champ à cheval sur des mots: dépend des compilateurs (options)
 - Accès aux adresses des champs interdit
- Syntaxe :

```
struct {  
    type ident:nombre de bits du champ; }  
  
struct{unsigned char i:2,j:1,k:4,l:1;} x; /* x est sur  
un octet */  
  
x.i=2; x.j=1; x.k=10; x.l=0;  
/* x=10110100 en binaire ou b4 en hexa  
*/  
printf("Accès aux champs i=%d j=%d  
k=%d l=%d\n",x.i,x.j,x.k,x.l);  
printf("accès à l'octet complet : %d  
%x \n",x,x);
```



Fichier : champbit.c

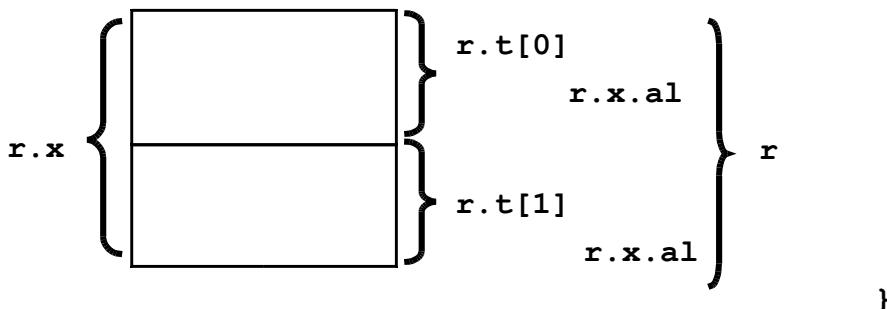
```
#include <stdio.h>  
main() {  
    struct{unsigned char  
    i:2,j:1,k:4,l:1;} x; /* x est sur  
    un octet */  
  
    x.i=2; x.j=1; x.k=10; x.l=0;  
    /* x=10110100 en binaire ou b4 en hexa  
     */  
    printf("Accès aux champs i=%d j=%d  
    k=%d l=%d\n",x.i,x.j,x.k,x.l);  
    printf("accès à l'octet complet : %d  
    %x \n",x,x);  
  
    x=127; /* Error */  
}
```

Union

- Unions : accès à une **même** zone mémoire selon des types différents
- Exemple : voir un entier 16 bits comme 2x8 bits ou 16 bits;

```
union ident {  
    type1 identificateur1;  
    type2 identificateur2; };
```

- gestion par le programmeur
- taille de l'union : taille du plus grand type



- Exemple : union.c : Un short et un tableau de 2 octets et 2 octets

```
main() { int i;  
union REG {  
    unsigned short ax;  
    unsigned char t[2];  
    struct {unsigned char ah,al; } x; };  
union REG r;  
  
r.ax= 0x5621;  
printf("AX: %x\n",r.ax);  
printf("t[0]:%x t[1]:%x\n",r.t[0],r.t[1]);  
printf("AH:%x AL:%x\n",r.x.ah,r.x.al);  
r.t[0]= r.t[1]=0;  
printf("AX: %x\n",r.ax);  
printf("t[0]:%x t[1]:%x\n",r.t[0],r.t[1]);  
printf("AH: %x AL: %x\n",r.x.ah,r.x.al);  
r.x.al=1; r.x.ah=1;  
printf("AX: %x\n",r.ax);  
r.x.al=0x0f; r.x.ah=0xf0;  
printf("AX: %x\n",r.ax);  
}
```

Structure (Exos)

- Définir une structure représentant une matière (Math, physique). Une matière est définie par son nom (“mathématique”), son coefficient, son volume horaire et son code (bijection entre une matière et son code)
- Définir une structure représentant une note, qui contient une valeur et le code correspondant à une matière
- Comment définir une variable notesEtudiant qui est un tableau de notes, une variable matieresExistantes qui est le tableau des matières disponibles.
- Faire une fonction qui affiche les noms des matières et la note correspondante à partir du tableau de notes d'un élève et du tableau des matières. On suppose qu'il existe une fonction code2int qui détermine l'indice de la matière dont le code est donné en paramètre dans le tableau matieresExistantes
 - `int code2int(int code, MATIERE* matieresExistantes, int m)`
- Faire une fonction qui calcule la moyenne des notes d'un élève à partir d'un tableau de notes et du tableau des matières



Fichiers et Entrées / Sortie

Fichiers, Flots



- Qu'est ce qu'un **fichier** : un ensemble d'informations stockées sur un support permanent (disque dur, clé usb, bande ...)
 - fichiers sources en C, document word, programmes, DLL, etc...
- A quoi servent les fichiers ?
 - Conserver des résultats et des données
 - stocker les programmes
 - Appliquer un traitement à toutes les données d'un fichier
 - Éviter de taper toutes les données utiles à un programme
 - Etc...
- Les **entrées/sorties** sont les opérations de lecture/écriture/contrôle entre mémoire programme et le monde extérieur (périphériques: clavier, écran, disque, processus, réseau)
- Elles sont manipulées via la notion de **flot**
 - une entité informatique abstraite qui permet d'unifier et de traiter de la même manière pour le programmeur tous les périphériques de sortie.
 - On lit ou on écrit avec les mêmes fonctions sur l'écran, le réseau, un fichier, l'imprimante, etc...

Texte ou binaire

- Quand vous affichez un nombre sur l'écran avec `printf("%d",i)`
 - `i` est une variable **entière**, codée sur 4 octets, en base 2.
 - Il y a création d'une **chaîne de caractères** contenant les **chiffres** du nombre (base 10)
 - Si `i` vaut 10, la chaîne contient les 3 caractères '1' , '0' et fin de chaîne
 - Si `i` vaut 256845, la chaîne contient les 7 caractères '2' , '5' , '6' , '8' , '4' , '5' et fin de chaîne.
 - Ensuite, le système écrit la chaîne à l'écran.
 - Les nombres affichés prennent un nombre d'octets différent, chaque chiffre est codé sur un octet
 - mais ils sont lisibles facilement sur n'importe quelle machine (less, more, éditeur de texte, etc..)
 - Dans un fichier, il se passe la même chose, ce sont alors des fichiers texte.
- L'autre solution est d'écrire directement les 4 octets composant le nombre
 - Tous les entiers sont alors écrits avec 4 octets, sans aucune transformation
 - Ce sont des fichiers binaires,
 - on les lit difficilement directement. Il faut utiliser la commande `od`, et interpréter soi-même la sortie

Texte (n1.txt) ou binaire (n1.bin)

```
Terminal — bash — 56x5
pouilly2:ex desvignes$ more n1.bin
^@^@^@^A^@^@^?^@^A^@^G[1^A^@^A^@^A
pouilly2:ex desvignes$ more n1.txt
1 127 55536 123456789 65537
pouilly2:ex desvignes$
```

En plus, sur INTEL, dans un entier,
les octets de poids faible sont en tête
et ceux de poids fort en queue.

```
Terminal — bash — 74x9
pouilly2:exemples desvignes$ od -t x1 n1.bin
0000000 01 00 00 00 7f 00 00 00 01 00 15 cd 5b 07
0000020 01 00 01 00
0000024
pouilly2:exemples desvignes$ od -t x1 n1.txt
0000000 31 20 31 32 37 20 36 35 35 33 36 20 31 32 33 34
0000020 35 36 37 38 39 20 36 35 35 33 37 0a
0000034
pouilly2:exemples desvignes$
```

Code ascii du caractère 6, puis de 5, de 5, de 3 et de 6 129

Manipulation des flots



■ 4 étapes

- Définition d'une variable de type flot
 - | pour manipuler les fichiers à travers le langage utilisé
- Ouverture du fichier
 - | établit la liaison entre la variable de type flot créée et le fichier physique
 - | alloue la mémoire nécessaire (buffer)
- Opérations de lecture ou d'écriture
 - | Lecture : récupère un nombre dans le fichier et le met dans une variable
 - | Ecriture : écrit un nombre dans le fichier
- Fermeture du fichier
 - | Vide le buffer et libère la mémoire allouée
 - | Supprime la liaison variable-fichier physique

■ Remarques

- 3 flots prédéfinis
 - | stdout : associé par défaut à l'écran
 - | stderr : associé par défaut à l'écran
 - | stdin : associé par défaut au clavier

Définition, ouverture, fermeture de flots



- Inclure le fichier d'entête

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

- Définition d'une variable f de type FILE* : Descripteur de flot

```
FILE* f;
```

- Ouverture de fichier

```
FILE * fopen(char * nomdefichier, char *moded_ouverture)
```

- création des zones mémoire et initialisation des variables nécessaire
- mode ouverture = "r","w","a", "r+","w+","a+" suivi de "t"ou"b"
 - r : lecture seule
 - w: écriture. Ouverture en début de fichier ou création "Efface" l'ancien contenu
 - a: ajout en fin de fichier ou création
 - r+: lecture/écriture d'un fichier existant
 - w+: Création en lecture/écriture : "Efface" l'ancien contenu
 - a+ : ouverture en lecture/écriture , positionnement en fin de fichier ou création
 - t : fichier texte
 - b : fichier binaire (Attention aux caractères de contrôle comme la fin de fichier si b n'est pas précisé)

- Fermeture d'un flot

```
int fclose(FILE* leflotafermer)
```

Définition, ouverture, fermeture

```
#include <stdio.h>
#include <errno.h> /* pour utiliser la fonction perror */

int main( void ) { FILE* f1; FILE* f2;

    f1 = fopen( "n1.txt", "r" ) ;
    if ( f1 == NULL ) perror( "Erreur ouverture lecture n1.txt" ) ;
    else {
        puts("ouverture en lecture de n1.txt reussie");
        if ( fclose( f1 ) != 0 ) perror( "Une erreur s'est produite à la fermeture" ) ;
    }

    f2 = fopen( "/usr/bin/less", "w" ) ;
    if ( f2 == NULL ) perror("Erreur ouverture écriture /usr/bin/less" ) ;
    else {
        puts("ouverture en écriture de /usr/bin/less reussie");
        if ( fclose( f2 ) != 0 ) perror( "Une erreur s'est produite à la fermeture" ) ;
    }
}
```

```
macbook-pro-de-administrateur:exemples desvignes$ gcc fic0.c
macbook-pro-de-administrateur:exemples desvignes$ ./a.out
ouverture en lecture de n1.txt reussie
Erreur ouverture écriture /usr/bin/less: Permission denied
macbook-pro-de-administrateur:exemples desvignes$
```

Lecture de fichiers texte

```
int fscanf(FILE *f, char* control, adresse des arg) ;
```

- Lecture dans un flux de nombres scalaires (entiers, réels) ou de chaînes de caractères
 - Lit les caractères formant un nombre ('0'...'9', '.',') dans un fichier texte jusqu'à trouver un séparateur (' ', ',', ';', '\n', '\t'...)
 - **convertit** la chaîne de caractère en nombre selon le format préciser dans la chaîne control:
 - entier décimal : %d
 - réel simple précision : %f
 - réel double précision : %lf
 - Retourne le nombre d'objets lus

Contenu du fichier donnes.txt

- Exemple : lectfic1.c

```
main() { FILE* fp; int c,d; double x;
fp = fopen("donnes.txt","rt");
if (fp==NULL) puts("Ouverture impossible");
else { fscanf(fp,"%d %lf %d",&c,&x,&d);
printf("Premier %d, Deuxième %lf, Troisième %d\n",c,x,d);
}
}
```

on lit 1 entier, puis 1 réel, puis 1 entier dans le fichier

Ecriture de fichiers texte



```
int fprintf(FILE *f, char* control, arg) ;
```

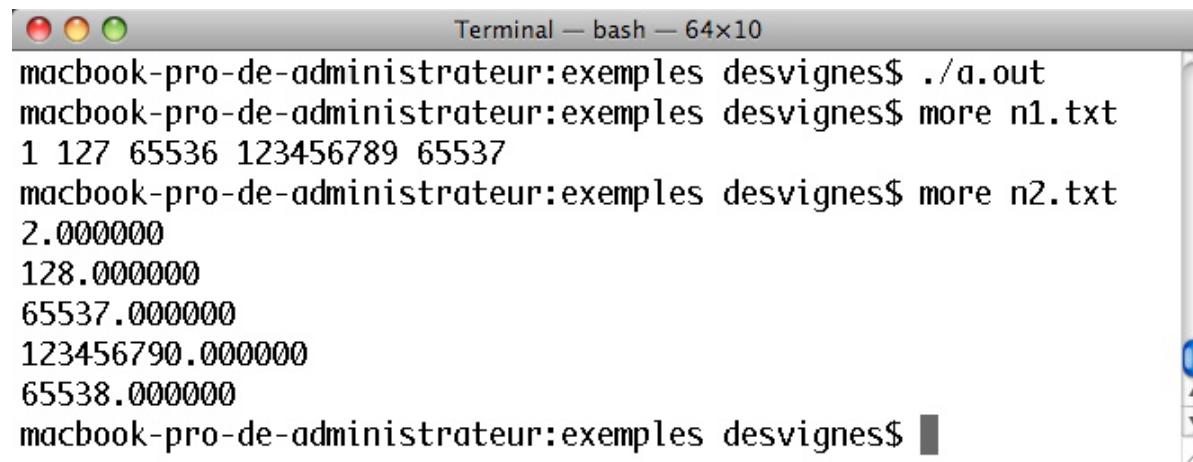
- écriture dans un flux de nombres scalaires : entiers, réels, chaîne
 - Convertit le nombre en chaîne de caractère selon le format préciser dans la chaîne control :
 - entier décimal : %d
 - réel simple précision : %f
 - réel double précision : %lf
 - Ecrit les caractères formant un nombre ('0'..'9','.') dans un fichier texte
 - Retourne le nombre d'objets écrits

Exemple

- Lecture des nombres dans un fichier et écriture des valeurs+1 dans un autre fichier

```
#include <stdio.h>

main() { FILE* f1; FILE* f2; double x;
    /* Ouverture des deux fichiers en lecture ou écriture */
    if ( (f1=fopen("n1.txt","r")) ==NULL) return (1);
    if ( (f2=fopen("n2.txt","w")) ==NULL) return (1);
    while ( fscanf(f1,"%lf",&x) == 1)
        /* tant que la lecture d'un nombre réel x est réussie dans n1.txt */
        fprintf(f2,"%lf\n",x+1);
        /* Ecriture de x+1 dans le nouveau fichier n2.txt */
    fclose(f1); fclose(f2);
}
```



The screenshot shows a macOS Terminal window titled "Terminal — bash — 64x10". The window contains the following text:

```
macbook-pro-de-administrateur:exemples desvignes$ ./a.out
macbook-pro-de-administrateur:exemples desvignes$ more n1.txt
1 127 65536 123456789 65537
macbook-pro-de-administrateur:exemples desvignes$ more n2.txt
2.000000
128.000000
65537.000000
123456790.000000
65538.000000
macbook-pro-de-administrateur:exemples desvignes$
```

Fichiers binaires (1)

- Fichier binaire : copie de la représentation mémoire de la variable dans un fichier
 - Tous les objets de même type ont la même taille
 - 1 int = 4 octets, 1 double = 8 octets,
 - Accès direct à un élément possible
 - Tous les objets du fichier ayant la même taille,
on sait calculer la position du ième objet par rapport au début du fichier
 - Dépend de la machine et du SE
 - Big Indian/ Little Indian pour les entiers
 - Norme IEEE pour les flottants (Cray)
 - ↗ Fichiers de données binaires non compatibles
 - Pas de marque de fin de lignes/ fichiers
- Exemple : n1.txt, n1.bin : même contenu mais en texte ou en binaire
- Différence fichier texte/fichier binaire : 5 entiers : 1 127 65536 123456789 65537

N1.txt : contenu affiché en hexadécimal

0000000	3120	3132	3720	3635	3533	3620	3132	3334
0000020	3536	3738	3920	3635	3533	370a		

N1.bin	1 ^{er} nombre	2 [°] nombre	3 [°] nombre	4 [°] nombre
0000000	0100	0000	7f00	0000
	0000	0100	0100	15cd
0000020				5b07

Fichiers binaires (2)

- Ouverture d'un fichier binaire
 - flag b dans le mode d'ouverture

- lecture dans un fichier binaire

```
int fread(void* t, int size, int nbel, FILE *fp);
```

- lecture de nb objets de taille size à partir du flux fp
- range les objets lus (nb * size octets) à l'adresse de l'objet t.
- Cette adresse est souvent celle d'un tableau.
- Retourne le nombre d'objets lus ou 0 si erreur ou fin de fichier
- **Le tableau doit être alloué et contenir assez de place**

- Ecriture dans un fichier binaire

```
int fwrite(void *t, int size, int nbel, FILE *fp);
```

- écriture de nb objets de taille size à partir du flux fp dans l'objet t.
- Retourne le nombre d'objets lus ou 0 si erreur ou fin de fichier

Fichiers binaires (3)

- Exemple : lecture de n1.bin (5 entiers) et écriture de x+1 dans n2.bin

```
main() { FILE* f1, *f2; int i;
    int t[5];
    f1=fopen("n1.bin","rb");
    f2=fopen("n2.bin","wb");
    if (fread(t,sizeof(*t),5,f1) <5) /* Lecture des 5 entiers */
        printf("Impossible de lire 5 entiers");
    else { for (i=0;i<5;i++) t[i]=t[i]+1; /* on écrit aussi t[i]++; */
        if (fwrite(t,sizeof(*t),5,f2) <5) /* écriture des 5 entiers */
            printf("Impossible d'écrire 5 entiers");
    }
    fclose(f1); fclose(f2);
```

On lit les 5 entiers en 1 seule opération,
mais il faut que t puisse contenir ces 5 entiers

On écrit les 5 entiers en 1 seule opération

```
pouilly2:exemples desvignes$ od -t x1 n1.bin
00000000  01 00 00 00 7f 00 00 00 00 00 01 00 15 cd 5b 07
00000020  01 00 01 00
00000024
pouilly2:exemples desvignes$ od -t x1 n2.bin
00000000  02 00 00 00 80 00 00 00 01 00 01 00 16 cd 5b 07
00000020  02 00 01 00
```

Fichiers binaires (4)

Positionnement dans un fichier binaire

```
int fseek(FILE *fp, int offset, int from);
```

- Positionne la prochaine lecture/ecriture à offset octets de from (0:debut, 2:fin ou 1:position actuelle) du fichier

```
main() { FILE* fp; int a; int t[5];
```

```
/* Lecture ET écriture */
```

```
    fp=fopen("Fichier_a_lire","r+b");
```

```
    fseek(fp,2*sizeof(*t),0);
```

```
    /* On se positionne sur le  
    3ième réel dans le file*/
```

```
    a=16785;
```

```
    if (fwrite(&a,sizeof(*t),1,fp) <1)  
        printf("Impossible d'crire");
```

```
    /* On revient au début du fichier */
```

```
    fseek(fp,0,0);
```

```
    // Lecture et affichage des nombres
```

```
    while (fread(&a,sizeof(a),1,fp) ==1)  
        printf("%d ",a);
```

```
    puts("");
```

```
}
```

Debut du
fichier

1	127	16785	123456	65537
---	-----	-------	--------	-------



Fichiers Exos



- Ecrire un programme qui affiche à l'écran le contenu du fichier texte f.c
 - Ecrire un programme qui compte les entiers stockés dans un fichier binaire ne contenant que des entiers
 - Ecrire un programme qui affiche à l'écran le contenu d'un fichier binaire. Ce fichier contient des éléments comprenant un entier, un réel double précision et une chaîne de 32 caractères.
 - Ecrire un programme qui affiche à l'écran le contenu du ième élément du fichier binaire précédent.
 - Ecrire un programme qui remplace le contenu du ième élément du fichier binaire précédent par l'élément contenant 0,0,0," "
 - Ecrire un programme qui transforme tous les octets d'un fichier texte donnees.txt en leur ajoutant une valeur constante a.
-



Allocation dynamique

Allocation dynamique ?



Exemple : le format d'image PPM contient le magic number “P5” sur la première ligne, puis le nombre de lignes et de colonnes de l'image, puis toutes les données de l'image

Si mon programme recopie une image PPM, il doit

ouvrir le fichier

lire les tailles,

lire les données pour les ranger dans une matrice

Probleme : on ne sait pas quelle est la taille de la matrice utile.

Comment créer une matrice dont la taille est lue et connue quand on execute le programme et inconnue quand on écrit le code C ?

Allocation dynamique ?

- Que faire quand on ne connaît pas la dimension d'un tableau que l'on doit utiliser lorsque l'on écrit le programme
 - Solution 1 : créer le plus grand tableau possible, et utiliser la partie dont on a besoin quand on exécute le programme
 - Le tableau est créé à la compilation, il a toujours la même taille
 - Solution 2 ; créer le tableau lorsque l'on connaît sa taille, c'est à dire pendant l'exécution du programme : c'est l'allocation dynamique
 - Le tableau a juste la bonne taille, cette taille est différente selon les exécutions
- Allocation dynamique : les 3 éléments utiles
 - Créer un pointeur du type du tableau : double* p;
 - La fonction void* malloc(int n, int d) permet de créer un tableau de « n » éléments, chaque élément ayant la dimension « d ». Cette fonction retourne l'adresse du tableau créé si l'allocation est possible, NULL sinon. Le tableau est initialisé avec des octets nuls.
 - La fonction free(void* p) permet de libérer la mémoire allouée pour le tableau « p », alloué par malloc (ou calloc)

Exemple

Exemple : alloc0.c

```
main() { int i=0, dim=0;  
    double* t1=NULL; double* p=NULL;  
    printf("Donner la dimension voulue ");  
    scanf("%d", &dim);  
  
    t1=calloc(dim, sizeof(*t1));  
  
    for (i=0; i<dim; i++) t1[i]=2*i+1;  
    for (i=0; i<dim; i++) printf("%lf ", *(t1+i));  
    for (p=t1; p<t1+dim; p++) printf("%lf ", *p);  
  
    free(t1);  
  
    /* ATTENTION : on ne peut plus utiliser t1 ici */  
}
```

Création d'un tableau de dim réels
Attention : c'est le type de t1 (double*) qui détermine la nature des éléments, et non la fonction calloc

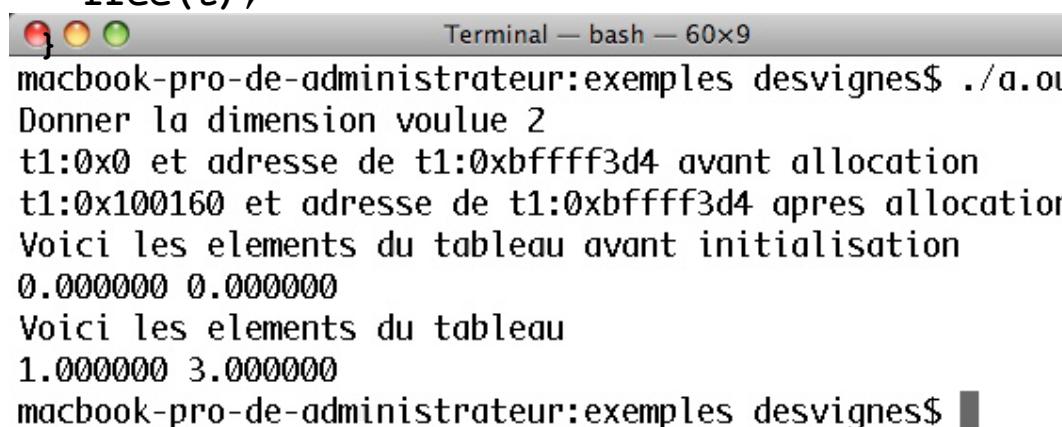
On peut maintenant utiliser indifféremment les notations t[i] et *(t+i) et même *p

Le tableau est devenu inutile : on libère la mémoire allouée

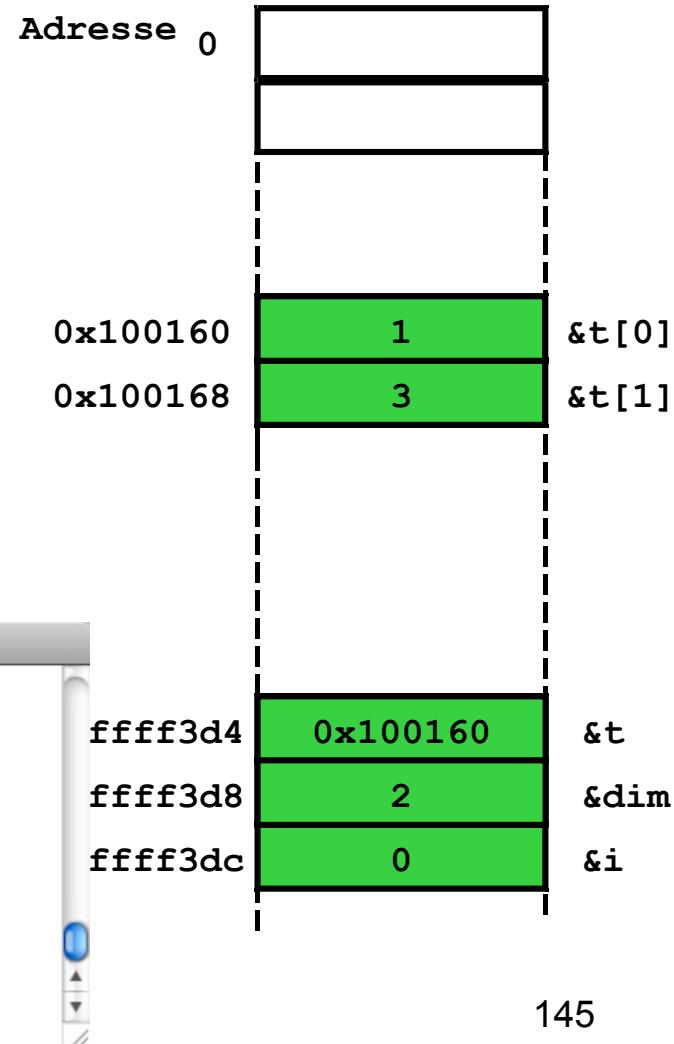
Exemple

✓ Exemple : alloc0.c

```
main() { int i=0, dim=0;  
    double* t=NULL;  
    printf("Donner la dimension voulue ");  
    ➔ scanf("%d", &dim);  
    printf("t1:%p et adresse t1%p avant\n", t, &t);  
    ➔ t=calloc(dim, sizeof(*t));  
    printf("t1:%p et adresse t1%p après\n", t, &t);  
    ➔ for (i=0; i<dim; i++) printf("%lf ", t[i]);  
    for (i=0; i<dim; i++) t[i]=2*i+1;  
    puts("Voici les éléments du tableau");  
    for (i=0; i<dim; i++) printf("%lf ", t[i]);  
    free(t);
```

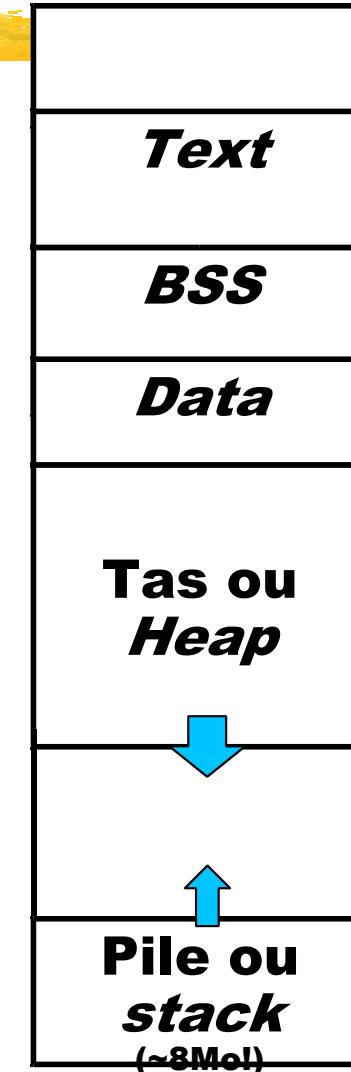


macbook-pro-de-administrateur:exemples_desvignes\$./a.out
Donner la dimension voulue 2
t1:0x0 et adresse de t1:0xbffff3d4 avant allocation
t1:0x100160 et adresse de t1:0xbffff3d4 après allocation
Voici les éléments du tableau avant initialisation
0.000000 0.000000
Voici les éléments du tableau
1.000000 3.000000
macbook-pro-de-administrateur:exemples_desvignes\$



Mémoire et allocation

- Intérêt de l'allocation dynamique
 - Dimension inconnue lors de la conception du programme
 - Données de grande taille
 - Données de taille variables : tableaux, listes, ...
- Allocation dans une zone mémoire gérée par **l'utilisateur** : le tas : heap
- Les variables locales sont gérées par le compilateur sur la pile : stack de taille limitée
- Le code est stocké dans une zone mémoire appelée text
- Les variables globales sont gérées dans la zone « BSS »



Mémoire et allocation (exos)

- Faire une fonction qui prend en paramètres un tableau de double et sa dimension, et qui clone (crée un nouveau tableau et recopie) ce tableau. La fonction retourne le clone.
- Faire le programme principal qui utilise cette fonction. Si le clone est modifié, le tableau initial est-il modifié ?
- Quelle différence entre un tableau dynamique et un tableau automatique
- Pourquoi le programme suivant est-il faux ?

```
int * mauvais1 () { int tab[10];
    for (i=0;i<10;i++) tab[i]=0;
    return tab;
}
main() { int * p; int i;
    p=mauvais1();
    for (i=0;i<10;i++) printf("%d", p[i]);
}
```

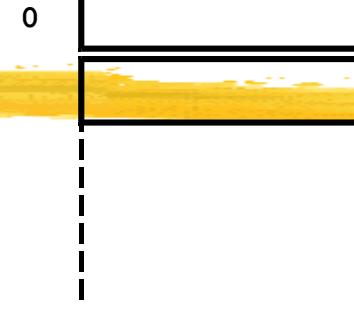
- Reprendre la fonction 1 ci dessus en utilisant le passage par variable pour le clone. Que devient le programme principal ?

Matrices

$$\begin{pmatrix} 2 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_{N-2} \\ v_{N-1} \end{pmatrix} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \dots \\ f(x_{N-2}) \\ f(x_{N-1}) \end{pmatrix}$$
$$A \quad x \quad = \quad b$$

Tableau multi dimensions

Adresse



- Il suffit de spécifier toutes les dimensions

- type identificateur[dim1][dim2]....;
- Exemple : int t[2][3]; Tableau de 2 lignes et 3 colonnes
- Les éléments sont stockés de manière contiguë, ligne par ligne

Exemple : tab2d1.c

```
main() { int i,j;  int t[2][3];
for (i=0; i<2; i++) for (j=0; j<3; j++)
    t[i][j]=i+j;
printf("Valeur du tableau ");
for (i=0; i<2; i++) for (j=0; j<3; j++)
    printf("%d ",t[i][j]);
printf("\nAdresses des elements du tableau ");
for (i=0; i<2; i++) for (j=0; j<3; j++)
    printf("%p ",&t[i][j]);
}
```

macbook-pro-de-administrateur:exemples desvignes\$./a.out

Valeurs du tableau 0 1 2 1 2 3

Adresses du tableau 0xbffff3c0 0xbffff3c4 0xbffff3c8 0xbffff3cc 0xbffff3d0 0xbffff3d4

Tableau multi dimensions

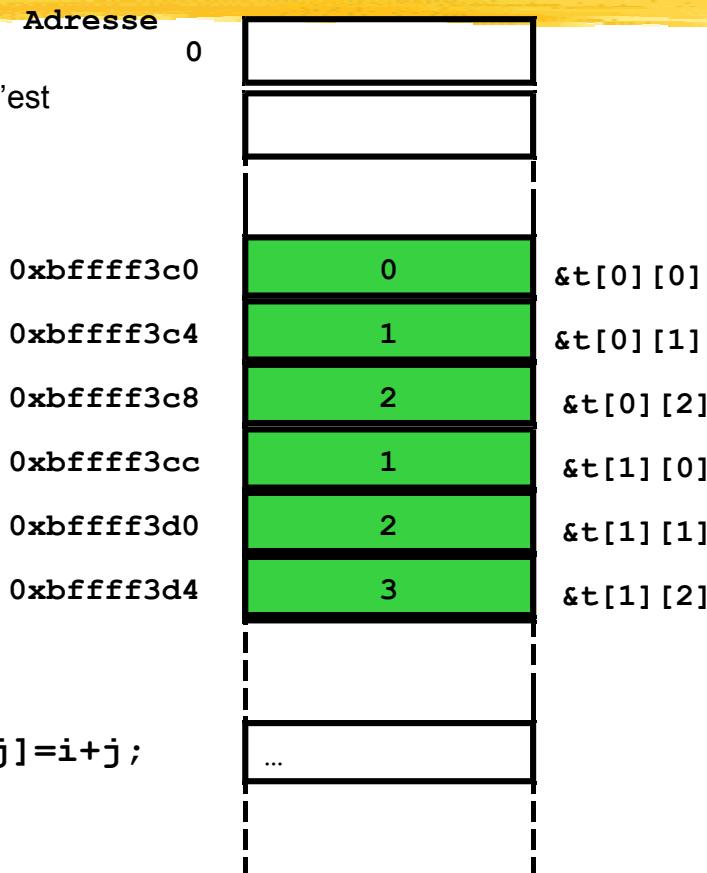
- Comment le compilateur traduit $t[i][j]$ d'un tableau int $t[2][3]$;

- on démarre au début du tableau (l'adresse du premier élément) : c'est « t »
- on saute i lignes (chaque ligne contient 3 éléments), on a alors l'adresse de la ligne i : c'est « $t+3*i$ »
- on saute les j premiers éléments de cette ligne : on a l'adresse de l'élément d'indice i et j : c'est « $t+3*i + j$ »
- On utilise l'opérateur d'indirection *
- Soit $*(\text{int}^* t+3*i+j)$ et c'est $t[i][j]$

Conclusion : on a besoin de connaître le nombre de colonnes pour trouver un élément du tableau

- Qu'affiche le programme suivant

```
main() { int i,j;  int t[2][3];
  for (i=0; i<2; i++) for (j=0; j<3; j++) t[i][j]=i+j;
  printf("\nAdresse du tableau %p \n",t);
  printf("\nAdresse des lignes du tableau \n");
  for (i=0; i<2; i++) printf("%p %p",&t[i], t+i);
  printf("\nAdresse des elements du tableau \n");
  for (i=0; i<2; i++) for(j=0;j<3;j++) printf("%p ",&t[i][j]);
}
```



Fonction et tableau nD

- Les paramètres de fonctions qui sont des tableaux de N dimensions doivent préciser N-1 dimensions dans le prototype et l'entête de fonction

Exemple

```
#include <stdio.h>
#define N_COL 5
```

```
void init_tab(int tab[][N_COL], int nl)
{ int i,j;
  for (i=0; i<nl; i++) for (j=0; j<N_COL; j++) tab[i][j]=i+j;
}
```

```
int main() { int ligne,colonne;
  int mon_tab[3][N_COL]; int tab2[3][N_COL+1];
  init_tab(mon_tab,3);
  init_tab(tab2,3);
  for (ligne=0; ligne<3; ligne++) {
    for (colonne=0; colonne<N_COL; colonne++)
      printf("%d ",mon_tab[ligne][colonne]);
    printf("\n");
  }
}
```

Préciser le nombre de lignes n'est pas utile
Préciser le nombre de colonnes est INDISPENSABLE
Et c'est une constante

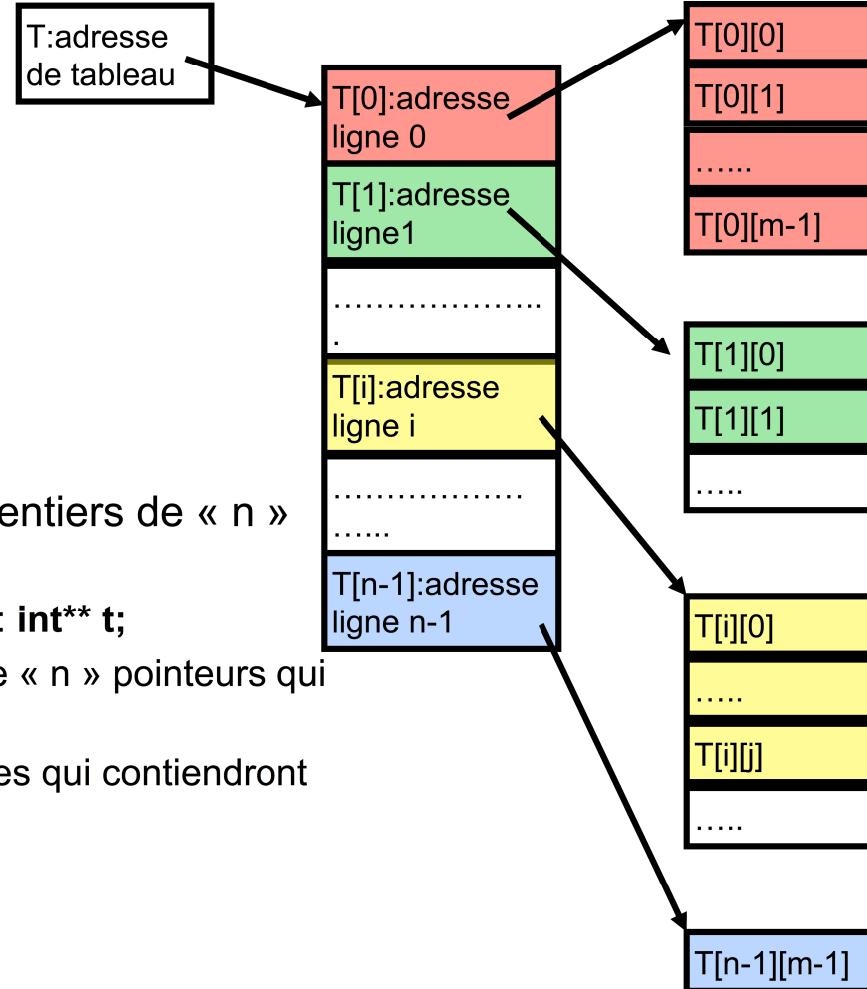
OK pour mon_tab, qui a NB_COL colonnes

INTERDIT pour tab2
qui a NB_COL+1 colonnes

Tableau nD

- Ces tableaux sont peu utilisés, car, pour des matrices 2D par exemple
 - Ils ont un nombre de colonnes fixes
 - les fonctions ayant des matrices en paramètres dépendent du nombre de colonnes : il faut faire autant de fonctions que de taille de tableaux !!!
- On utilise de préférence des tableaux 1D ou les pointeurs et l'allocation dynamique

Matrice 2D dynamique



- Construction dynamique d'une matrice d'entiers de « `n` » lignes et « `m` » colonnes

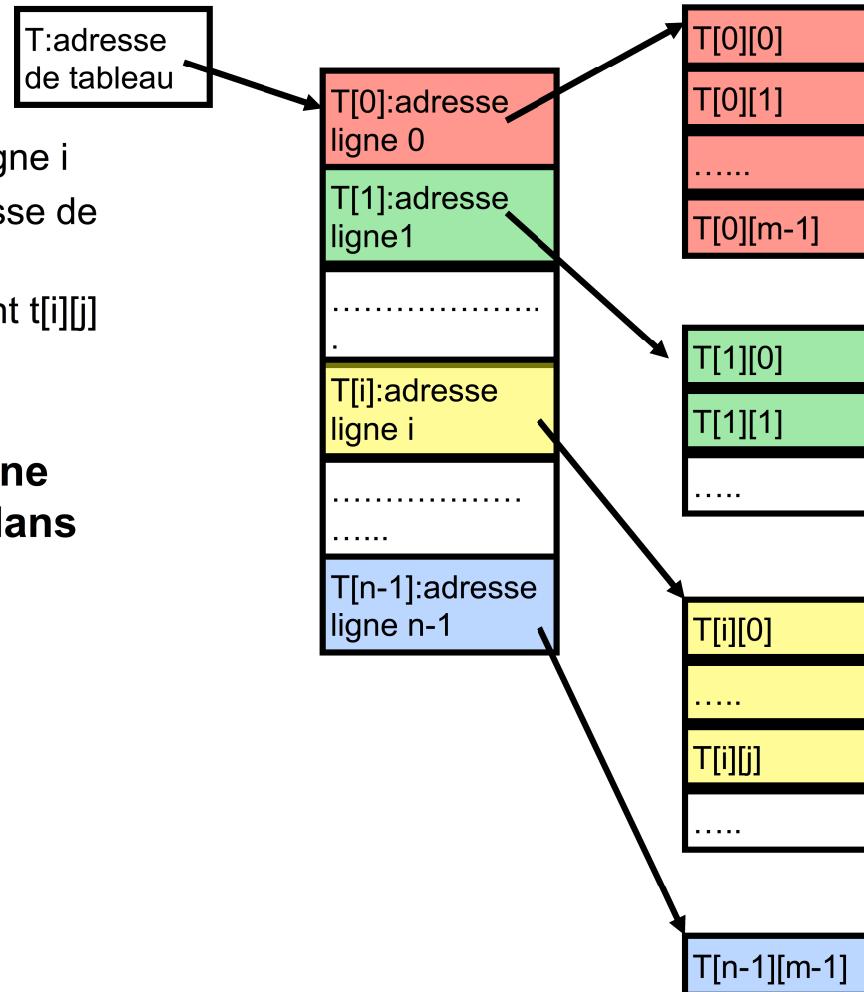
- Il faut déclarer une matrice sous la forme : `int** t;`
- Il faut créer dynamiquement un tableau de « `n` » pointeurs qui contiendront chacun l'adresse d'une ligne
- Il faut créer dynamiquement les « `n` » lignes qui contiendront chacune « `m` » éléments

Matrice 2D dynamique

Comment se retrouve un élément $t[i][j]$

- $t[i]$ ou bien $*(t+i)$ contient l'adresse de la ligne i
- $t[i]+j$ ou bien $(*(t+i)+j)$ contient donc l'adresse de l'élément d'indice i,j
- $*(t[i]+j)$ ou bien $*(*(t+i)+j)$ est donc l'élément $t[i][j]$

Conclusion : aucune dimension n'est nécessaire pour trouver la valeur de l'élément, les prototypes de fonctions ne doivent pas spécifier les dimensions dans ce cas



Matrice 2D

✓ Exemple : alloc4.c

```
#include <stdio.h>
int ** alloue(int n, int m) { int i,j;
    int **p ;

    p=malloc(n,sizeof(*p));
    if (p==NULL) return NULL;
    else
        for (i=0 ; i<n ; i++) {
            p[i]=malloc(m,sizeof (**p));
            if (p[i]== NULL) return NULL;
        }
    return p;
}

void destructureur(int **p, int n, int m){
    int i;
    for (i=0; i<n; i++) free(p[i]);
    free(p);
}
```

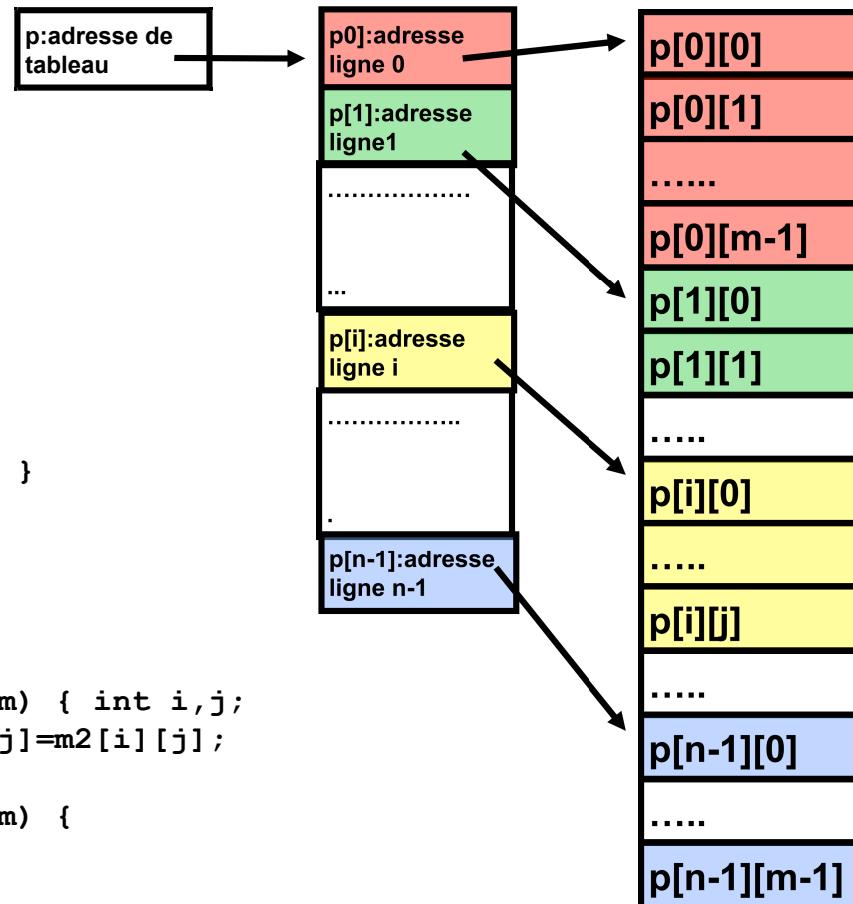
```
void aff(int **m, int nl, int nc) {
    int i,j;
    for (i=0; i<nl; i++){
        for (j=0;j<nc;j++)
            printf("%d ",m[i][j]);
        puts("");
    }
}
main() { int a,b,i,j ;
    int** mat ;
    puts ("dimension ? ");
    scanf("%d %d",&a,&b );
    mat= alloue(a,b);
    for (i=0; i<a; i++)
        for (j=0;j<b;j++)
            mat[i][j]=i+j;
    aff(mat,a,b) ;
    destructureur(mat,a,b) ;
}
```

Matrice 2D : plus fort

- Toutes les lignes sont allouées en une fois et sont donc contigues
- Il faut calculer les adresses des lignes
- Exemple : alloc5.c

```
int ** alloue(int n, int m) { int i,j;
    int ** p;

    p=(int **)calloc(n,sizeof(*p));
    if (p==NULL) return NULL;
    else {
        *p = (int *)calloc(n*m,sizeof(**p));
        if (*p ==NULL) [ free(p); return NULL; ]
        else
            for (i=1 ; i<n ; i++) p[i]=p[i-1]+m;
    }
    return p;
}
void copie1(int** m1, int** m2, int n, int m) { int i,j;
    for (i=0;i<n;i++) for(j=0;j<m;j++) m1[i][j]=m2[i][j];
}
void copie2(int** m1, int** m2, int n, int m) {
    memcpy(*m1,*m2,n*m*sizeof(**m1));
}
```



Des lignes de taille différentes

- Comment récupérer les arguments de la ligne de commande UNIX
- Le premier paramètre de main (argc) est le nombre de mot, le suivant (argv) est le tableau de ces mots
 - Attention :ce sont des chaînes de caractères : conversion à faire si vous avez des nombres
- Main n'a que 2 versions : sans paramètres ou avec 2 paramètres
- Sur le même principe, on peut faire des matrices avec des lignes de longueur variable

```
main (int argc , char** argv) {  
    int i;  
    if (argc == 1)  
        printf("Pas d'arguments\n");  
    else {  
        printf("%d arguments\n" , argc);  
        for (i=1; i<argc; i++)  
            printf("%s\n", argv[i]);  
    }  
} /* argv[0]: nom du programme */
```



```
pouilly2:exemples desvignes$ ./a.out  
Pas d'arguments  
pouilly2:exemples desvignes$ ./a.out un deux  
3 arguments  
un  
deux  
pouilly2:exemples desvignes$ ./a.out un deux trois quatre  
5 arguments  
un  
deux  
trois  
quatre  
pouilly2:exemples desvignes$
```

Exercices



- Comment déclarer un tableau comportant 3 dimensions (128, 256, 256)
- Faire une fonction qui libère la matrice 2D contiguë
- Faire une fonction allouant dynamiquement ce tableau
- Faire un programme principal utilisant cette fonction
- On suppose avoir un fichier texte contenant les titres des livres de la bibliothèque de l'école, à raison d'un titre par ligne. Il y a 250000 livres, la longueur maximale d'un titre est 256octets, la longueur moyenne 32 octets.
 - Comment déclarer un tableau t1 (non alloué dynamiquement) pouvant contenir ces titres ?
 - Comment déclarer un tableau t2 (qui sera alloué dynamiquement) pouvant contenir ces titres ?
 - Comparer les tailles qui seront utilisées par ces deux versions de tableau.
 - Faire une fonction réalisant la lecture des titres dans le tableau t2 Cette fonction retournera le tableau t1.
 - La fonction fgets(tableau, 256, f1) permet de lire une ligne du fichier f1, et de la stocker dans le tableau déjà créé.
 - La fonction strcpy(s1,s2) permet la copie de s2 dans s1.
 - On suppose avoir une fonction int nblivres(char* fichier) qui donne le nombre de livres contenus dans le fichier.

Complexité



ZOUZEL
POURQUOI FAIRE SIMPLE
QUAND ON PEUT FAIRE
COMPLIQUÉ ?!

Complexité (1)

- Combien de temps peut prendre mon programme ?
- Se termine-t-il quel que soit le nombre de données traitées ?
- Coût d'un algorithme
 - taille mémoire
 - temps d'exécution
- Complexité : regrouper dans une mesure intuitive ces informations sur le comportement d'un programme en fonction du volume des données traitées.
 - Complexité spatiale : taille mémoire
 - Complexité temporelle : définir ici les opérations coûteuses
 - Complexité pratique : programme
 - Complexité théorique : algorithme
 - Complexité
 - moyenne
 - dans le pire des cas
 - dans le meilleur des cas

Complexité (2)

Exemple : un tri

On cherche le maximum du tableau

On l'échange avec le dernier élément du tableau pas encore trié

On recommence sur les éléments non triés

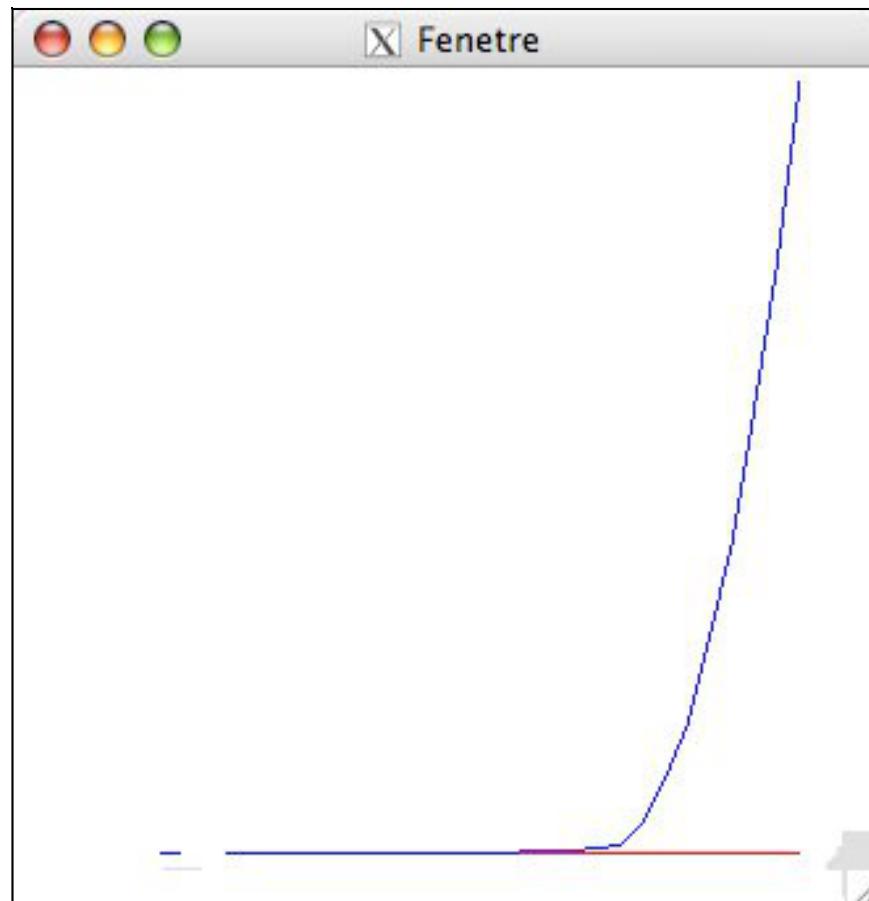
```
int triselection(int* tab, int n) { int i,j, imax;  
    for (i=0; i<n-1; i++) {imax=0;  
        for (j=1; j<n-i; j++)  
            if (tab[j]>tab[imax]) imax=j;  
        swap(&tab[n-i-1],&tab[imax]);  
    }
```

Nombre de comparaisons en fonction de n : ??

Nombre d'échanges en fonction de n : ??

Complexité (3)

- Exemple :
- le temps d'exécution en fonction du nombre de données triées
 - tri par sélection en bleu
 - du tri heapsort en rouge



Complexité (4)

Mesure utilisée

- Comportement asymptotique
 $o(n)$, $o(n^2)$, $o(n \log(n))$...

n=	1	2	4	8	16	32
log	0	1	2	2	4	5
n	1	2	4	8	16	32
$n \log n$	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296

Conséquence :

- avoir une idée de la complexité pratique
- algorithme parfois impossible à utiliser
- toujours un problème même si les machines sont de plus en plus rapides, car les volumes de données traitées augmentent

Récurrence, Récursivité



Récursivité (1)

- Définition
 - **objet informatique autoréférent**
- Exemples
 - Factorielle : $n! = n \times (n-1)!$
 - Factorielle (n)= $n \times$ Factorielle($n-1$)
- Pourquoi n'est ce pas correct ?
- Programmation récursive
 - Réduire le problème initial à un problème de taille inférieure
 - Trouver la formule de récurrence i.e le cas général
 - Préciser le cas connu
 - Taille du problème :
 - Paramètre n décroissant
 - Décomposition du cas général
 - $\text{fac}(n)=n*\text{fac}(n-1)$
 - Cas particulier et arrêt
 - $\text{fac}(1) = 1$

```
int fac(int n) {  
    return(n * fac(n-1));  
}
```

Récursivité (2)

```
/* recur1.c */
#include <stdio.h>

int fac(int n) { int y;
    printf("Entrée dans factorielle avec n = :%d\n", n);
    if(n>1) y=n*fac(n-1);
    else y=1;
    printf("Fin de l'etape %d; valeur %d\n", n, y);
    return y;
}
main() { int a, res;
    puts("Entrer un nombre"); scanf("%d", &a);
    res = fac(a);
    printf("Factorielle de %d = %d \n", a, res);
}
```

Récursivité (3)

```
main() { r=f 24 }
```

```
if(4>1) y=4*: 6  
return y;
```

```
if (3>1) y=3*: 2  
return y;
```

```
if(2>1) y=2*: 1  
return y;
```

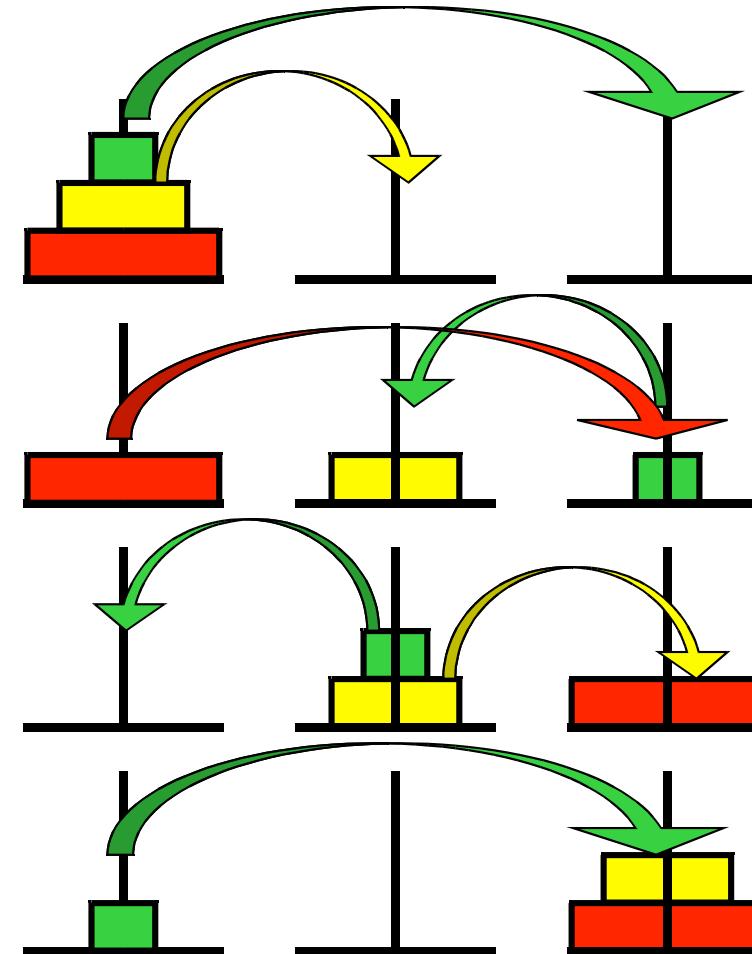
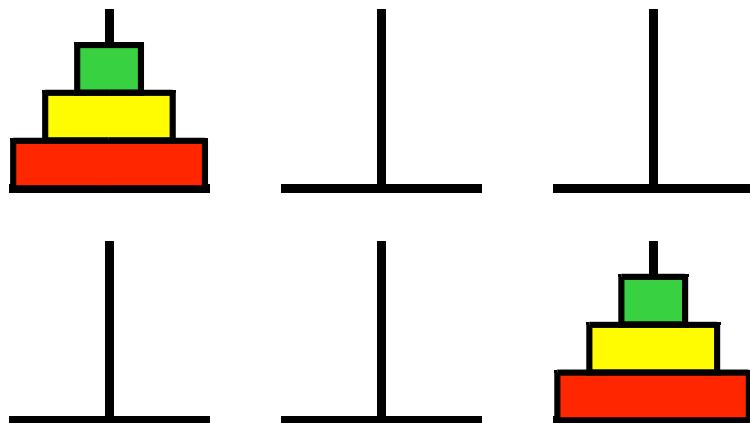
```
if (1>1) y=1;  
return(y)
```

```
int fac(int n) { int y;  
if(n>1) y=n*fac(n-1);  
else y=1;  
return y;  
}
```

```
r 24
```

Récursivité (4)

- Tours de Hanoï : transfert de N disques de tailles décroissantes entre 2 piquets A et B en respectant les tailles décroissantes et en utilisant un piquet intermédiaire
- Combien de solutions ?



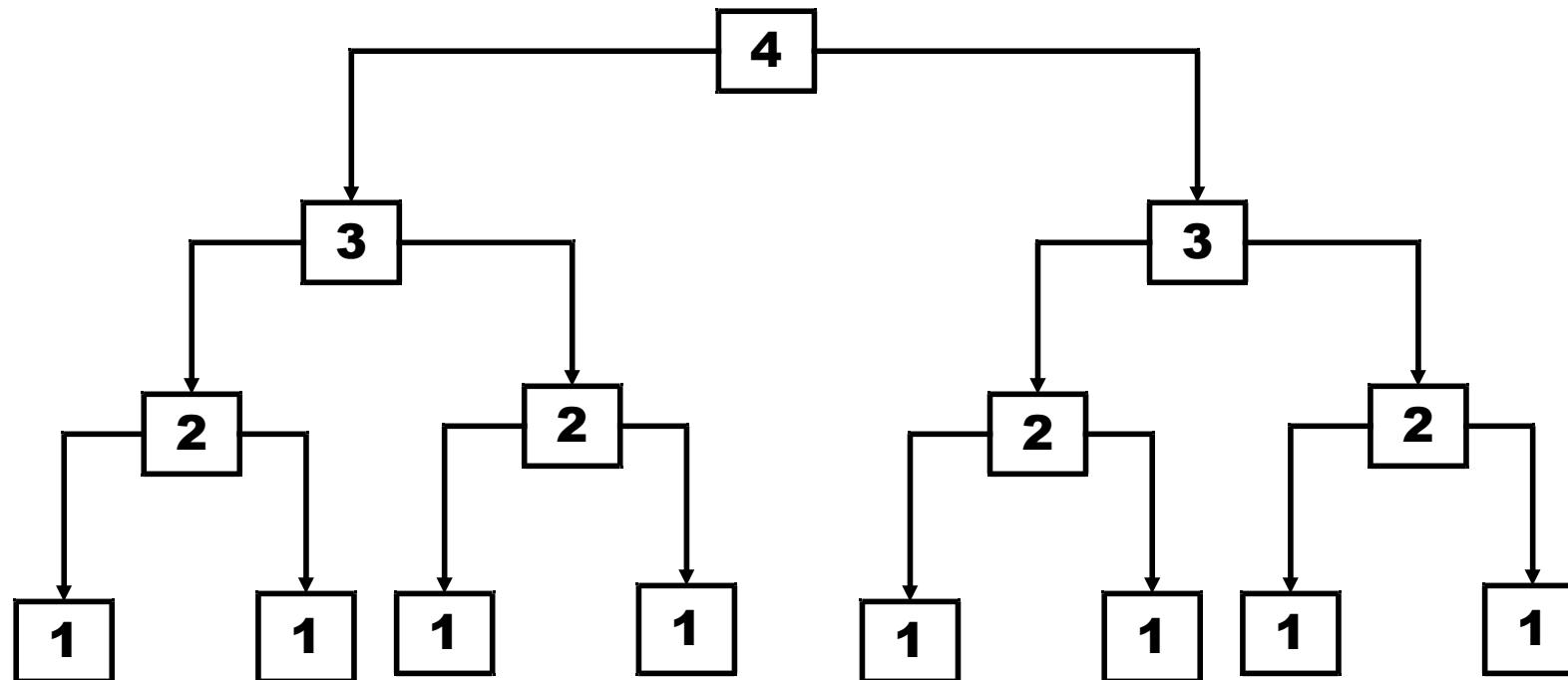
Récursivité (5)

- | paramètre : N
- | cas trivial : transférer 1 disque
- | cas général : déplacer N disques de A vers B
 - | déplacer N-1 disques de A vers C
 - | transférer 1 disque de A vers B
 - | déplacer N-1 disques de C vers B
- | Paramètres ; N,A,B,C

```
void hanoi(int n, int dep, int arr, int inter) {  
    if (n==1) printf( "deplace 1 disque de %d vers %d",dep,arr);  
    else {  
        hanoi(n-1,dep,inter,arr);  
        printf("transfere 1 disque de %d vers %d\n",dep,arr);  
        hanoi(n-1,inter,arr,dep);  
    }  
}  
main() { int a;  
puts("Entrer le nombre de disques\n"); scanf("%d", &a);  
hanoi(a,1,2,3);  
}
```

Récursivité (6)

- Est-il possible de programmer Hanoï sans récursivité ?
- Quel est le nombre d'appels récursifs ?



Récursivité (7)



- Simplicité de programmation
 - rapide à écrire
- Coût
 - mémoire : pile
 - passage des paramètres
 - création variables locales
 - CPU
 - appel de fonctions
 - sauvegarde de « l'état précédent »
- Conclusion
 - Utilisation à bon escient
 - à éviter pour les problèmes non récursifs

Récursivité croisée (1)

- Sous programme SP1 appelant SP2
- Sous programme SP2 appelant SP1
- Exemple : le baguenaudier
 - n pions à placer sur les n cases consécutives d'une tablette
 - Ce sont les codes de Gray
- Règles
 - "jouer un coup" signifie poser ou enlever un pion
 - soit sur la première case
 - soit sur la case suivant la première case occupée
- Exemple 1 : coups possibles
 - enlever le pion de la case 1
 - enlever le pion de la case 2
- Exemple 2 : coups possibles
 - poser le pion dans la case 1
 - poser le pion dans la case 4
- Exemple 3 : coups possibles
 - enlever le pion dans la case 1
 - poser le pion dans la case 2

o	o			o	o		
1	2	3	4	5	6	7	8

		o		o	o		
1	2	3	4	5	6	7	8

o		o	o	o			
1	2	3	4	5	6	7	8

■ Comment faire pour placer n points

o	o	o	o	o			
1	2	3	4	5	6	7	8

Récursivité croisée (2)

Comment faire pour placer le point 6 ?

o	o	o	o	o			
1	2	3	4	5	6	7	8

Vider les 4 premiers

- On ne sait pas comment, mais tant pis

				o			
1	2	3	4	5	6	7	8

Poser le 6ième

- On sait comment,

				o	o		
1	2	3	4	5	6	7	8

Remettre les 4 premiers

- On ne sait pas comment, mais tant pis

o	o	o	o	o	o		
1	2	3	4	5	6	7	8

Conclusion

- On a transformé un problème de rang n en 2 problèmes de rang n-2
- On sait faire le cas n=0 ou n=1

Donc on sait que le problème a une solution

Récursivité croisée (3)

- Pour poser n pions, si n-1 pions sont posés :
 - vider la tablette des n-2 premiers pions
 - poser le n^{ième}
 - remplir à nouveau la tablette avec les n-2 premiers pions,
- Algorithme 1 : Remplir les n premiers points
 - SSP REMPLIR (tab,n)
 - Si n>1 alors
 - REmplir (tab, n-1)
 - VIDER (tab, n-2)
 - POSER (tab,n)
 - REmplir (tab, n-2)
 - Sinon si n==1 POSER(tab,1)
- Algorithme 2 : Vider les n premiers points
 - SSP VIDER (tab,n)
 - Si n>1 alors
 - VIDER (tab, n-2)
 - ENLEVER(tab, n)
 - REmplir (tab,n-2)
 - VIDER(tab, n-1)
 - Sinon si n==1 ENLEVER(tab,1)

Récursivité : pour en savoir plus



Récursivité : le sudoku

8	2	5	1	5	4	9	7	3
	5	3	2	9	6	1		
			6	5	4			
6		7	4	8	2	3	1	
3	1	7	2	6	5		8	
3	2							
7	8	5	3	2	1			
9	5			7		2	4	

8	2	6	1	5	4	9	7	3
7	4	5	3	2	9	6	1	8
1	9	3	8	7	6	5	4	2
5	6	9	7	4	8	2	3	1
2	8	4	9	1	3	7	6	5
3	1	7	2	6	5	4	8	9
6	3	2	4	9	1	8	5	7
4	7	8	5	3	2	1	9	6
9	5	1	6	8	7	3	2	4

Récursivité : le sudoku

- si le sudoku est plein, retourner GAGNE
- sinon
 - recherche de la meilleure case a jouer : i,j
 - si aucune case possible,
 - alors pas de solution, retourner PERDU
 - sinon
 - tant qu'il y a des valeurs possibles pour cette case
 - Mettre une valeur possible dans le sudoku en i,j
 - trouver la solution du sudoku avec une case inconnue de moins.
 - Si on a trouvé une solution, retourner GAGNE
 - sinon remettre la case i,j à vide et tester une autre valeur
 - retourner PERDU // Si on arrive ici, aucune solution pour la case i,j

Récursivité : le sudoku

```
int resolution(char tab[][][DIM2]) { int i,j,k,l;
    char choix[DIM2];
    /* Quelle est la meilleure case à jouer ? */
    if ( (k=meilleur_choix(tab,&i,&j))==DIM2+1) return GAGNE;
    /* Si k=0, pas de possibilités de jeu : échec de cet essai */
    if (k==0) return IMPOSSIBLE;
    /* On met dans le tableau choix tous les coups possibles */
    liste_choix(tab,i,j,choix);
    /* On teste toutes les possibilités de jeu jusqu'à trouver une solution */
    for (l=0; l<k; l++) {
        tab[i][j]=choix[l];
        /* Appel de la fonction de résolution en ayant mis un chiffre dans la case i,j.
        Si on gagne, on arrête, sinon, on essaye une autre possibilité */
        if (resolution(tab)==GAGNE) return GAGNE;
        else tab[i][j]=VIDE;
    }
    /* Ici, tous les essais pour la case i,j n'ont pas donné de solution */
    return IMPOSSIBLE;
}
```

Exos : Récursivité

- Exo 1 : Combinaison p éléments parmi n: $C(p,n)=C(p-1,n-1)+C(p,n-1)$ avec $C(0,n)=C(n,n)=1$ et $0 \leq p \leq n$.
- Exo 2 : inverser l'ordre des éléments d'un tableau de nombres.
- Exo 3 : Palindrome : déterminer si une chaîne de caractère est un palindrome par la méthode suivante : un mot est un palindrome si sa première lettre est identique à sa dernière et si la sous chaîne $2..n-1$ est un palindrome
- Exo 4 : Anagrammes : afficher les anagrammes d'un mot par la méthode suivante : on prend un mot et on permute sa première lettre avec chacune des autres lettres, y compris elle-même. Ensuite pour chacun des nouveaux mots obtenus, on fait les permutations entre la deuxième lettre et toutes les autres lettres, etc.
- Exo 5 : les 8 reines : comment placer 8 reines sur un échiquier sans qu'elles soient en prises ? On essaye de placer une reine sur la première ligne libre, sans quelle soit en prise. Puis on essaye de placer n-1 reine sur les n-1 lignes suivantes.