INF111

Initiation à la programmation impérative en C

http://ama.liglab.fr/~amini/Cours/L1/INF111/

Massih-Reza Amini - Jean-Baptiste Orfila

Université Joseph Fourier
Laboratoire d'Informatique de Grenoble
Massih-Reza.Amini@imag.fr
jbaptiste.orfila@gmail.com

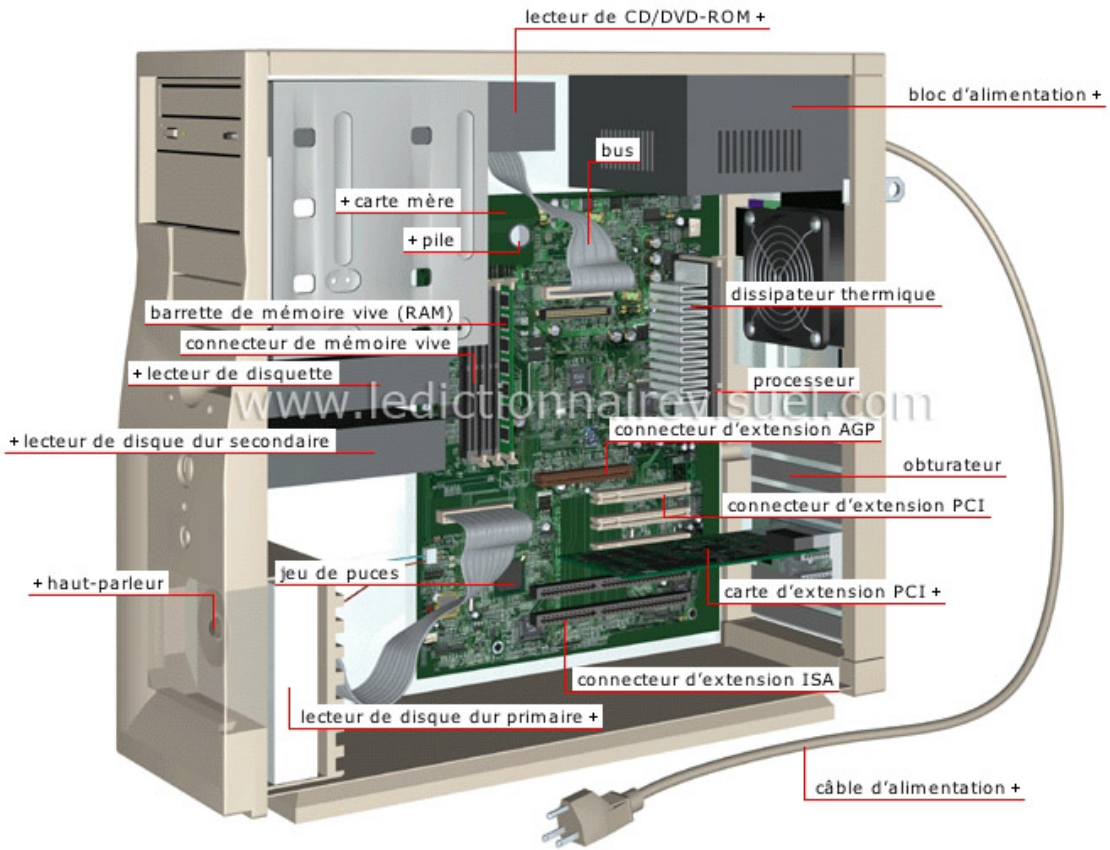
UNIVERSITE JOSEPH FOURIER
SCIENCES.TECHNOLOGIE.SANTÉ

UNIVERSITE JOSEPH FOURIER
SCIENCES.TECHNOLOGIE.SANTÉ

Ordinateur

UNIVERSITE JOSEPH FOURIER
SCIENCES. TECHNOLOGIE. SANTÉ

Ordinateur



Histoire de C

- Le langage C est développé en 1972 par Dennis Ritchie dans les laboratoires *Bell Systems*
  - Le langage est dérivé du langage B de Ken Thompson, qui lui même était basé sur le langage BCPL, dévéloppé par Martin Richards.
- Plusieurs systèmes d'exploitation sont écrits en C, et ce langage était *de facto* fourni avec le système UNIX 5
  - *The C programming Language*, Brian Kernigham and Dennis Ritchie, Prendtice-Hall 1978
- En 1983, ANSI crée un groupe pour standardiser C
  - ANSI C est finalisé en 1989 et ISO l'adopte en 1990.

Popularité de C

Sep 2015	Sep 2014	Change	Programming Language	Ratings	Change
1	2	↑	Java	19.565%	+5.43%
2	1	↓	C	15.621%	-1.10%
3	4	↑	C++	6.782%	+2.11%
4	5	↑	C#	4.909%	+0.56%
5	8	↑	Python	3.664%	+0.88%
6	7	↑	PHP	2.530%	-0.59%
7	9	↑	JavaScript	2.342%	-0.11%
8	11	↑	Visual Basic .NET	2.062%	+0.53%
9	12	↑	Perl	1.899%	+0.53%
10	3	⇊	Objective-C	1.821%	-8.11%
11	29	⇈	Assembly language	1.806%	+1.22%
12	13	↑	Ruby	1.783%	+0.50%
13	15	↑	Delphi/Object Pascal	1.745%	+0.59%
14	14		Visual Basic	1.532%	+0.26%
15	17	↑	Pascal	1.298%	+0.40%
16	18	↑	Swift	1.188%	+0.34%
17	19	↑	MATLAB	1.181%	+0.36%
18	20	↑	PL/SQL	1.082%	+0.27%
19	21	↑	R	1.045%	+0.24%
20	31	⇈	COBOL	0.994%	+0.42%

source:http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

UNIVERSITE JOSEPH FOURIER
SCIENCES.TECHNOLOGIE.SANTÉ

Structure d'un programme C

Informe le compilateur que les fonctions standard I/O seront utilisées

Une paire /* */ définit une zone de commentaires, ignorée par le Compilateur.

```
#include <stdio.h>

/* Zone de commentaires
   non interprétée par le
   compilateur */

int main ()
{
   float a;

   printf("Entrez un nombre réel\n");
   scanf("%f", &a);
   printf("Le carré de %f est %f\n",a,a*a);

   return(1);
}
```

Sortie écran

Saisie depuis l'écran

Paire d'accolades définissant un bloque de codes

Variable locale à la fonction main()

Tout programme C contient nécessairement la fonction main, dans ce cas int main() ne prend pas de paramètres et retourne un entier

Organisation et Evaluation

Organisation

- 12 semaines de cours
  - Les 6 premières : TD (1.5h) / TD (1.5h) / TP (3h) - MRA/JBO
  - Les 6 dernières : TD (1.5h) / TD (1.5h) / TP (1.5h) - JBO/MRA

Calcul de la note finale

Note finale = $0.6 * \text{Note(Examen final)} + 0.2 * \text{Note(CC1)} + 0.2 * \text{Note(CC2)}$

- Examen final: épreuve sur table de 3h (avant le 19 Décembre)
- Contrôle continu 1 (CC1) épreuve sur table de 2h (vers le 2 novembre)
- Contrôle continue 2 (CC2) une moyenne sur
  - deux contrôles de TD (15 à 20 minutes chacun),
  - 1 note de TP
  - 1 note de projet
  - Présence - participation en TD/TP

Types de données en C

Type de donnée	Signification	Taille en octet	Plage de valeurs acceptée
char	Caractère	1	-128 à 127
unsigned char	Caractère non-signé	1	0 à 255
int	Entier	2 (pro. 16 bits)	$-32\,768$ à 32 767
		4 (pro. 32 bits)	$-2\,147\,483\,648$ à 2 147 483 (
unsigned int	Entier non signé	2 (pro. 16 bits)	0 à 65 535
		4 (pro. 32 bits)	0 à 4 294 967 295
long int	Entier long	4	$-2\,147\,483\,648$ à 2 147 483 (
unsigned long int	Entier long non signé	4	0 à 4 294 967 295
float	Flottant (réel)	4	$3.4e^{-38}$ à $3.4e^{38}$
double	Flottant double	8	$1.7e^{-308}$ à $1.7e^{308}$
long double	Flottant double long	10	$3.4e^{-4932}$ à $3.4e^{4932}$

- La taille des *types* dépend de la machine, pour la connaître **sizeof(type)**

Opérateurs en C

Type	Opérateur	Action
Arithmétique	$-$	Soustraction
	$+$	Addition
	$*$	Multiplication
	$/$	Division
	$\%$	Modulo
	$--$	Décrémentation (par 1)
	$++$	Incrémentation (par 1)
	$+=$	Augmentation ($a+=b$ vaut dire $a=a+b$)
	$-=$	Diminution ($a-=b$ vaut dire $a=a-b$)
Relationnelle	$>$	Strictement plus grand que
	$>=$	Plus grand ou égal que
	$<$	Strictement plus petit que
	$<=$	Plus petit ou égal que
	$==$	Egal à
	$!=$	Différent de
Logique	&&	ET
	||	OU
	!	NON

Expressions booléennes

- Soit C et c deux variables de type caractères, quelle est l'expression booléenne qui teste si c est une lettre minuscule de l'alphabet $c \in \{'a', \ldots, 'z'\}$ et que C est la lettre majuscule de c?
- Quelle est l'expression booléenne qui teste si l'entier n est un multiple de 3 ou de 5?
- Quelle est l'expression booléenne qui teste si le réel x vérifie l'inégalité

$$|x| > 3$$

- Quelle est l'expression booléenne qui teste si un caractère n'est pas une lettre de l'alphabet?

Déclaration et initialisation des variables

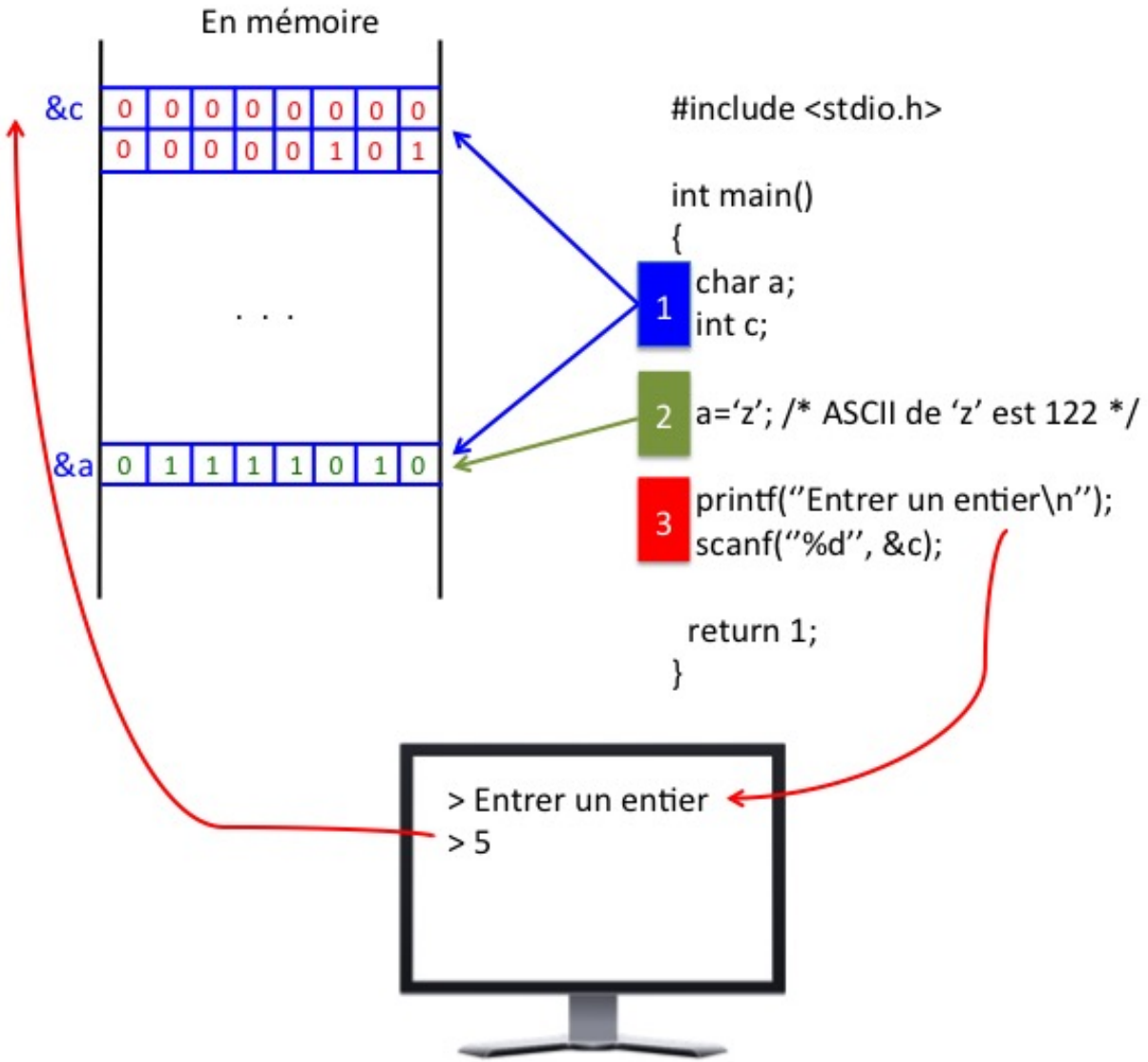
- Les variables doivent être déclarées avant toute utilisation
  - Cette déclaration doit précéder la première instruction dans un bloc
- Les variables déclarées dans un bloc sont locales à ce bloc
  - Ils ne peuvent pas être accédés en dehors du bloc
- Les variables peuvent être initialisées au moment de leur déclaration ou après dans le bloc.

```
 #include <stdio.h>
int l; // La variable l est globale au programme
int main ()
{
   int i=1, j; // Les variables i et j sont locales à la fonction main
   j=0;

   instruction;
}
```

UNIVERSITE JOSEPH FOURIER
SCIENCES.TECHNOLOGIE.SANTÉ

Déclaration et initialisation des variables (2)



Structures de contrôle: structure conditionnelle if

- La structure conditionnelle *if* permet de réaliser un test et d'exécuter une instruction ou non selon le résultat de ce test. Sa syntaxe est la suivante :

```
if(test1) {
   instruction;
}
else if(test2) {
   instruction;
}
else
{
   instruction;
}
```

Exemple

```
if(b*b-4*a*c>0){
   printf(" Deux solutions réelles distinctes \n");
}
else if(b*b-4*a*c==0){
   printf(" Une seule solution \n");
}
else{
   printf(" Pas de solutions réelles \n");
}
```

Structures de contrôle: structure conditionnelle if

Exemple

1. Nous considérons le programme suivant :

```
#include <stdio.h>
int main()
{
  int a, b;
  printf("Entrez deux nombres entiers \n");

  scanf("%d", &a);
  scanf("%d", &b);
}
```

Complétéz ce programme pour qu'il affiche trois messages différents selon que *a* est supérieur à *b*, *b* est supérieur à *a* ou les deux sont égaux.

2. Ecrivez un programme qui affiche le maximum de trois nombres *a*, *b*, et *c* saisis par l'utilisateur.

3. Ecrivez un programme qui étant donnés deux entiers positifs saisis par l'utilisateur représentant les rayons de deux disques concentriques affiche l'aire de la surface de l'anneau entre ces deux disques.

Structures de contrôle: structure conditionnelle if

1. Ecrivez un programme qui étant donnés trois entiers *a*, *b*, et *c* saisis par l'utilisateur affiche les trois entiers dans l'ordre croissant.

2. Ecrivez un programme qui affiche si la lettre saisie par l'utilisateur est voyelle ou consonne.

3. Ecrivez un programme qui affiche la valeur absolue de la différence de deux nombres réels saisis par l'utilisateur.

4. Ecrivez un programme qui demande un réel représentant le ph d'une eau puis affiche son acidité selon *très acide* si le ph est dans $[0, 2[$, *acide* s'il est dans $[2, 7[$, *neutre* s'il vaut 7, *basique* s'il est dans $]7, 12[$ et *très basique* sinon. On suppose que l'utilisateur saisie toujours une valeur positive.

Structures de contrôle: boucle for

- La boucle **for** définit une instruction d'initialisation, une condition de terminaison et un bloc d'instructions exécuté à chaque fin de boucle. La condition de terminaison est testée à chaque nouvelle boucle. La syntaxe du boucle **for**:

```
for(initialisation; cond.  entrée; opération sur la var.  de contrôle)
{
    Corps de la boucle;
}
```

Exemple

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

```
int main(){
  int i;
  for (i = 100; i != 65; i-=5 ){
    printf ("%d \n", i * i);
  }
}
```

Structures de contrôle: boucle for

- Pour écrire une instruction de boucle, se poser toujours ces 5 questions :
  - Quelle est la variable de contrôle de la boucle?
  - Quelles sont les actions à répéter à chaque tour?
  - Quelle est la valeur initiale de la variable de contrôle?
  - Comment faire évoluer la variable de contrôle après chaque tour?
  - Quelle est la condition d'entrée dans la boucle?

Exemple

- La factorielle du nombre n, est définie par $n! = 1 \times 2 \times ... \times (n-1) \times n$. Par convention $0! = 1$. Ecrivez un programme qui calcule la factorielle d'un nombre n dont la valeur sera saisie par l'utilisateur.
- Ecrivez un programme qui calcule le terme de rang n de la suite $U_n = a \times U_{n-1}$ avec $U_0 = 5$ et les valeurs de n et a sont saisies par l'utilisateur.

Structures de contrôle: boucle for

1. Ecrivez un programme qui affiche tous les diviseurs d'un entier N positif saisi par l'utilisateur.

2. Ecrivez un programme qui calcule le terme de rang n de la suite $U_n = aU_{n-1} + bU_{n-2}$ avec $U_0 = 1$, $U_1 = 2$, $a = 5$ et $b = 10$.

3. Ecrivez un programme qui demande à l'utilisateur de saisir une valeur entière `nbNotes`. Le programme demande ensuite à l'utilisateur la saisie de `nbNotes` valeurs, chaque valeur étant un entier compris entre 0 et 20 représentant une note, et puis calcule la moyenne des notes.

4. Ecrivez un programme qui demande un nombre entier et qui affiche un triangle rectangle isolcèle avec le caractère '*'.

Structures de contrôle: boucle for

1. Ecrivez un programme qui calcule la valeur d'un polynôme de degré n pour une entrée x_0 donnée.

$$\overbrace{\underbrace{\underbrace{\underbrace{a_n}_{*x_0 + a_{n-1}}}_{*x_0 + a_{n-2}}}_{\dots}}^{*x_0 + a_0}$$

2. Écrire la totalité d'un programme C, qui demande à l'utilisateur un entier n et qui calcule la somme des n premiers termes de la série harmonique

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

3. Ecrire un programme qui affiche la table de multiplication des nombres 1 à 10.

Structures de contrôle: boucle while

La boucle **while** exécute une instruction ou un bloc d'instructions spécifié, *tant que* une condition donnée en entrée est vraie. La syntaxe du boucle **while**:

```
 while(condition)
{
    Corps de la boucle;
}
```

1. Ecrivez un programme qui permet de déterminer si un entier positif *M* saisi à l'écran est un nombre premier. Le programme affichera à l'écran *M est premier* ou *M n'est pas premier* selon la valeur d'une variable booléenne `premier` à l'issue de la boucle.

2. Modifiez le programme pour qu'il affiche la liste de tous les nombre premiers inférieurs ou égaux à un nombre `MAX` saisi par l'utilisateur.

Structures de contrôle: boucle while

1. Soit la suite définie par $U_0 = 1$ et $U_n = 3 * U_{n-1} + 4$. Écrivez un programme qui calcule et affiche l'indice et le premier terme de la suite supérieur ou égal à une valeur `BORNE` saisie par l'utilisateur.

2. L'algorithme de la recherche dichotomiue pour trouver une approximation du zéro, noté par x_0, d'une fonction strictement monotone, f, ($f(x_0) = 0$) est la suivante: on calcule $f(a)$ et $f(b)$. Si $f(a)$ et $f(b)$ ne sont pas de signes opposés, alors il n'y a pas de zéro entre a et b. Sinon on détermine l'abscisse x du milieu du segment $[a, b]$ et on calcule $f(x)$. Si $f(x)$ est de même signe que $f(a)$ alors le zéro de la fonction est entre x et b. Si f est de même signe que $f(b)$ alors le zéro de la fonction est entre a et x. On arrête les itérations lorsque $|f(x)| < \epsilon$ avec ϵ positif et assez petit.

Structures de contrôle: boucle while

La constante d'Euler e peut-être calculée au moyen de la formule

$$e = \sum_{k=0}^{+\infty} \frac{1}{k!}$$

Ce calcul se prête facilement à une exécution itérative: partant de $e = 0, k = 0, terme = 1 (= \frac{1}{0!})$, on ajoute *terme* à e puis on incrémente k et on multiplie *terme* par $1/k$ avant de recommencer. Comme on ne va pas calculer une somme infinie, il faut décider à quel moment on stoppe le calcul. Puisque la somme converge vers e, l'idée est de s'arrêter lorsque deux valeurs consécutives de cette somme sont *suffisamment* proches. On fixe pour cela une valeur d'erreur, et on arrête le calcul lorsque ,

$$\frac{(e - e')}{e} \leq \epsilon$$

où e et e' représentent deux valeurs consécutives calculées pour la somme. Ecrivez un programme qui calcule la valeur de e avec une précision inférieure à $\epsilon = 10^{-3}$.

Tableaux

Un **tableau** de taille N est un ensemble de N variables de même type, les variables sont manipulées individuellement et elles sont désignées par un indice $[0, N-1]$

```
#include <stdio.h>
int main() {
      int i, indMax;
       float tab[10], eleMax;
       ... ; // Saisie des éléments de tab
       eleMax=tab[0];
      indMax=0;
      for(i=1; i<10; i++){
            if(tab[i] > eleMax){
                  eleMax=tab[i];
                  indMax=i;
            }
      }
      ... ;
      return 1;
}
```

Tableaux

- Initialisation d'un **tableau** de taille N.

```
int N= 5;
int Tab[N]= {3, 4, 5, 6, 1};
```

- On souhaite calculer la moyenne de $N = 5$ notes stockées dans un tableau de flottants, `notes`. Le tableau `coef` contient le coefficient associé à chaque note dans `notes`. La note finale est obtenue en calculant l'expression :

$$\text{note finale} = \frac{\sum_{i=0}^{N-1} c_i \times n_i}{\sum_{i=0}^{N-1} c_i}$$

Où c_i (resp. n_i) est le $i-$ième élément du tableau `coef` (resp. `notes`).

Tableaux

- Ecrire un programme qui teste si les éléments d'un tableau sont triés dans l'ordre croissant.
- Soient les tableaux suivants stockant les coordonnées de deux vecteurs $\vec{u}$ et $\vec{v}$

```
float u[5]={1.2, 4.1, 3.3, 2.6, -1.1};
float v[5]={3.3 2.1 -4.1 1.7 2.4};
```

  Ecrire un programme qui calcule le produit scalaire $\langle \vec{u}, \vec{v} \rangle$.
- Ecrire un algorithme qui calcule le plus grand écart dans un tableau (l'écart est la valeur absolue de la différence de deux éléments).

Tableaux de caractères: chaînes de caractères

- Les tableaux de caractères, ou chaîne de caractères, se terminent par le caractère '\0'.

```
char str[]="str est un tableau de caractères";
```

- Le code correspondant aux chaînes de caractères est %s.
- Ecrire un programme qui transforme la chaîne de caractères suivante
  `char tabchar[]="Ceci est un autre exemple de chaîne de caractères";`
  en remplaçant les minuscules par des majuscules et réciproquement, les caractères spéciaux ne seront pas changés. Après transformation, la chaîne modifiée sera affichée à l'écran.
- Ecrire un programme qui indique si une chaîne de caractères est un palindrome. Un palindrome est un mot qui se lit de la même façon à l'endroit et à l'envers (par exemple les mots *rever* et *radar* sont des palindromes).

Génération de nombres aléatoires

Les nombres aléatoires sont générés avec les fonctions **srand** et **rand**. La fonction **srand** prend une valeur en paramètre et s'en sert comme *graine*. C'est avec cette fonction que la fonction **rand** pourra renvoyer une séquence de nombres aléatoires. Si vous passez deux fois le même nombre à **srand**, la fonction **rand** renverra toujours la même séquence. Une idée est donc de faire passer en paramètre à **srand** la valeur du timestamp (s'incrémentant à chaque seconde), ainsi la *graine* ne sera jamais deux fois la même (sauf si on fait deux srand dans un intervalle inférieur à 1 seconde).

```
#include <stdlib.h>
#include <time.h>
int main() {
      int i;
      srand(time(NULL));
      for(i=1; i≤10; i++){
            printf("%d", rand()%100); // rand()%N un nombre aléatoire entre 0 et N-1
      }
      return 1;
}
```

Génération de nombres aléatoires

- Combien de fois faut-il jouer pour gagner au jeu de dé ? Ecrivez un programme qui demande à l'utilisateur de deviner le résultat d'un lancer de dé, puis lance le dé et vérifie si le joueur a bien deviné. Le jeu se continue juqu'à ce que le joueur gagne. Le programme affiche alors le nombre de tentatives du joueur.

- Ecrire un programme C qui
  1. initialise un tableau de 50 caractères avec les caractères 'a' ... 'z' générés uniformément,
  2. affiche le tableau
  3. calcule la plus petite lettre (ordre alphabétique) de l'alphabet présente dans le tableau et donne l'indice de sa première occurrence.
  4. affiche les lettres présentes, sans duplications
  5. affiche les lettres absentes de ce tableau
  6. affiche la fréquence d'apparition de chacune des lettres de l'alphabet
  7. affiche la lettre la plus fréquente

Tableaux : génération de nombres aléatoires

- On met le résultat de 10000 tirages aléatoires des entiers entre 0 à 3 dans le tableau, tab, (int tab[10000]). Ecrivez un programme qui calcule le nombre fois où chacun des entiers dans l'intervalle 0...3 a été tiré, et vérifier que **rand()** effectue un tirage aléatoire équiprobable.

- A partir de **rand()**, on veut faire un tirage non équiprobable. On veut par exemple que la valeur 0 représente 17% des cas, la valeur 1, 28%, la valeur 2, 50% et la valeur 3, 5%. Pour cela on considère un intervalle de 100 valeurs. Si la valeur tirée est comprise entre 1 et 17 alors la valeur affectée au résultat est 0, si la valeur tirée est comprise entre 18 et 45 (ce qui représente un intervalle de 28 valeurs) alros la valeur affectée au résultat est 1, si la valeur tirée est comprise entre 46 et 95 alors la valeur affectée au résultat est 2 et si la valeur est compromise entre 96 et 100 alors la valeur affectée au résultat est 3. Ecrivez un programme qui, à partir d'un tableau de N valeurs représentant les pourcentages de tirages d'un ensemble de N résultats, effectue un tirages selon ces pourcentages. Testez le programme sur 10000 tirages avec un ensemble de résultats = {0,...,N-1}

Tri par sélection

Sur un tableau de n éléments (numérotés de 0 à n-1), le principe du tri par sélection est le suivant :

1. rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 0 ;
2. rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 1 ;
3. continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

Tableaux 2D

```
#include <stdio.h>
#define nb_lignes 2
#define nb_colonnes 3
int main()
{
   int i,j;
   int Tab[nb_lignes][nb_colonnes] = {{1,2,3},{16,17,18}};
   for(i=0;i<nb_lignes; i++)
   {
      for(j=0;j<nb_colonnes; j++)
      {
         printf("%d ",Tab[i][j]);
      }
      printf("\n");
   }
   return(1);
}
```

Définition d'une fonction

`Type` *identificateur*`(Type-Param1` *Nom-Param1*`,Type-Param2` *Nom-Param2*`)`

Les fonctions en C ne permettent de passer des paramètres que par valeur; c'est-à-dire que la valeur passée comme paramètre est copiée à l'intérieur de la fonction puis est utilisée comme variable locale. Le résultat peut être transmis explicitement comme paramètre de retour.

```
#include <stdio.h>
int miseAuCarre(int x) // paramètre de retour de type entier
{
   int xSquare;
   xSquare=x*x;
   return(xSquare); // passage du paramètre en retour
}
int main()
{
   int nbEntree, nbCarre;
   printf("Entrez un nombre:  ");
   scanf("%d", &nbEntree);
   nbCarre=miseAuCarre(nbEntree);
}
```

Qu'affiche le programme suivant?

```
#include <stdio.h>
int f1(int a, int b) {
   return(a*b);
}
int f2(int x, int y) {
   int res;
   res=x+f1(x,y);
   return(res);
}
int f3(int u, int v,int w) {
   int res;
   res=f2(u,v)+f2(v,w);
   return(res);
}
int main()
{
   printf("%d\n", f3(3,2,4));
   return(1);
}
```

On considère la fonction `main` suivante

```
#include <stdio.h>
int main()
{
   int a,b,c;
   a=5;
   b=3;
   c=min2Int(a,b);
   printf("Le minimum de %d et %d est %d\n", a,b,c);
   return(1);
}
```

- Définissez la fonction `min2Int` appelée dans la fonction `main` précédente et qui retourne le minimum de deux entiers.
- En utilisant la fonction `min2Int` définie à la question précédente. écrivez une nouvelle fonction `min3Int` qui affiche le minimum de trois entiers.

Qu'affiche le programme suivant?

```
#include <stdio.h>
#include <stdbool.h>
int f1(int a, int b)
{
   return(a*b);
}
int f2(int a, _Bool b)
{
   if(b){
      return(a*a);
   }
   else{
      return(-1);
   }
}
int f3(int a, char bc)
{
   if(bc=='b'){
      return(a*a);
   }
   else{
      return(-1);
   }
}
int main()
{
   printf("%d\n", f1(3,2)+f2(2,true)-f3(4,'g'));
   printf("%d\n", f1(3,2)+f2(2,false)-f3(4,'c'));
   return(1);

}
```

Portée des variables

```
#include <stdio.h>
int a, b, c;
int f1(int x)
{
   return(a*x);
}
int g()
{
   int c;
   c=a;
   a=b;
   b=c;
   return(c);
}
int main()
{
   int x,y;
   a=2;
   b=3;
   c=4;
   x=f(a);
   y=g();
   printf("%d\n%d\n%d\n%d\n%d\n",x,y,a,b,c);
   return(1);

}
```

Passage de tableaux en paramètres

Qu'affiche le programme suivant?

```
#include <stdio.h>
#define 5
void transformation(int tab[], int taille)
{
   int i;
   for(i=0; i<taille; i++){
      if(tab[i]%2 == 0){
         tab[i]=tab[i]/2;
      }
   }
}

int main()
{
   int tabf[N]={1,2,6,3,7};
   int i;
   for(i=0; i<N; i++){
      printf("%d ",tabf[i]);
   }
   printf("\n");
   transformation(tabf, N);
   for(i=0; i<N; i++){
      printf("%d ",tabf[i]);
   }
   printf("\n");
   return(1);

}
```

Quelques fonctions sur les tableaux

- Écrivez une fonction `tabAbs` qui étant donné un tableau d'entiers, remplace tous les nombres négatifs par leur valeur absolue. Écrivez une fonction `afficheTab` qui affiche le contenu d'un tableau d'entiers. Écrivez un programme complet permettant de tester la fonction `tabAbs`.
- Écrivez une fonction `nbInf` qui compte le nombre de valeurs strictement inférieures à *x* dans un tableau de flottants.
- Écrivez une fonction `Average` qui calcule la moyenne des valeurs supérieures à *x* dans un tableau d'entiers.
- Écrivez une fonction `conjonction` qui calcule la conjonction (ou `ET` logique) des valeurs d'un tableau de booléens.

Tri par insertion

L'objectif est d'écrire un programme qui insère des éléments dans un tableau tout en les triant. Nous distinguerons la taille `taille` du tableau (c'est à dire le nombre maximum d'éléments qu'il peut contenir) du nombre `nbE1` d'éléments déjà insérés dans le tableau. Au début, le tableau est vide et à chaque nouvelle valeur, nous allons rechercher où l'insérer dans le tableau puis nous allons décaler les éléments suivants pour faire de la place. Il n'est plus possible d'ajouter de nouvel élément dès que `nbE1=taille`.

- Écrivez une fonction `indiceInsert` qui, étant donnés un tableau de flottants `tab` de taille `taille` contenant `nbE1` éléments triés par ordre croissant et un flottant *f*, retourne l'indice auquel `f` doit être inséré pour que le tableau reste trié.
- Écrivez une fonction `insertElt` qui réalise l'insertion de `f` dans `tab` si l'élément n'est pas déjà présent et si le tableau n'est pas plein. Cette fonction retournera la valeur `true` si l'insertion a été réalisée.

Démineurs

1. Écrivez une fonction à valeurs entières `compte` qui prend deux tableaux à deux dimension de même taille, `champ` et `visite`; le nombre de lignes (ou de colonnes) de ces tableaux, N; et une position (`i`, `j`) et qui renvoie
   - -1, dans le cas où `champ[i][j]==1`;
   - la somme des valeurs des cases adjacentes à `champ[i][j]`, dans le cas où `champ[i][j]==0`.

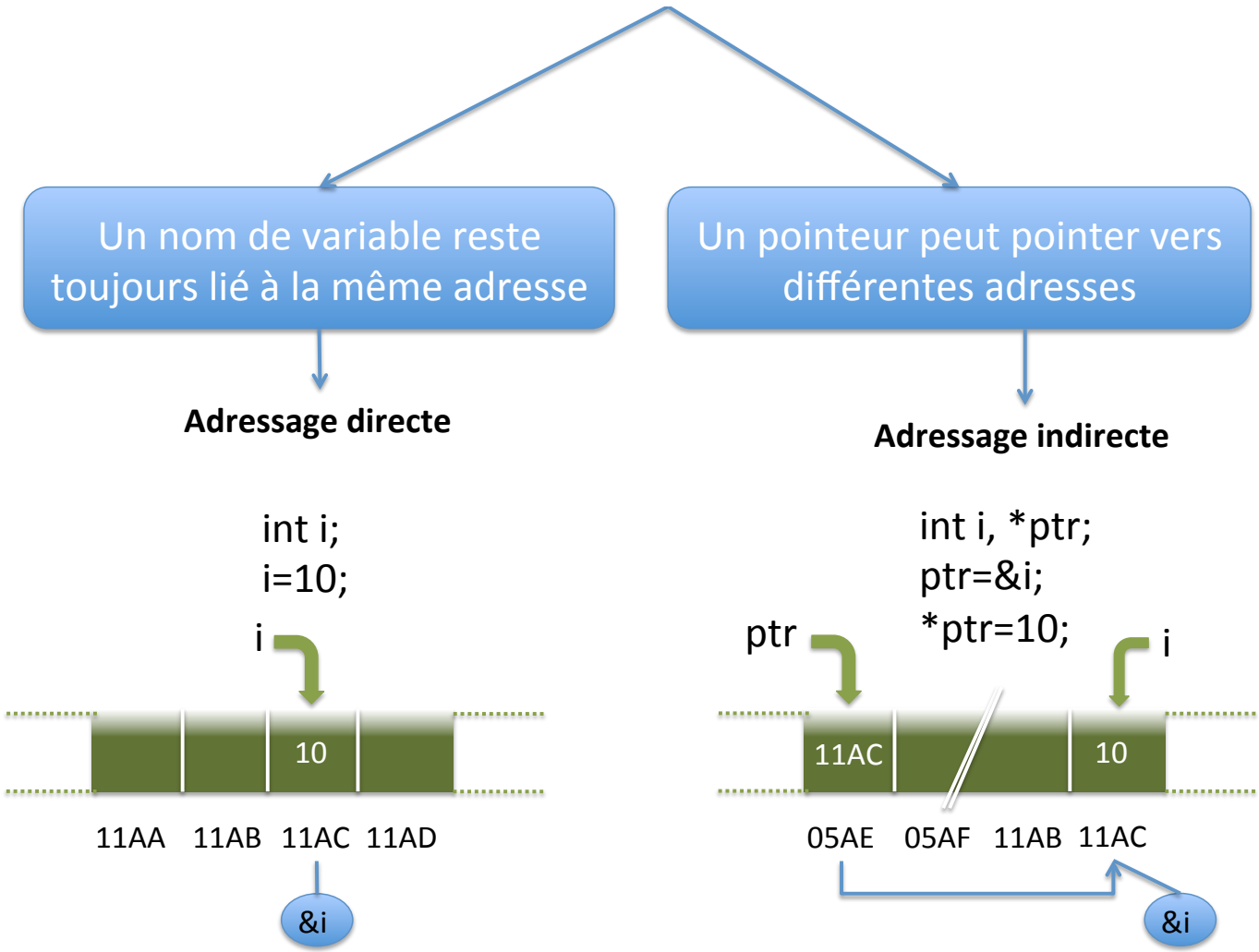
   On suppose que les valeurs de `i` et de `j` sont comprises entre 1 à $N-2$ (il n'est pas nécessaire de les vérifier), que les cases des premières lignes et colonnes ainsi que les cases des dernières lignes et colonnes de `champ` et de `visite` sont toutes égales à 0.
2. Écrivez la fonction `main` qui appellera la fonction `compte` tant que cette dernière n'a pas renvoyé la valeur -1 ou que la somme des -1 dans le tableau `visite` n'a pas atteint le nombre total des 1 (de mines) dans le tableau `champ` fixé et connu à l'avance. Les valeurs 1 du tableau `champ` sont affectées aléatoirement avant l'appel de la fonction `compte`, et toutes les cases du tableau `visite` sont initialisées à -1. Après l'appel de la fonction `compte` et dans le cas où la valeur renvoyée est différente de -1, cette dernière sera stockée dans le tableau `visite` à la position (`i`, `j`). Les positions (`i`, `j`) sont itérativement demandées à l'utilisateur et on affichera le contenu du tableau `visite` après chaque itération.

Qu'est ce qu'un pointeur?

- Un **pointer** est une variable qui contient **l'adresse** d'une autre variable.
- L'**adresse** est une valeur numérique désignant un emplacement en *mémoire*, souvent exprimée en Hexadécimal.
- En C, il existe deux **modes d'adressage** sur ordinateur
  - adressage **direct**: l'adresse est donnée directement, sans calcul intermédiaire. Le contenu de la *variable* est accessible directement par le nom de cette variable.
  - adressage **indirect**: l'accès au contenu d'une variable est obtenu en passant par un pointeur qui contient l'adresse de la variable.

Qu'est ce qu'un pointeur?

- Pointeurs et noms de variables jouent le même rôle: ils donnent accès à un emplacement dans la mémoire.



Qu'est ce qu'un pointeur?

Que fait le programme suivant?

```
#include <stdio.h>
void f1(int *param1, int param2)
{
    int var_loc=3;
    *param1=var_loc*param2;
    param2=var_loc+1;
}

int main()
{
    int v1, v2;
    v1=10;
    v2=3;
    f1(&v2,v1);
    return(1);
}
```

Passage de paramètres par référence

Qu'affiche le programme suivant?

```
#include <stdio.h>
void f1(int a, int b, int *c, int d)
{
    *c=a+b;
    d=a*b;
}

int main()
{
    int e,f,g,h;
    e=10;
    f=3;
    f1(e,f,&g,h);
    printf("g=%d h=%d\n",g,h);
    return(1);
}
```

Passage de paramètres par référence

- Écrivez une fonction `statistique` qui calcule la moyenne, ainsi que les valeurs minimum et maximum d'un tableau de réels. Les valeurs calculées seront affichées, non pas dans la fonction `statistique`, mais dans la fonction `main` appelant cette fonction.

- Écrivez une fonction `Ecart` qui calcule le plus petit et le plus grand écart entre les éléments d'un tableau de réels. Les valeurs calculées seront affichées, non pas dans la fonction `Ecart`, mais dans la fonction `main` appelant cette fonction.

Passage de paramètres par référence

On considère une fonction *f* ayant deux paramètres *i*1 et *i*2 de type entier et renvoyant un résultat de type entier. La fonction calcule la somme de deux paramètres, **l'enregistre dans une variable locale** *i*3 (de type entier), puis retourne la valeur de cette variable.

1. Écrivez la fonction *f*.

2. Les instructions suivantes d'appel de *f* provoquent-elles une erreur de typage? Pourquoi?
   - **int** a; a=f(1,2);
   - **int** a,b,c; a=f(b,c);
   - **int** a,b,c; a=f(&b,c);

Passage de paramètres par référence

Donnez les états successifs de la mémoire (pile d'exécution) lorsque l'exécution du programme suivant atteint la ligne de déclaration dans la fonction `main`.

```
#include <stdio.h>
void f(int i1, int *p_i2)
{
    int i3;
    int i4=*p_i2;
    i3=i1+i4;
    i4=i3;
    *p_i2=i4;
    return i3;
}

int main()
{
    int i1=1, i2=2, res=0;
    res=f(i1,&i2);
    return 1;
}
```

Passage de paramètres par référence

Soit le programme suivant

```
#include <stdio.h>
.....  racine(..........)
{
   ..................................
}

int main()
{
   int NbR;
   float a,b,c;
   printf("Entrez les coefficients du polynôme du 2nd degré\n");
   scanf("%f %f %f",&a,&b,&c);
   /* Appel de la fonction racine avec les coefficients a, b, et c saisis */
   NbR=racine(......);
   if(NbR==2)
      printf("Le polynôme admet deux racines réelles\n");
   if(NbR==1)
      printf("Le polynôme admet une racine double\n");
   else
      printf("Le polynôme n'admet pas de racines réelles\n");
   return 1;

}
```

Passage de paramètres par référence

- Donnez le prototype de la fonction `racine` qui doit retourner le nombre de racines du polynôme $ax^2 + bx + c = 0$ et, ajoutez dans la fonction `main` l'appel à la fonction `racine`.
- Écrivez la fonction `racine`.
- Nous souhaitons qu'en plus de retourner le nombre de racines, la fonction `racine` permette de récupérer les valeurs des racines. Donnez le prototype de la fonction `racine` modifiée et modifiez la fonction `main` si nécessaire.
- Écrivez la nouvelle fonction `racine`.

Passage de paramètres par référence

Qu'affiche le programme suivante?

```
#include <stdio.h>
void swap(int *a, int *b)
{
    int temp=*a;
    *a = *b;
    *b = *temp;
}

int main()
{
    int b=0, c=3;
    printf("\n%d %d\n",b,c);
    swap(&b,&c);
    printf("\n%d %d\n",b,c);
    return 1;
}
```

Soit le programme `debug.c` suivant :

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define DIM 9
4
5  int addition(int a, int *b){
6    return a+b;
7  }
8
9  int main(){
10   int i, b[DIM]={1,2,3,4,5,6,7,8,9,10};
11   printf("Calcul du cumul des elements du tableau:\n");
12   for(i=0; i<DIM; i++)
13     s=addition(s,b[i]);
14   for(i=0; i<(DIM-1); i++)
15     printf(" %d +",b[i])
16   printf(" %d = %d\n",b[DIM-1],s);
17   return 1;
18 }
```

Voici les erreurs obtenues lors de la compilation du programme `debug.c`

```
debug.c:  In function 'addition' :
debug.c:6:  warning:  return makes integer from pointer without
a case
debug.c:  In function 'main'
debug.c:10:  warning:  excess elements in array initializer
debug.c:10:  warning:  (near initialisation for 'b')
debug.c:13:  error:  's' undeclared (first use un this function)
debug.c:13:  error:  (Each undeclared identifier is reported
only once
debug.c:13:  error:  for each function it appears in.)
debug.c:13:  warning:  passing argument 2 of 'addition' makes
pointer from integer without a cast
debug.c:15:  error:  expected ';'
```

- Corrigez le programme pour qu'il n'y ait ni "error" ni "warning" lors de la compilation.
- Donnez l'affichage produit par le programme après les corrections.

- On suppose que l'on dispose d'une fonction

```
1 void minimumP(int tab[], int taille, int *min, int *indM)
```

qui calcule la valeur et la position de l'élément minimum d'un tableau. Ecrivez le code de la fonction `minimumP` ainsi que la fonction `main` appelant la fonction.

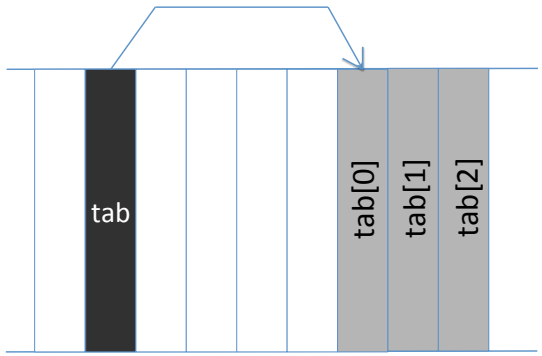
- Écrivez une fonction `nb_pos_neg` qui prend en argument un tableau et sa taille et permet d'obtenir en un seul appel le nombre de valeurs strictement positives dans le tableau **et** le nombre de valeurs strictement négatives.

- En utilisant la fonction précédente, écrivez un programme qui indique s'il y a plus, autant ou moins de valeurs positives que négatives dans le tableau suivant:
-1 3 7 0 7 8 -2 -4 1 0

Tableaux et pointeurs

Tableau = pointeur

int `tab`[3]; $\rightarrow$



Allocation de 3 cases contigües dans la mémoire

`tab` $\Longleftrightarrow$ &(`tab`[0])

En réalité la variable `tab` *n'est pas matérialisée sauf si c'est un paramètre de fonction.*

`tab`[i] = la i^{ieme} valeur après celle pointée par `tab`

Donc `tab`[0] $\Longleftrightarrow$ *tab

`tab`[i] $\Longleftrightarrow$ *(tab+i)

Le compilateur connait le type des éléments du tableau, il fait des sauts de *N* octets en *N* octets selon le type des éléments du tableau.

Tableaux et pointeurs

```
void init(int t[], int n){
    int i;
    for(i=0; i¡n; i++)
        t[i] = 3*i;
}
void main(){
    int tab[3];
    init(tab,3);
    return 1;
}
```

⇔

```
void init(int *t, int n){
    int i;
    for(i=0; i¡n; i++)
        *(t+i) = 3*i;
}
void main(){
    int tab[3];
    init(tab,3);
    return 1;
}
```

Pointeurs et passage par référence

Écrivez une fonction `compte` qui ne retourne rien et qui prend en paramètres un tableau de caractères `tab`, et un caractère `car`.

La fonction doit permettre de récupérer le nombre d'apparitions du caractère dans la chaîne de caractères.

Comme la fonction ne retrouve aucune valeur, elle doit rendre don résultat grâce à une paramètre passé par référence.

prototype de compte :
`void compte(char tab[], char car, int *nb);`

Recette pour utiliser les librairies graphiques sur une machine Linux

- Récupérer les fichiers nécessaires dans /Public/INF111/src/graphsimple
- Compiler graphsimple.c : `gcc -c graphsimple.c`
- Compiler graphlib_w2.c: `gcc -c graphlib_w2.c`
- Lier les objets obtenus: `ar -r libgraph.a graphsimple.o graphlib_w2.o`
- Créer la librairie: `ranlib libgraph.a`
- créer un répertoire pour les `include` et y mettre les fichiers graphsimple.h (par exemple `/home/toto/include`)
- Créer un répertoire `lib` et y mettre la librairie libgraph.a (par exemple `/home/toto/lib`)
- Compiler les applis : `gcc monprog.c -I/home/toto/include -L/home/toto/lib -lgraphe -lX11`