



La référence aéronautique

www.enac.fr →



Langage C : objectifs

- **Les objectifs du cours :**

- **Objectif général :** Ecrire et exécuter des applications simples en langage C.
- **Objectifs proposés :**
 - Compiler et exécuter un programme simple.
 - Utiliser la bibliothèque standard du langage.
 - Déclarer des constantes et des variables de types élémentaires, utiliser les opérations associées (déclaration, portée, affectation, opérateurs).
 - Utiliser les structures de contrôle du langage (if, switch, for, while).
 - Passer des adresses de variables (pointeurs) en paramètre de fonction.
 - Manipuler des tableaux.
 - Manipuler des chaînes de caractères, afficher et récupérer des informations sur le terminal.
 - Manipuler des types énumérés et des structures.
 - Ecrire une application à plusieurs modules, en écrivant des headers appropriés.
 - Lire et écrire des fichiers de données.
 - Effectuer des allocations dynamiques (tableaux, structures).



Langage C : Sommaire

Séance 1 : cours/tp1. Eléments de langage C

- LES OBJECTIFS DE CE COURS
- ETAPES PERMETTANT L'EDITION, LA MISE AU POINT, L'EXECUTION D'UN PROGRAMME : *Edition du programme source, Compilation du programme source, Editions de liens, Exécution du programme*
- LES DIFFERENTS TYPES DE VARIABLES : *Les types prédefinis*
- LES INITIALISATIONS
- LES DECLARATIONS DE CONSTANTES
- SORTIES DE NOMBRES OU DE TEXTE A L'ECRAN : LA FONCTION PRINTF
- AUTRES FONCTIONS DE SORTIES (puts, putchar)
- LES OPERATEURS DE BASE
- LES OPERATEURS LOGIQUES
- INCREMENTATION – DECREMENTATION
- L'INSTRUCTION SI ... ALORS ... SINON ...



Séance 2 : cours/tp2. Saisie de nombres et structures de contrôle

- NOTION DE FLUX D'ENTREE
- LA FONCTION GETCHAR
- LA FONCTION SCANF, SSCANF ET SPRINTF
- L'INSTRUCTION AU CAS OU ... FAIRE ... (switch)
- LA BOUCLE TANT QUE ... FAIRE ... (while)
- L'INSTRUCTION POUR ... (for)
- L'INSTRUCTION REPETER ... TANT QUE ... (do ... while)
- OPERATEURS COMBINES



Langage C : Sommaire



Séance 3 : cours/tp3. Les Fonctions, les tableaux, découpage modulaire

- **LES TABLEAUX DE NOMBRES (INT ou FLOAT)**
- **INITIALISATION DES TABLEAUX**
- **FONCTIONS ET PASSAGE DE PARAMETRES PAR VALEUR**
- **LE PASSAGE DE PARAMETRES ENTRE FONCTIONS OU ENTRE FONCTIONS ET PROGRAMME PRINCIPAL.**
- **FONCTIONS RECURSIVES**
- **DECOUPAGE MODULAIRE D'UN PROGRAMME**
- **LES FICHIERS D'EN-TETE UTILISATEUR**
- **LES DIRECTIVES DU PREPROCESSEUR**

Séance 4 : tp1 (if, boucle, printf, scanf)

Séance 5 : cours/tp4. Les Pointeurs

- **L'OPERATEUR ADRESSE &**
- **LES POINTEURS**
- **DECLARATION DES POINTEURS**
- **AFFECTATION D'UNE VALEUR A UN POINTEUR**
- **PASSAGE DE L'ADRESSE DE VARIABLE EN PARAMETRE**
- **PASSAGE DE TABLEAUX EN PARAMETRES**
- **ALLOCATION DYNAMIQUE**
- **PORTEE DES VARIABLES LOCALES**



Langage C : Sommaire



Séance 6 : cours/tp5. Les structures et chaînes de caractères

- **LES STRUCTURES**
- **STRUCTURES ET TABLEAUX**
- **POINTEUR SUR STRUCTURE**
- **LES DECLARATIONS DE TYPE SYNONYMES: TYPEDEF**
- **LES CHAINES DE CARACTERES (utilisation de string.h, chaînes dynamiques)**
- **LES PARAMETRES DU MAIN**

Séance 7 : tp2 : tableau de structures (Annuaire).

Séance 8 : cours/tp6. Découpage modulaire d'un programme, Les fichiers.

- **RESUME SUR VARIABLES ET FONCTIONS : VARIABLE GLOBALE, STATIC EXTERN, VARIABLE LOCALE STATIC**
- **MANIPULATION DE FICHIERS TEXTES et BINAIRES**

Séance 9 : tp3. Lecture/Ecriture dans des fichiers textes

Séance 10 : tp noté de synthèse.





Séance 1 : Eléments de langage C



Construction d'un programme :

2 méthodes pour traduire un programme :

- L'interprétation. Les instructions du programme sont traduites une à une et exécutées (pas de fichier exécutable généré) : langage Php, Python.
- La compilation. Tout le programme source est traduit en instructions machines (fichier exécutable) : langage C, C++.

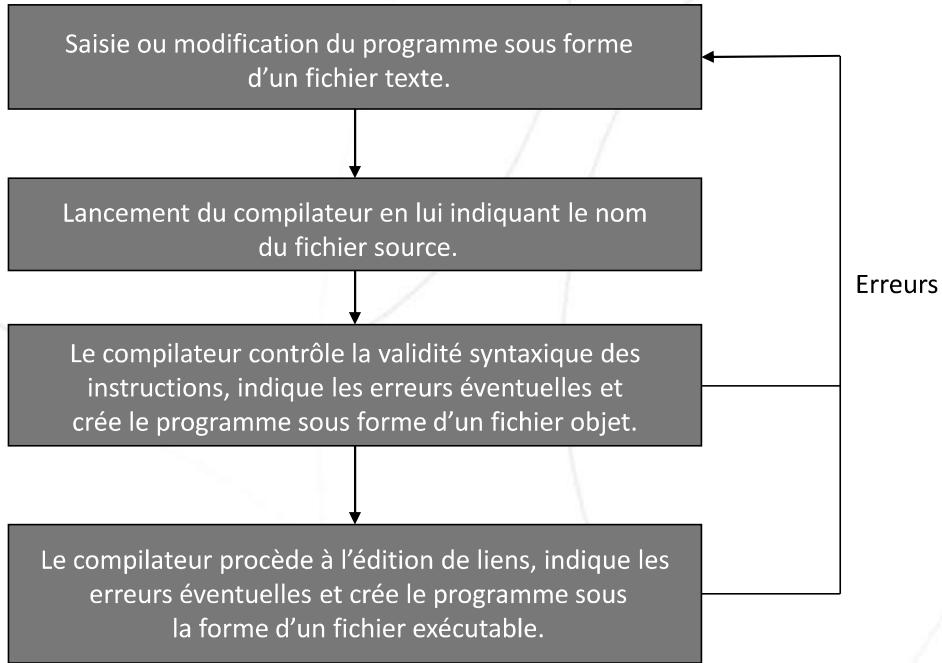
Il existe des méthodes « dites intermédiaires » : compilation (génération d'un « pseudo-code » ou bytecode, fichier « *.class ») puis interprétation → langage Java.





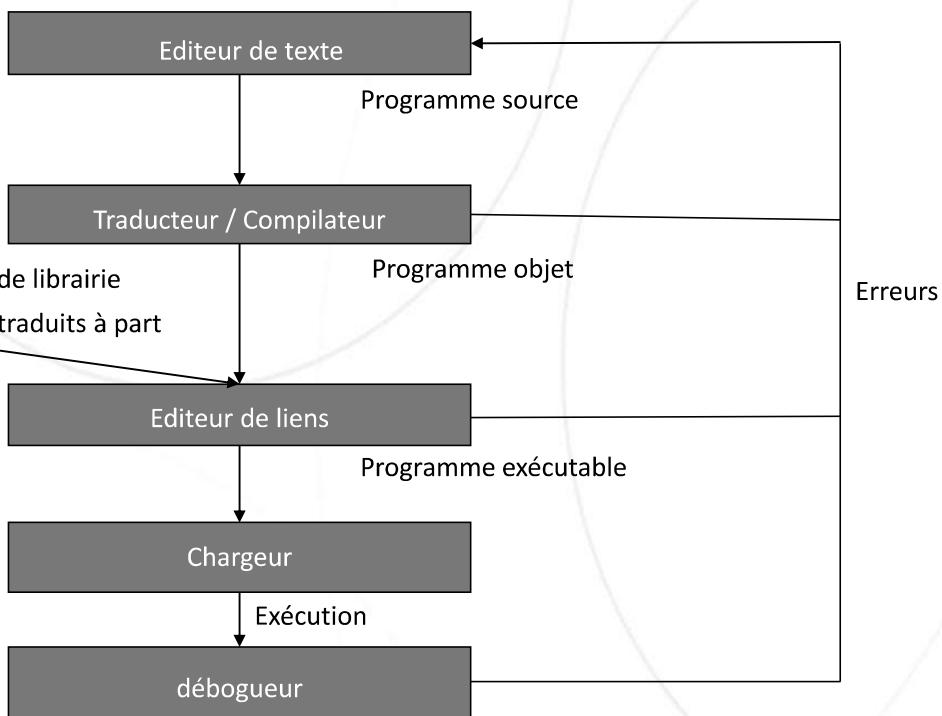
Langage C : séance 1

La programmation : du source à l'exécutable



Langage C : séance 1

Environnement de programmation minimum :





Langage C : séance 1



Définitions :

Compilateur :

programme qui traduit le code source écrit dans un langage de haut niveau (facilement compréhensible par l'humain) vers un langage d'assemblage ou langage machine, généralement représenté sous forme binaire (code objet ou bytecode).

Editeur de liens :

programme qui permet de créer des fichiers exécutables ou des bibliothèques dynamiques ou statiques, à partir de fichiers objets.

Lien statique : le fichier objet et la bibliothèque sont liés dans le même fichier exécutable.

Lien dynamique : le fichier objet est lié avec la bibliothèque, mais pas dans le même fichier exécutable (liens établis lors du lancement de l'exécutable).



Langage C : séance 1



Chaque programme chargé en mémoire dispose de plusieurs segments :

- le segment de code, contenant les instructions du programme en langage machine.
- le segment de données, contenant les données déclarées comme **globales** dans le programme.
- le segment de pile, permettant d'exécuter des programmes contenant des sous-programmes, et stockant le contexte d'exécution du sous-programme, ses **paramètres** ainsi que ses **variables locales**.
- le tas, ou mémoire dynamique, pour stocker des données. La taille de cette zone varie au cours de l'exécution du programme.





Langage C : séance 1



Les différents blocs mémoire utilisés par un programme :

<i>Program</i>	<ul style="list-style-type: none"> • Code machine • Données statiques globales • Liens avec bibliothèques
<i>Heap</i>	<ul style="list-style-type: none"> • Tas (→ malloc)
<i>Free memory</i>	<ul style="list-style-type: none"> • Mémoire libre (utilisée pour la croissance du tas et de la pile)
<i>links</i>	<ul style="list-style-type: none"> • Liens dynamiques : <p>Pour charger dynamiquement des bibliothèques.</p>
<i>Stack</i>	<ul style="list-style-type: none"> • Pile : <p>Variables locales de chaque fonction, adresse de retour, ...</p>



La référence aéronautique



Langage C : séance 1



Caractéristiques générales du langage C :

- Un programme écrit en langage C se stocke dans un fichier texte ayant l'extension « .c ».
- Ce fichier doit posséder la fonction principale d'entrée : la fonction « main »
- Pour utiliser des fonctions d'entrées/sorties du langage C, on doit inclure un fichier standard avec une directive au préprocesseur.
- Exemple de programme en langage C :

```
#include <stdio.h>
int main()
{
    printf("bonjour");
    return 0;
}
```

bloc d'instructions {

Instruction « printf » du fichier « stdio.h » ←

Instruction « return » du langage c ←



La référence aéronautique



Langage C : séance 1



Caractéristiques générales du langage C :

- Distinction entre minuscules et majuscules, les mots réservés du langage en minuscules.
- Notion de déclaration de variables : toute variable est typée et déclarée avant son usage, en début de bloc.
- Notion d'affectation symbolisée par le signe =.
- Les commentaires :


```
// jusqu'à la fin de ligne
```

 ou


```
/* sur plusieurs
         lignes
      */
```

Rm : contrairement au langage Python, l'indentation en langage C n'est pas significative pour le compilateur.



La référence aéronautique

www.enac.fr

13



Langage C : séance 1



Les types prédéfinis :

- **void** : aucune valeur.
utilisé en retour de fonction, pour indiquer aucune valeur renournée.
ex : **void f1();** // donc pas d'instruction **return** obligatoire dans f1
- **char** : un caractère (1 octet).
- **short , int, long** : pour des entiers.
- **float, double** : pour des réels.
- **enum** : pour définir un type énumératif.
exemple :
typedef enum {lundi,mardi,mercredi, jeudi , vendredi ,samedi,dimanche} Semaine;
Semaine unJour = lundi;
- **pas de type booléen** : 0(zéro) c'est faux,
toute valeur différente de 0 , c'est vrai.
- **typedef** : Permet de définir un nouveau type à partir d'un type existant.
Syntaxe : **typedef typeExistant nouveauType;**
Exemple : **typedef int Euro;**



La référence aéronautique

www.enac.fr

14



Langage C : séance 1



Les valeurs constantes :

- Constante entière :

En base dix : 12
 En base octale : 012 // précédé de zéro
 En base héxa : 0X12 // précédé de zéro et x (ou X)
- Constante réelle :
 plusieurs notations possibles : 3. , .4 , 2e4 , 2.3e4
- Constante caractère :
 entouré de quote : 'a'
 pour les caractères non imprimables : '\001' (valeur ASCII du caractère)
 et pour les caractères d'échappement : '\n', '\t', ...
- Constante chaînes de caractères :
 entouré de guillemets : "bonjour"
 Le compilateur rajoute un caractère '\0' en fin de chaîne (marqueur).



Langage C : séance 1



Déclaration et initialisation des variables :

- Variable entière :
`int i; // variable déclarée mais non initialisée`
`long j=0; // variable déclarée et initialisée`
- Variable réelle :
`float varf = .4 ;`
`double vard = 0 ;`
- Variable caractère :
`char car = 'a' ;`
- Variable chaînes de caractères :
`char chaine[20] ; // taille précisée obligatoirement si non initialisée`
`char unechaine[] = "bonjour"; // taille non obligatoire ici`





Langage C : séance 1



Déclaration de variables constantes :

Il existe deux méthodes :

- Première méthode :

Déclaration d'une variable, dont la valeur sera constante pour tout le programme:

```
const float PI=3.14159 ;
```

On utilise le mot réservé « const » et on écrit par convention les constantes en majuscules.

Dans ce cas, le compilateur réserve de la place en mémoire (ici 4 octets), pour la variable PI, mais dont on ne peut changer la valeur.

- Deuxième méthode :

Définition d'un symbole à l'aide de la directive de compilation **#define**

```
#define PI 3.14159  
float perimetre, rayon= 6;  
perimetre = 2* rayon * PI;
```



Langage C : séance 1



Les Entrées/Sorties :

- Affichage écran :

- On utilise la fonction « printf » de la librairie standard du langage C.

➔ Demander à utiliser la librairie standard des E/S :

```
#include <stdio.h> // à mettre en début de fichier
```

- Paramètres de la fonction « printf » :

```
printf( "bonjour" ); ➔ bonjour
```

Ou

```
int i=8;  
printf("valeur de i=%d", i); ➔ valeur de i=8
```

- Un format à spécifier pour chaque type de variable :

%d	: entier
%f ou %lf	: float ou double
%c	: char
%s	: chaînes de caractères
%p	: adresse (pointeur)





Langage C : séance 1

Les Entrées/Sorties : le cas particulier des chaînes de caractères

- Affichage écran : la fonction « printf »

- On utilise la fonction « printf » de la librairie standard du langage C (stdio.h).

```
printf("bonjour");
```

- Si on souhaite stocker la chaîne dans une variable, on utilise un tableau de caractères :

```
char chaine[]="bonjour"; // chaine ne pourra contenir que 8 caractères max ('\0' inclus).
```

Ou

```
char chaine[10] = "bonjour"; // taille de chaine : 10 max
```

Ou

```
char chaine[10]; // taille à préciser obligatoirement dans ce cas
```

```
strcpy(chaine, "bonjour"); // fonction (de "string.h") qui recopie "bonjour" dans chaine
```

Puis pour l'affichage :

```
printf("%s", chaine);
```

Pour les sauts de ligne : on utilise le caractère spécial '\n'.

```
printf("%s\n", chaine);
```



Langage C : séance 1

Les Entrées/Sorties : le cas particulier des chaînes de caractères

- Affichage écran : la fonction « puts »

- On utilise la fonction « puts » de la librairie standard du langage C (stdio.h).

```
puts("bonjour"); // saut de ligne automatique : équivalent à printf("bonjour\n");
```

- Pour afficher le contenu d'une chaîne de caractères :

```
char chaine[] = "bonjour";
puts(chaine);
```

Le paramètre de « puts » doit être **obligatoirement** une chaîne de caractères !

- Affichage d'un caractère : la fonction « putchar »

Si la variable c est un **char** :

```
putchar(c); // équivalent à printf("%c\n", c);
```





Langage C : séance 1



Quelques opérateurs de base :

- **Une expression** → suite d'opérateurs et d'opérandes retournant toujours une valeur.

exemple : $x=y$
 $x+f(x)$

- Une expression terminé par un point virgule (;) → **une instruction**.

- **L'opérateur d'affectation** : =

exemple : $x=y; // x prend la valeur de y$

affectation multiple :

$i=j=k=l;$

équivalent à : $k=l; j=k; i=j;$



La référence aéronautique

www.enac.fr

21



Langage C : séance 1



Quelques opérateurs de base :

- **Les opérateurs arithmétiques** :

- unaires : + , -

- binaires :

+ (addition).

- (soustraction).

/ (division) si les deux opérandes entières → résultat entier (1/2 → 0)
 sinon résultat réel (1.0/2 → 0.5)

* (multiplication).

% (modulo) reste de la division entière (3%2 → 1).

- **Les opérateurs booléens** :

! Inversion booléenne

&& ET logique

|| OU logique

- **Les opérateurs relationnels** :

== égalité

!= différent de

< ou <=

> ou >=



La référence aéronautique

www.enac.fr

22



Langage C : séance 1



L'instruction « si ... alors ... sinon... »:

Syntaxe :

```
if ( expression_condition )
{
    instruction1;
}
else
{
    instruction2;
}
```

Parenthèses obligatoires

Le bloc est facultatif pour une seule instruction,
Mais conseillé pour une meilleure lisibilité !

La clause « else » est facultative



Langage C : séance 2



Séance 2 : Saisie et instructions de contrôle





Langage C : séance 2



Les Entrées/Sorties :

- Saisie clavier :

- On utilise la fonction « `scanf` » de la librairie standard du langage C.

➔ Demander à utiliser la librairie standard des E/S :

```
#include <stdio.h> // à mettre en début de fichier
```

- Paramètres de la fonction « `scanf` » :

```
int i=8;           & ➔ opérateur d'adresse
scanf("%d", &i);
```

- Premier paramètre : un format à spécifier pour chaque type de variable (idem `printf`).

- !!!! Le deuxième paramètre (ici `&i`) correspond à l'adresse mémoire où se trouve l'entier.

- !!!! Le deuxième paramètre doit toujours correspondre à l'adresse où l'on doit écrire la donnée.

- valeur renournée par `scanf` : le nombre d'affectations correctes.



La référence aéronautique

www.enac.fr

2



Langage C : séance 2



Saisie de chaînes de caractères avec « `scanf` » :

Si `ch` est une chaîne de caractères : `char ch[10];`

- Saisie limitée de caractères :

```
scanf(" %5s" , ch);
```

Avec 5, nombre max de caractères à saisir.

- Saisie comprenant un (ou plusieurs) espace(s) :

```
scanf("%[^\\n]" , ch);
```

- Saisie avec espace et 5 caractères max par exemple :

```
scanf("%5[^\\n]" , ch); // avec 5 nombre max de car.
```



La référence aéronautique

www.enac.fr

3



Langage C : séance 2



La fonction « sscanf » :

- Permet d'extraire des données à partir d'une chaînes de caractères.
- Syntaxe : `sscanf(dans , comment , où);`
Recherche l'information dans le premier argument, sous la forme du comment, et le range dans le où.

La fonction « sscanf » retourne le nombre de valeurs extraites correctement.

- exemple :

```
int entier;  
float reel;  
char tampon[51];  
...  
printf("entrez le message \n");      // on tape : «12  3.14»  
scanf("%50[^\\n]", tampon);  
sscanf(tampon,"%d %f",&entier,&reel);    // entier vaut 12, reel vaut 3.14
```



Langage C : séance 2



Saisie d'un caractère : la fonction « getchar » :

Elle appartient à la bibliothèque « stdio.h ».

Usage :

```
char car;  
car=getchar(); // équivalent à scanf("%c", &car);
```





Langage C : séance 2



La fonction « sprintf » :

- Permet de convertir en chaîne de caractères des données numériques.
- Syntaxe : `sprintf(chaîne résultat, format, variables);`
Recherche l'information dans le(s) dernier(s) argument(s), sous la forme du format, et le stocke dans la chaîne résultat.
- exemple :


```
int entier=10;
float reel=3.14;
char tampon[50];
...
sprintf(tampon,"%d %f",entier,reel);
printf("%s", tampon); // affiche la chaîne "10 3.14"
```
- Privilégier l'usage de `snprintf` :


```
snprintf(tampon,50, "%d %f",entier,reel); // évite le débordement du tampon (50 max ici)
```



La référence aéronautique



Langage C : séance 2



L'instruction « switch » de base :

Syntaxe :

```
switch (expression_selection)
{
    case valeur1 : i1;i2;...; in;
    case valeur2 : i21;i22;...; i2n;
    ...
    default : idef1;idef2; ... ; ← facultatif
}
```

Exemple :

```
int i=1;
switch ( i )
{
    case 1: printf("valeur 1 ");
    case 2 : printf("valeur 2 ");
    default : printf("valeur defaut ");
}
```

Affichage → valeur 1 valeur 2 valeur defaut



La référence aéronautique



Langage C : séance 2



L'instruction « switch » : usage très fréquent.

Syntaxe :

```
switch (expression_selection)
{
    case valeur1 : i1;i2;...; in; break;
    case valeur2 : i21;i22;...; i2n; break
    ...
    default : idef1;idef2; ... ;
}
```

L'instruction break permet de sortir du bloc switch

Exemple :

```
int i=2;
switch ( i )
{
    case 1: printf("valeur 1 "); break;
    case 2 : printf("valeur 2 ");break;
    default : printf("valeur defaut ");
}
```

Affichage → valeur 2



Langage C : séance 2



Les instructions itératives :

• L'instruction « while » :

syntaxe :

ou

while (expression_condition)
instruction;

Parenthèses obligatoires

```
while (expression_condition)
{
    instruction1;
    ...
    instructionn;
}
```

• L'instruction « do ... while » :

syntaxe :

```
do {
    instruction ou bloc
} while (expression_condition);
```





Langage C : séance 2



Les instructions itératives :

• L'instruction « for » :

syntaxe : **for (expression1 ; expression2 ; expression3)**
instruction ou bloc instructions

signification :

- expression1 : initialisation de la variable compteur.
- expression2 : test donnant la condition de poursuite des itérations.
- expression3 : modification de la variable compteur.

exemple :

```
int tab[10];
int i;
for (i=0; i<10; i++)
    tab[i]=0;
```

```
char ch[10];
int i;
for (i=0;i<10;i++)
    ch[i]='\0';
```

L'instruction « break » dans une boucle permet d'en sortir.

L'instruction « continue » dans une boucle permet de passer à l'itération suivante.



Langage C : séance 2



Les opérateurs de base :

• Les opérateurs d'affectation composés :

Pour les opérateurs : +, -, *, /, %, &, |, ^, <<, >>
on peut remplacer :

x = x **opérateur** y; par x **operateur** = y;

Exemple : x+= y; équivalent à x=x+y;

• Les opérateurs d'incrémentation :

surincrémentation : (opérande++) (idem avec opérande--)

Exemple : y= a*(b++);
équivalent à : y=a*b;
 b=b+1;

préincrémentation : (++opérande) (idem avec --opérande)

Exemple : y=a*(++b);
équivalent à : b=b+1;
 y=a*b;





Langage C : séance 2



Les opérateurs de base :

• L'opérateur conditionnel :

Syntaxe : **expression1 ? expression2 : expression3**

Signification :

si (**expression1 !=0**) **alors** **expression2** est évaluée
sinon **expression3** est évaluée.

Exemple : **(a!=0) ? (b=a):(b=c)**



• L'opérateur virgule :

Syntaxe : **expression1, expression2**

Signification : L'expression 1 est évaluée et sa valeur oubliée, l'expression 2 est évaluée et sa valeur correspond à la valeur de l'ensemble.

Exemple : **x= (i=1, k=3); // x vaudra 3, i vaudra 1 et k vaudra 3 après exécution**

L'opérateur de taille :

Syntaxe : **sizeof nom_type** ou **sizeof nom_variable**

Signification : retourne la taille en octets de nom_type.

Exemple : **char ch[10];
int i;**

➔ **sizeof ch** retourne 10
➔ **sizeof i** retourne 4 (nbre d'octets occupé par un int en mémoire, dépend du SE).



La référence aéronautique

www.enac.fr

12



Langage C : séance 3



Séance 3 : Les Fonctions, les tableaux



La référence aéronautique

www.enac.fr

1



Langage C : séance 3



Les tableaux statiques :

- Déclaration d'un tableau statique :

type nom_tableau[taille];

Si la variable est locale → stocké dans la pile.

Si la variable est globale → stocké dans le segment des données globales.

Toutes les données du tableau seront consécutives en mémoire.

L'expression « nom_tableau » correspond à l'adresse du premier élément.

L'expression « nom_tableau » est une adresse constante.

Exemple de déclaration d'un tableau de 10 entiers :

```
int montableau[10];
```

- Déclaration et initialisation d'un tableau statique :

type nom_tableau[taille]={ valeur1, valeur2, ... };

Le nombre de valeur doit être inférieure ou égal à taille.

on peut écrire : **type nom_tableau[]={ valeur1, valeur2, ... };**



Langage C : séance 3



Les tableaux statiques :

- Déclaration d'un tableau statique :

int nom_tableau[3]={0,0,0};

Occupation mémoire de ce tableau :

3 (cases) x 4 (octets/int) = 12 octets consécutifs.

`sizeof(nom_tableau)` → 12

adr1	0
adr2	0
adr3	0

Adresse mémoire de la première case :

« nom_tableau » → adr1

`printf("%p", nom_tableau);` → affiche la valeur de l'adresse en hexa (adr1)

`printf("%p", nom_tableau+1);` → affiche la valeur de l'adresse en hexa (adr2)





Langage C : séance 3



Les tableaux statiques de tableaux :

- Déclaration d'un tableau statique de tableaux :

type nom_tableau[taille1][taille2];

Toutes les données seront consécutives en mémoire.

`int t[2][3]={1,2,3,4,5,6};` ou `int t[2][3]={{1,2,3},{4,5,6}};`

1	2	3	4	5	6
---	---	---	---	---	---

L'expression « nom_tableau » correspond à l'adresse du premier élément(`&t[0][0]`).
L'expression « nom_tableau » est une adresse constante.

Accès : `t[0][1] → 2`

Aucune vérification de débordement :

On peut écrire : `t[40][50]=8;` → compilation : ok, exécution : pb!!!



La référence aéronautique

www.enac.fr

4



Langage C : séance 3



Le passage de tableaux statiques en paramètre de fonction :

• Passage de tableau :

Tout paramètre de fonction déclaré comme tableau d'éléments de type T est automatiquement converti en un pointeur vers T à la compilation.

On peut donc écrire :

Pour un tableau à une dimension,

`void f(T t[]);` ou `void f(T * t);`

Pour un tableau à deux dimensions , il faut toujours préciser la dernière dimension :

`void g(T t[][10]);` ou `void g(T (*t)[10]);`



La référence aéronautique

www.enac.fr

5



Langage C : séance 3



Type de retour d'une fonction :

- Pas de type de retour : void

Exemple :

```
void messageFin() {
    printf("bonne fin de journée");
}
```

- Avec un type non void :

Exemple : ↓

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

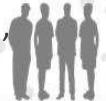
Type de retour non void donc au moins un return dans la fonction

On sort de la fonction dès qu'un « return » est exécuté.

Appel de la fonction :

```
k=maxEntier( i,j );
ou k=maxEntier(10,20);
```

} Il faut récupérer la valeur de retour à l'appel, Sinon elle est perdue.



Langage C : séance 3



Passage par valeur de paramètres :

Exemple :

```
void afficherEntier(int j) {
    printf(" valeur = %d" , j);
}
```

Paramètre formel de la fonction

```
int main() {
    int i=5;
    afficherEntier(i);
    return 0;
}
```

Variable locale (au main)

La variable i est un paramètre réel ou effectif lors de l'appel à la fonction





Langage C : séance 3



Passage par valeur de paramètres :

Exemple :

```
void afficherEntier(int j) {
    printf(" valeur = %d", j);
}
```

```
int main() {
    int i=5; ← Variable locale (au main)
    afficherEntier(i);
    return 0;
}
```

i : 5

Etape 1 de l'exécution :

création de la variable i dans la pile (zone des variables locales) et initialisation



Langage C : séance 3



Passage par valeur de paramètres :

Exemple :

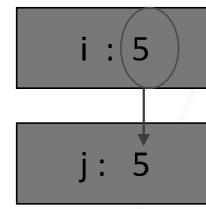
```
void afficherEntier(int j) {
    printf(" valeur = %d", j);
}
```

Paramètre formel de la fonction

```
int main() {
    int i=5;
    afficherEntier(i);
    return 0;
}
```

La variable i est un paramètre réel ou effectif lors de l'appel à la fonction

Recopie de la valeur du paramètre réel (i) pour initialiser le paramètre formel j



Etape 2 de l'exécution :

création de la variable j dans la pile, et initialisation avec la valeur du paramètre réel.





Langage C : séance 3



Passage par valeur de paramètres :

Exemple :

```
void afficherEntier(int j) {
    printf(" valeur = %d", j);
}

int main() {
    int i=5;
    afficherEntier(i);
    return 0;
}
```

Paramètre formel de la fonction

i : 5
j : 5

Etape 3 de l'exécution :

Affichage de la valeur du paramètre formel j.



Langage C : séance 3



Passage par valeur de paramètres :

Exemple :

```
void afficherEntier(int j) {
    printf(" valeur = %d", j);
}

int main() {
    int i=5;
    afficherEntier(i);
    return 0;
}
```

i : 5
~~j : 5~~

Etape 4 de l'exécution :

Retour dans le main : destruction des variables de la fonction afficheEntier.





Langage C : séance 3



Passage par valeur de paramètres :

Résumé du passage par valeur :

- **Le passage par valeur ne modifie pas le paramètre réel.**

```
void f(T t1);
T t=val1;
f(t);
// t a toujours la même valeur (val1)
```

- **Lors d'un passage par valeur, la valeur du paramètre réel peut être une constante.**

```
void f(int t1);
f( 10); // autorisé
```



Langage C : séance 3



Principe de la récursivité :

La récursivité permet de décrire un processus en faisant appel à ce même processus.

Exemple : calcul de la factorielle.

```
fact(0)=1
fact(n) =n x fact(n-1)
```

En langage C :

- toute fonction en C peut-être récursive.
- une fonction est récursive si elle s'appelle elle-même.
- la récursivité est plus coûteuse en mémoire pour le système qu'une simple boucle.





Langage C : séance 3



Fonction récursive :

Certaines fonctions qui s'écrivent avec une boucle peuvent également s'écrire en utilisant le principe de récursivité.

Exemple :

Calcul de la somme des n premiers entiers positifs :

```
/* version itérative : */
long somme(long n)
{
    long i,s=0;
    for (i=1 ; i<=n ; i++)
        /* accumulation dans une variable */
        s+=i;
    return s;
}
```

```
/* version récursive : */
long somme(long n)
{
    long s=0;
    if (n==1) /* condition d'arrêt */
        s=1;
    else
        s=n+somme(n-1);
    return s;
}
```



Langage C : séance 3



Fonction récursive :

Calcul de l'occupation mémoire maximale (en pile) de l'exécution des fonctions précédentes :

➔ Appel : somme(3)

```
/* version itérative : */
```



```
/* version récursive : */
```

Appel somme(0)	{	s : 1
Appel somme(1)	{	n : 0
Appel somme(2)	{	s : 0
Appel somme(3)	{	n : 1
		s : 0
		n : 2
		s : 0
		n : 3





Langage C : séance 3



Fonction récursive : quel intérêt ?

Pour écrire certains algorithmes, il faudrait écrire un nombre important de boucles imbriquées (donc algorithme très compliqué et lourd à écrire, à corriger et à faire évoluer).

→ En utilisant la récursivité, certains de ces algorithmes peuvent s'écrire en quelques lignes !

→ Quelques critères pour une utilisation possible et justifiée de la récursivité :

- La complexité du problème se réduit bien en utilisant la récursivité.
- La méthode itérative ne peut pas se mettre en place, est trop lourde ou peu adaptée.
- Plusieurs solutions à votre problème sont envisageables.
- La complexité du problème est imprévisible, et pas forcément linéaire (explorer des tableaux contenant des tableaux, par exemple).

Les algorithmes de construction de structures de données complexes de types arbres sont plus faciles à implémenter en utilisant la récursivité.



Langage C : séance 3



Découpage modulaire d'un programme en langage C :

Un programme écrit en langage C est rarement écrit dans un seul module (ou fichier). Le code source s'étend souvent sur plusieurs modules, dont chacun a un rôle bien précis.

Nous allons à travers un exemple comprendre comment décomposer son programme en modules différents et voir comment est construit le programme final (l'exécutable).

Etape 1 : création des modules d'en-tête.

Il s'agit de fichiers ayant une extension « .h » (h pour header). Ces fichiers contiennent des définitions de types et les déclarations des fonctions à développer.

Etape 2 : création des modules de définitions des fonctions.

Ces fichiers reprennent les fonctions déclarées précédemment et les définissent (ajout du corps de la fonction).

Etape 3 : création du programme principal (main).

Le programme « main » appelle les fonctions.





Langage C : séance 3



Découpage modulaire d'un programme en langage C :

Fichier « mesfonctions.h »

```
#include <stdio.h>
#ifndef _MESFONCTIONS_H
#define _MESFONCTIONS_H
void mafonction(char * ch);
#endif
```

Fichier « mesfonctions.c»

```
#include "mesfonctions.h"
void mafonction(char * ch)
{
    printf("message : %s \n",ch);
}
```

Fichier « monprogramme.c »

```
#include <stdio.h>
#include "mesfonctions.h"
int main()
{
    // appel de la fonction
    mafonction("bonjour");
    return 0;
}
```

Dans chaque module « .h », on doit commencer par les deux directives au préprocesseur `#ifndef` et `#define` pour éviter les inclusions multiples, et terminer par `#endif`.

Dans les modules de définitions de fonctions (ici le fichier « mesfonctions.c »), on inclut (directive `#include`) le module d'en-tête correspondant (ici « mesfonctions.h »).

Dans le programme principal (ici « monprogramme.c »), on inclut le module d'en-tête.



La référence aéronautique

www.enac.fr

18



Langage C : séance 3



Le préprocesseur :

Le préprocesseur permet de réaliser une phase de prétraitement du fichier source avant la compilation : inclusion de fichiers, substitution de termes ,...

•Pour demander à inclure un autre fichier source :

On utilise la directive « `#include` » :

`#include <stdio.h>`

➔ demande l'inclusion du fichier d'en-tête « stdio.h » de la librairie standard du C, contenant toutes les déclarations des fonctions d'Entrées/Sorties.

`#include "monfichier.h"`

➔ demande l'inclusion du fichier d'en-tête « monfichier.h » de mon répertoire courant contenant des déclarations de mes fonctions.

on peut écrire aussi : `#include "c:/mesdocuments/monfichier.h"`



La référence aéronautique

www.enac.fr

19



Langage C : séance 3



Le préprocesseur :

- Pour demander à définir une expression :

On utilise la directive « #define » :

```
#define MAX 100 // pas de ; à la fin !
```

...

```
if (i<MAX) ...
```

Le préprocesseur remplacera l'expression MAX par sa valeur (100) juste avant la phase de compilation.

Intérêt : MAX n'est pas une variable, elle n'occupe pas de place en mémoire.

Inconvénient : MAX n'est pas typée, donc pas de possibilité de vérifier des typages.

• Les macros :

On utilise toujours la directive « #define » :

```
#define MAX((a), (b)) ((a)>(b)?(a) : (b))
```

...

```
if (k != MAX(i,j) ) ...
```



La référence aéronautique

www.enac.fr

20



Langage C : séance 3



Le préprocesseur :

• La directive d'inclusion conditionnelle :

A chaque directive « #include ... », il y a inclusion du contenu du fichier.

Le préprocesseur ne gérant pas les inclusions multiples, si le contenu a déjà été inclus, il y aura une inclusion multiple des mêmes fonctions → échec de l'édition de liens.

Le programmeur doit donc empêcher les inclusions multiples. Il faut donc systématiquement concevoir les fichiers d'en-tête de la façon suivante :

Fichier « monfichier.h » :

```
#ifndef __MONFICHIER_H__
#define __MONFICHIER_H__ // on reprend le nom du fichier en majuscule et _
...
//mettre ici toutes les définitions de types et les déclarations des fonctions
...
#endif // fin du fichier « monfichier.h »
```

Il n'a plus qu'à faire un #include "monfichier.h" dans les fichiers où l'on souhaite utiliser les types et les fonctions de ce fichier.



La référence aéronautique

www.enac.fr

21



Utilité des pointeurs :

- **Passage de paramètres par adresse dans une fonction :**

Les pointeurs sont utilisés pour le passage de paramètres dans une fonction dont on souhaite modifier le contenu.

- **Utilisation de la mémoire dynamique (ou Tas) :**

Une partie de la mémoire appelée « mémoire dynamique » est disponible par un programme en langage C pour y stocker des données. Mais son accès se fait uniquement par des pointeurs.

- **Construction de données récursives :**

La construction de données récursives (liste chaînée, arbre , ...) ne peut se faire que dans la mémoire dynamique, donc avec l'utilisation de pointeurs pour y accéder.



Langage C : séance 5



Les opérateurs de base :

• L'opérateur d'adresse & :

Permet de récupérer l'adresse d'une variable.

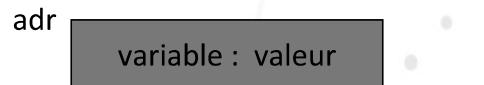
Exemple : int i=2;

```
→ int * pti = &i;
    *pti=3;
```

• Notion de pointeur :

Déclaration et initialisation d'un pointeur :

```
type * nom_pointeur=NULL;
```



Affectation d'une valeur à un pointeur :

```
nom_pointeur= & variable;
```

Accès au contenu d'une adresse :

```
*nom_pointeur= valeur;
```



Affichage d'une adresse : printf("%p", nom_pointeur);

www.enac.fr

3



Langage C : séance 5



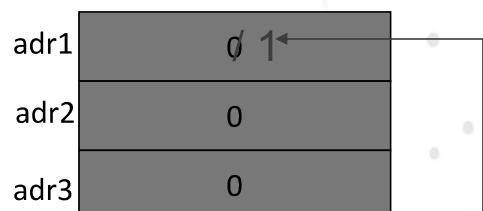
Les tableaux statiques :

• Déclaration d'un tableau statique :

```
int nom_tableau[ 3 ]={0,0,0};
```

Occupation mémoire de ce tableau :

3 (cases) x 4 (octets/int) = 12 octets consécutifs.
sizeof(nom_tableau) → 12



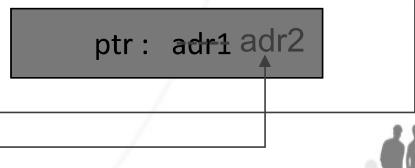
Adresse mémoire de la première case :

« nom_tableau » → adr1
printf("%p", nom_tableau); → affiche la valeur de l'adresse en hexa.

Accès par un autre pointeur :

il est possible de faire :

```
int * ptr=nom_tableau;
*ptr=1;
ptr++;
```



www.enac.fr

4



Langage C : séance 5

Les tableaux statiques :

- Déclaration d'un tableau statique :

```
int nom_tableau[ 3 ]={0,0,0};
```

Accès par un autre pointeur :

il est possible de faire :

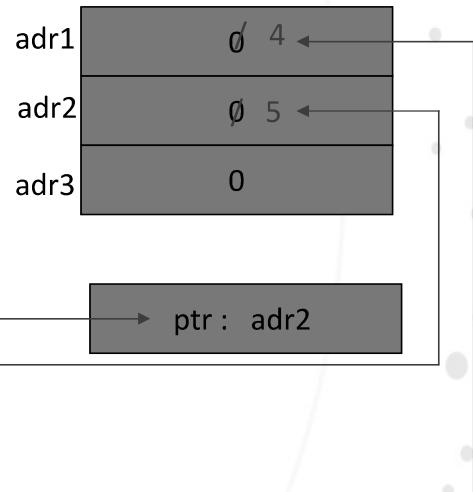
```
int * ptr=nom_tableau;
```

```
ptr++;
```

```
*ptr=5;
```

```
ptr--;
```

```
*ptr=4;
```



mais : nom_tableau=ptr; ↙ interdit car « nom_tableau » adresse constante.
« nom_tableau » est identique à « &nom_tableau » !!! (tableau statique)



Langage C : séance 5

Les tableaux statiques :

- Déclaration d'un tableau statique :

```
int nom_tableau[ 3 ]={0,0,0};
```

Accès par un autre pointeur :

```
int * ptr=nom_tableau;
```

Équivalence :

$$\text{ptr}[i] \rightarrow *(\text{ptr} + i)$$

On peut écrire : nom_tableau[i] ou *(nom_tableau + i)

mais attention :

sizeof(nom_tableau) → 12

sizeof(ptr) → 4 car ptr est un pointeur (adresse mémoire codé sur 4 octets).

sizeof(nom_tableau)/sizeof(nom_tableau[0]) → nbre de cases du tableau !





Langage C : séance 5



Passage par adresse de paramètres :

Exemple :

```
void lireEntier(int * j) {
    scanf("%d", j);
}

int main() {
    int i=5; ← Variable locale (au main)
    lireEntier(& i); ← La variable i est un paramètre réel ou effectif
    printf(" i=%d", i);
    return 0;
}
```

Paramètre formel de la fonction

Variable locale (au main)

La variable i est un paramètre réel ou effectif passée par adresse lors de l'appel à la fonction

La valeur du paramètre réel (& i) correspond bien à une adresse !



Langage C : séance 5



Passage par adresse de paramètres :

Exemple :

```
void lireEntier(int * j) {
    scanf("%d", j);
}

int main() {
    int i=5; ← Variable locale (au main)
    lireEntier(& i);
    printf(" i=%d", i);
    return 0;
}
```

i : 5

Etape 1 de l'exécution :

création de la variable i dans la pile (zone des variables locales) et initialisation





Langage C : séance 5



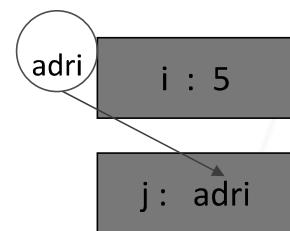
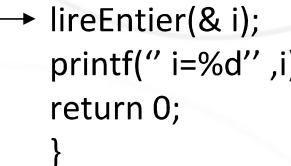
Passage par adresse de paramètres :

Exemple :

```
void lireEntier(int * j) {
    scanf("%d" , j);
}
```

Paramètre formel de la fonction

```
int main() {
    int i=5;
    lireEntier(& i);
    printf(" i=%d" ,i);
    return 0;
}
```



Etape 2 de l'exécution :

*création de la variable j dans la pile,
et initialisation avec la valeur du paramètre réel (&i → adri).*



Langage C : séance 5



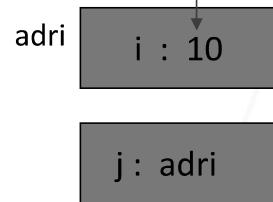
Passage par adresse de paramètres :

Exemple :

```
void lireEntier(int *j) {
    → scanf("%d" , j); → On tape 10
}
```

```
int main() {
    int i=5;
    lireEntier(&i);
    printf(" i=%d" ,i);
    return 0;
}
```

On tape 10



Etape 3 de l'exécution :

Saisie de la valeur d'un entier et stockage à l'adresse contenue dans le paramètre formel j.





Langage C : séance 5

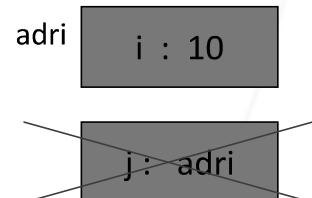


Passage par adresse de paramètres :

Exemple :

```
void lireEntier(int *j) {
    scanf("%d", j);
}
```

```
int main() {
    int i=5;
    lireEntier(& i);
    printf(" i=%d", i);
    return 0;
}
```



Etape 4 de l'exécution :

Retour dans le main : destruction des variables de la fonction lireEntier.



Langage C : séance 5



Passage par adresse de paramètres :

Exemple :

```
void lireEntier(int *j) {
    scanf("%d", j);
}
```

```
int main() {
    int i=5;
    lireEntier(& i);
    printf(" i=%d", i);
    return 0;
}
```



Etape 5 de l'exécution :

Affichage de la nouvelle valeur de i.

→ i=10





Langage C : séance 5



Passage par adresse de paramètres :

Résumé du passage par adresse :

- Le passage par adresse modifie le paramètre réel.

```
void f(T *t1);
T t=val1;
f(&t);      // la fonction f peut modifier la valeur de t
// t peut avoir une nouvelle
```

- Lors d'un passage par adresse, la valeur du paramètre réel ne peut pas être une constante.

```
void f(int *t1);
f( 10);    // interdit
```



La référence aéronautique

www.enac.fr

13



Langage C : séance 5

Déclaration et Réservation avec un pointeur (fonction malloc de <stdlib.h>):

- déclaration et initialisation à NULL.

```
int * pti= NULL;           adrpti
                           pti : 0
// NULL a pour valeur 0
```

Équivalent à :

```
int *pti;
pti=NULL;
```

- Réservation de l'espace en mémoire dynamique.

```
pti=(int *)malloc( sizeof(int));   adrpti
// réservation de l'espace pour un entier
*pti=5;                            pti : ad1
                                         ad1
                                         Ø 5
```

La fonction malloc retourne l'adresse de la zone mémoire allouée en mémoire dynamique, NULL si échec.
Elle reçoit en paramètre le nombre d'octets à réservé.

Les expressions suivantes : ont pour valeur :

pti	ad1
&pti	adrpti
*pti	5



La référence aéronautique

www.enac.fr

14



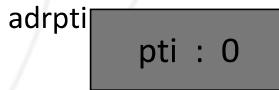
Langage C : séance 5



Déclaration et Réservation d'un tableau avec un pointeur :

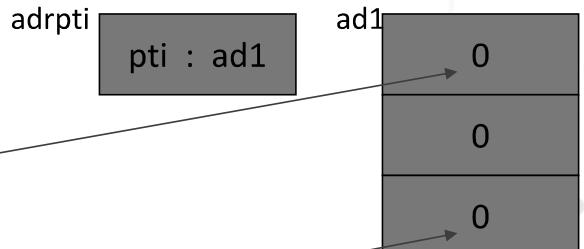
- déclaration et initialisation à NULL.

```
int * pti= NULL;
// NULL a pour valeur 0
```



- Réservation de l'espace en mémoire dynamique.

```
pti=(int * )malloc( sizeof(int) * 3);
// réservation de l'espace pour stocker 3 entiers.
```



on peut écrire ensuite :

*pti ou pti[0]

*(pti+i) ou pti[i]



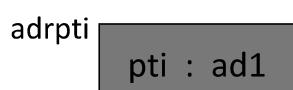
Langage C : séance 5



Déclaration et Réservation d'un tableau avec un pointeur :

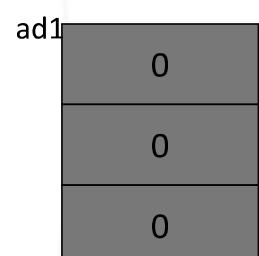
- déclaration et réservation.

```
int * pti= (int * )malloc( sizeof(int) * 3);
```



on peut écrire :

```
for (i=0; i<3; i++)
{
    printf(" valeur de pti[%d]=%d", i);
    scanf("%d", & pti[i]);
    // ou scanf ("%d", pti+i );
}
```



```
for (i=0; i<3; i++)
{
    printf(" valeur de pti[%d]=%d", i, pti[i]);
}
```





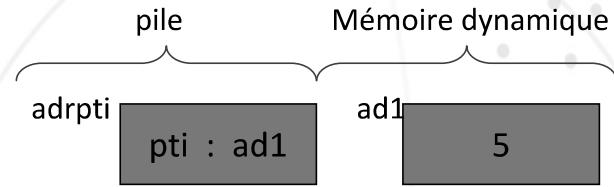
Langage C : séance 5



Libération mémoire en mémoire dynamique (fonction free de <stdlib.h>):

- déclaration et réservation.

```
int * pti= (int * )malloc( sizeof(int) );
*pti=5;
```

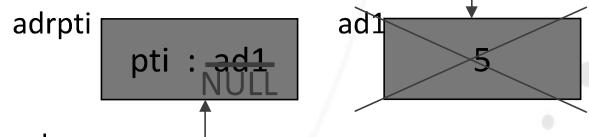


- Une zone allouée dans le tas (avec un malloc) doit être libérée si elle ne doit plus être utilisée.
→ libération par la fonction « free » (de <stdlib.h>)

free(pti);

La fonction free libère la zone allouée par le malloc.

// on écrase la valeur de l'adresse pour ne plus y retourner :
pti=NULL;



Langage C : séance 5



Arithmétique des pointeurs :

Une variable pointeur représentant une adresse, on ne peut qu'incrémenter une valeur à un pointeur ou soustraire deux pointeurs (pas d'addition, de multiplication ou de division de deux pointeurs).

- Incrémentation d'un pointeur :

```
int tab[5]={1,2,3,4,5};
int i,*pti;
pti= &tab[0];
for (i=0 ; i<5; i++,pti++)
    printf("%d \n", *pti);
```

- Différence de deux pointeurs :

```
int tab[10]={1,2,3,4,5};
int i,*ptr1, *ptr2;
ptr1= &tab[0];
ptr2=&tab[4];
printf("%d \n", ptr2-ptr1); // → 4
```





Langage C : séance 5



Autres fonctions d'allocation (dans <stdlib.h>) :

- la fonction « **calloc** » : **void* calloc(size_t n, size_t taille);**
(size_t correspond à un entier).

La fonction « **calloc** » permet d'allouer un nombre n (premier paramètre) de données de taille t (deuxième paramètre).

Exemple d'utilisation :

```
char * ch= calloc(10, sizeof(char)); // alloue un tableau de 10 char
```

Elle initialise également la zone allouée à 0 (ce que ne fait pas la fonction « **malloc** »).

- la fonction « **realloc** » : **void* realloc(void * ptr, size_t taille);**

Elle sert à réattribuer de la mémoire à un pointeur mais pour une taille mémoire différente. En cas d'agrandissement de la zone mémoire, elle déplace le contenu des données dans la nouvelle zone.

Exemple d'utilisation :

```
char * ch= calloc(10, sizeof(char)); // alloue un tableau de 10 char
strcpy(ch, "bonjour");
...
ch=realloc(ch, 20*sizeof(char)); // la nouvelle zone contient "bonjour"
```



La référence aéronautique

www.enac.fr

19



Langage C : séance 5



Déclaration et Réservation d'un tableau de pointeurs :

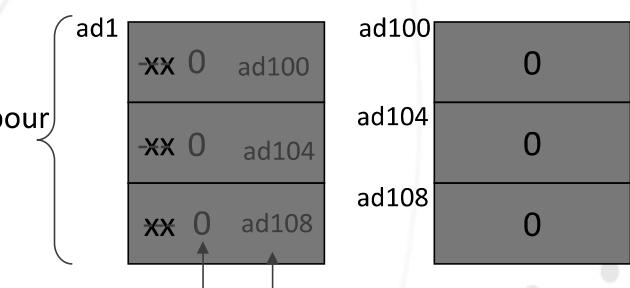
- déclaration (en variable locale) :

```
int * t[3];
int i;
for (i=0;i<3;i++)
    t[i]=NULL;
```

Zone réservée pour
le tableau t

pile

Mémoire dynamique



- Réservation :

```
for (i=0;i<3;i++)
    t[i]=(int *)calloc(1, sizeof(int));
```

xx → n'importe quelle valeur



La référence aéronautique

www.enac.fr

20



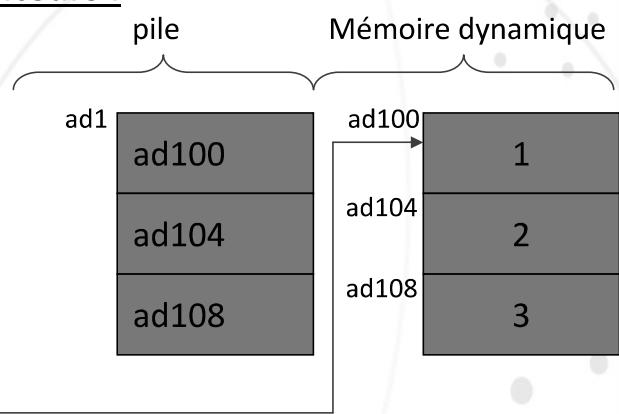
Langage C : séance 5



Déclaration et Réservation d'un tableau de pointeurs :

- utilisation de la variable t précédente :

```
// saisie des valeurs entières :
for (i=0;i<3;i++)
{
    printf("\nvaleur %d",i);
    scanf("%d", t[i]); // on tape 1,2,3
}
```



```
// affichage des valeurs entières :
for (i=0;i<3;i++)
    printf("\nvaleur=%d", *(t[i]));
```

- Libération :

```
for (i=0;i<3;i++)
{
    free(t[i]);
    t[i]=NULL;
}
```



Langage C : séance 5



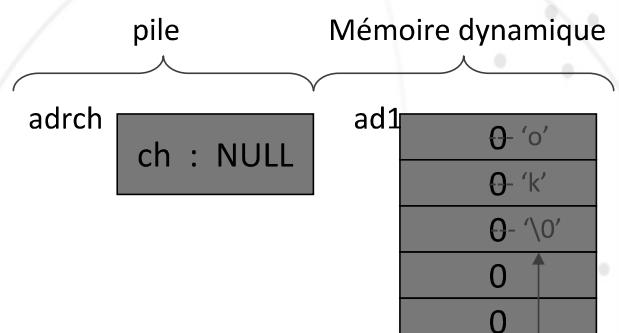
Déclaration et Réservation d'une chaîne de caractères dynamique :

- déclaration :

```
char * ch=NULL;
```

- Réservation :

```
ch=(char *)malloc(sizeof(char)*5);
// ou ch=(char *)calloc(5,sizeof(char));
// ou ch=(char *)realloc(ch,sizeof(char)*5);
```



- affectation :

```
strcpy(ch,"ok");
// saisie clavier :
scanf("%4s",ch);
// affichage écran :
printf("\n %s",ch);
```





Langage C : séance 5



Déclaration et Réservation d'un tableau multidimensionnel :

- déclaration :

```
int ** t=NULL;
```

- Réservation (pour n lignes de c colonnes):

// réservation des n lignes :

```
t=(int **)calloc(n,sizeof(int *));
```

// réservation de c colonnes d'entiers pour chaque ligne :

```
for (i=0;i<2;i++)
```

```
    t[i]=(int *)calloc(c, sizeof(int));
```

- utilisation :

```
for (i=0;i<n;i++)
```

```
    for (j=0;j<c;j++)
```

```
        t[i][j]=0; // ou scanf("%d",& t[i][j]); ou printf("%d",t[i][j]);
```

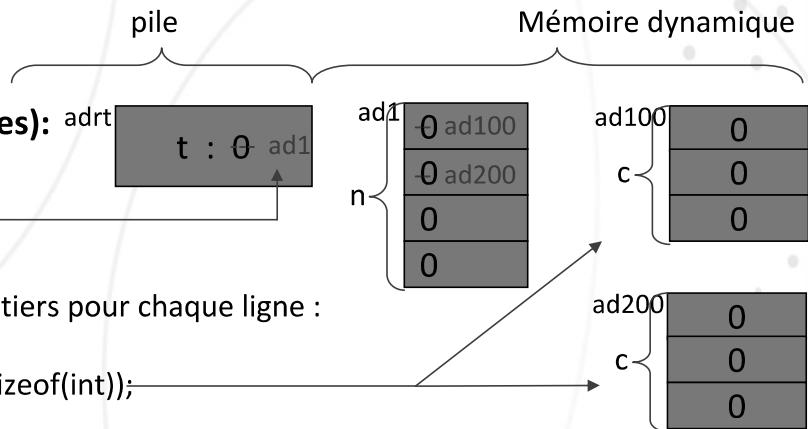
- libération :

```
for (i=0;i<2;i++)
```

```
    free(t[i]);
```

```
free(t);
```

```
t=NULL;
```



www.enac.fr

23



Langage C : séance 6



Séance 6 : Les structures et chaînes de caractères



www.enac.fr

1



Langage C : séance 6



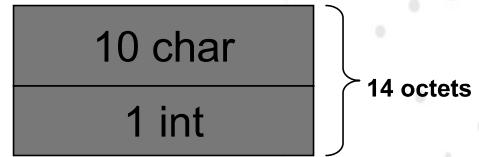
Les structures :

Une structure permet de définir un regroupement de données de types différents.

• Syntaxe 1 :

```
struct element {
    char nom[10];
    int age;
};

struct element e; // e est une variable
```



• Syntaxe 2 :

```
struct element {
    char nom[10];
    int age;
} e; // e est une variable
```



La référence aéronautique

www.enac.fr

2



Langage C : séance 6



Les structures :

• Syntaxe 3 : on utilise un `typedef`.

```
typedef struct {
    char nom[10];
    int age;
} element; // element est un nouveau type
element e; // e est une variable
```

• Utilisation : valable pour les 3 syntaxes précédentes.

Ou

```
e.age= 20;
scanf("%d",& e.age);
printf("%d", e.age);
```

Ou

```
strcpy(e.nom,"dupont");
scanf("%9s",e.nom); // e.nom est une chaîne de caractères
printf("%s",e.nom);
```



La référence aéronautique

www.enac.fr

3



Langage C : séance 6

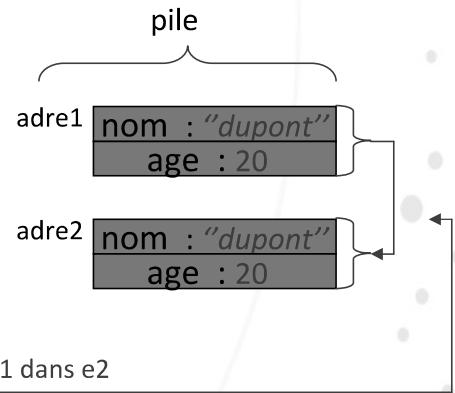


Affectation entre variables structure :

On peut affecter une variable structure dans une autre de même type.

```
element e1 , e2 , t [3]; // réservation par le système dans la pile
e1.age= 20;
strcpy(e1.nom,"dupont");

e2=e1;
t[0] = e1;
```



Langage C : séance 6

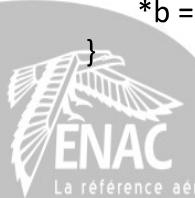


Les pointeurs sur structures :

Pour modifier des variables de type structure dans une fonction , on effectue un passage par adresse

```
int main()
{
    element e1, e2 ; // réservation par le système dans la pile
    ... ;           //initialisation des variables e1 e2
    permutation (&e1,&e2);
}

void permutation (element * a , element * b)
{
    element temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```





Langage C : séance 6



Les pointeurs sur structures :

Pour modifier des variables de type structure dans une fonction , on effectue un passage par adresse.

```
int main()
{
    element e1, e2 ; // réservation par le système dans la pile
    init( &e1,&e2) ;
    ...
}

void init(element * a , element * b)
{
    (*a) .age = 0;           // parenthèses indispensable car priorité des opérateurs !!!
    strcpy( (*a) . nom, « inconnu » );
    b -> age = 0;          // b->age est identique à (*b).age
    strcpy( b->nom, « inconnu » );
}
```

*L'écriture **(*a).age** est identique à **a->age**.
On privilégiera l'utilisation de **->** à celle de *****.*



La référence aéronautique

www.enac.fr

6



Langage C : séance 6



Les pointeurs sur structures :

Pour modifier des variables de type structure dans une fonction , on effectue un passage par adresse

```
int main()
{
    element e1, e2 ; // réservation par le système dans la pile
    saisir( &e1,&e2) ;
    ...
}

void saisir(element * a , element * b)
{
    ...
    scanf(« %d », &(*a) .age) ;
    scanf(« %s », (*a) . nom) ;
    ...
    scanf(« %d », & b-> age ) ;
    scanf(« %s », b->nom ) ;
}
```



La référence aéronautique

www.enac.fr

7



Langage C : séance 6



Les pointeurs sur structure :

On peut déclarer un pointeur sur structure.

•déclaration 1:

```
element e1; // réservation par le système dans la pile
element *e2; // e2 est un pointeur sur structure
e2=&e1; // e2 pointe sur la structure e1

(*e2).age= 20; // ou e2->age=20;
scanf("%d",& (*e2).age); // ou scanf("%d",& e2->age);
printf("%d", (*e2).age); // ou printf("%d", e2->age);

strcpy((*e2).nom,"dupont"); //ou strcpy(e2->nom,"dupont");
scanf("%9s", (*e2).nom); // ou scanf("%9s",e2->nom);
printf("%s",(*e2).nom); // ou printf("%s",e2->nom);
```

Rm : e1 n'est pas une adresse, e1 représente l'ensemble des données (age et nom).
&e1 correspond à l'adresse de la structure en mémoire (adresse1 ici).



Langage C : séance 6



Les pointeurs sur structure :

On peut allouer une structure en mémoire dynamique.

•déclaration 2:

```
element *e=NULL; // réservation par le système dans la pile
e=malloc(sizeof(element)); // allocation en mémoire dynamique

e->age= 20; // ou (*e).age=20;

strcpy(e->nom,"dupont"); //ou strcpy((*e).nom,"dupont");
...
free( e ); // libération en mémoire dynamique
e=NULL;
```



Langage C : séance 6



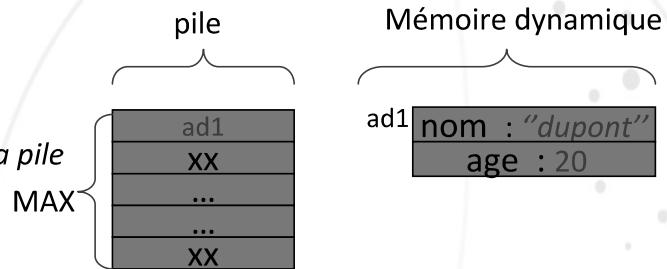
Tableau de pointeurs sur structure :

On peut déclarer un tableau de pointeurs sur structure.

•déclaration 3:

```
#define MAX 100
element * t[MAX]; // réservation par le système dans la pile
// allocation en mémoire dynamique :
t[0]=malloc(sizeof(element)); // → ad1

t[0]->age= 20;
strcpy(t[0]->nom,"dupont");
...
free( t[0]);
t[0]=NULL;
```



Langage C : séance 6



Les priorités des opérateurs :

1	Fonction Tableau Champ de structure	() [] -> .
2	Negation booleenne In(De)crementation Indirection Adresse	! ++ -- * &
3	Multiplication Division Modulo	*
4	Addition Soustraction	+
5	Relation logique	< <= > >=
6	Egalite	== !=
7	Et booleen	&&
8	Ou booleen	
9	Affectation	= += -=





Langage C : séance 6

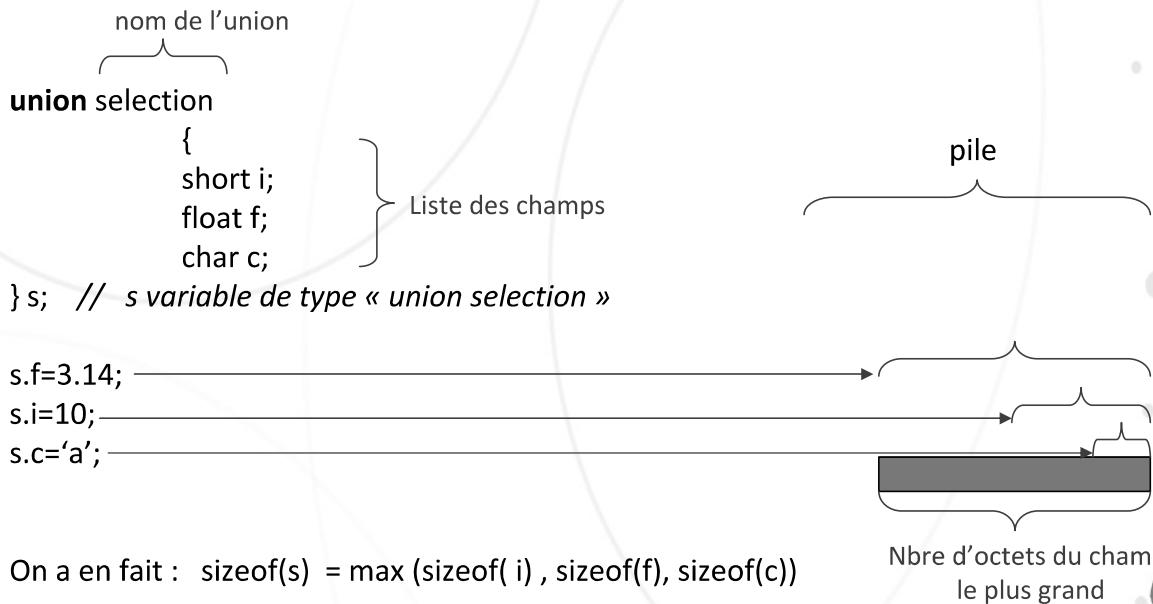


Les Unions :

Une union est une structure qui ne peut avoir qu'un seul champ actif à la fois.

- Syntaxe :

Exemple :



Langage C : séance 6



Les chaînes de caractères : on utilise un tableau de caractères.

- syntaxe :

```
char machaine[Nbre_car];
```

- dernier caractère de la chaîne : '\0' (\zéro) de valeur 0 (zéro)

- initialisation :

```
char tab[]={ 'a','e','i','o','u','y'}; // taille 6, pas de '\0' à la fin
char tab[]="aeiouy" ; // taille 7 car '\0' placé à la fin
```





Langage C : séance 6



Saisie de chaînes de caractères avec « scanf » :

- Saisie limitée de caractères :

scanf(" %5s" , ch);

Avec 5, nombre max de caractères à saisir.

- Saisie comprenant un (ou plusieurs) espace(s) :

scanf("%[^n]" , ch);

- Saisie avec espace et 5 caractères max par exemple :

scanf("%5[^n]" , ch); // avec 5 nombre max de car.



La référence aéronautique

www.enac.fr

14



Langage C : séance 6



La fonction « sscanf » :

- Permet d'extraire des données à partir d'une chaînes de caractères.

- Syntaxe : sscanf(*dans* , *comment* , *où*);

Recherche l'information dans le premier argument, sous la forme du *comment*, et le range dans le *où*.

- exemple :

```
int entier;
float reel;
char tampon[51];
...
printf("entrez le message \n");
scanf("%50[^n]", tampon);
sscanf(tampon,"%d %f",&entier,&reel);
```

// on tape : «12 3.14»

// entier vaut 12, reel vaut 3.14



La référence aéronautique

www.enac.fr

15





Langage C : séance 6



La fonction « sprintf » :

- Permet de convertir en chaîne de caractères des données numériques.
- Syntaxe : `sprintf(chaîne résultat, format, variables);`
Recherche l'information dans le(s) dernier(s) argument(s), sous la forme du format, et le stocke dans la chaîne résultat.
- exemple :


```
int entier=10;
float reel=3.14;
char tampon[50];
...
sprintf(tampon,"%d %f",entier,reel);
printf("%s", tampon); // affiche la chaîne "10 3.14"
```
- Privilégier l'usage de `snprintf` : ➔ à privilégier
`snprintf(tampon,50, "%d %f",entier,reel); // évite le débordement du tampon (50 max ici)`



La référence aéronautique



Langage C : séance 6



Fonctions standards de traitement sur les chaînes (string.h) :

- fonction pour calculer la longueur : **strlen**

```
char ch[]="dupont"; // ajout de '\0' en mémoire
int i=strlen(ch); // i vaut 6
```

La chaîne passée en paramètre à **strlen** doit se terminer obligatoirement par '\0'.

- fonction d'affectation d'une chaîne : **strcpy**

```
char ch1[20], ch2[20];
strcpy(ch1, "dupont" ); // ch1 reçoit « dupont »
strcpy(ch2, ch1);
```

- fonction d'affectation restreinte : **strncpy** ➔ à privilégier

```
char ch1[20], ch2[20];
strncpy(ch1, "dupont",4 ); // ch1 reçoit « dupo »
strncpy(ch2, ch1,3); // ch2 reçoit « dup »
attention : pas d'ajout de '\0' par strncpy !
```



La référence aéronautique





Langage C : séance 6

Fonctions standards de traitement sur les chaînes (string.h) :

- fonction de concaténation : **strcat**

```
char ch1[20], ch2 []="dupont";
strcpy(ch1, "bonjour ");
strcat(ch1,ch2);           // ch1 vaut "bonjour dupont"
```

- fonction de concaténation restreinte : **strncat**

← à privilégier

```
char ch1[20], ch2 []="dupont";
strcpy(ch1, "bonjour ");
strncat(ch1,ch2, 3);      // ch1 vaut "bonjour dup"
Cette fonction rajoute un '\0' à la fin.
```

- fonction de comparaison : **strcmp**

```
strcmp(ch1,ch2) // comparaison dans l'ordre alphanumérique !
→ retourne 0 (zéro) si les deux chaînes sont égales.
→ retourne une valeur négative si ch1 < ch2 (ex: "dupantel" < "dupont")
→ retourne une valeur positive si ch1 > ch2 (ex: "dupont" > "dupantel")
```

- fonction de comparaison restreinte : **strncmp**

← à privilégier

```
strncmp(ch1,ch2,n) // comparaison des n premiers caractères
→ idem que strcmp
```



Langage C : séance 6

Les paramètres du « main » :

• La déclaration du main :

La fonction « main » avec paramètres permet de récupérer des valeurs lors de l'exécution du programme (lancement en ligne). Ces valeurs sont récupérées dans des chaînes de caractères.

• Syntaxe :

```
int main(int argc, char * argv[])
{
    /* argc : nombre total d'expression
       argv : tableau de chaînes de caractères
       argv[0] : nom de la commande
       argv[1] : 1er paramètre, ... */
    ....
}
```

Nombre d'expressions de la commande

Tableau contenant chacune des expressions





Langage C : séance 6



Les paramètres du « main » :

•**Exemple :**

Fichier « toto.c » :

```
int main(int argc, char * argv[]) {  
    int i;  
    printf("nombre total =%d\n",argc);  
    for (i=0;i<argc;i++)  
        printf("%s\n",argv[i]);  
    return 0;  
}
```

Exécution dans un terminal :

```
prompt # ./toto bonjour 1 2 3  
nombre total =5  
toto  
bonjour  
1  
2  
3
```



La référence aéronautique

www.enac.fr

20



Langage C : séance 8



Séance 8 : les fichiers



La référence aéronautique

www.enac.fr

1



Langage C : séance 8



Découpage modulaire d'un programme en langage C :

Un programme écrit en langage C est rarement écrit dans un seul module (ou fichier). Le code source s'étend souvent sur plusieurs modules, dont chacun a un rôle bien précis.

Nous allons à travers un exemple comprendre comment décomposer son programme en modules différents et voir comment est construit le programme final (l'exécutable).

Etape 1 : création des modules d'en-tête.

Il s'agit de fichiers ayant une extension « .h » (h pour header). Ces fichiers contiennent des définitions de types et les déclarations des fonctions à développer.

Etape 2 : création des modules de définitions des fonctions.

Ces fichiers reprennent les fonctions déclarées précédemment et les définissent (ajout du corps de la fonction).

Etape 3 : création du programme principal (main).

Le programme « main » appelle les fonctions.



Langage C : séance 8



Découpage modulaire d'un programme en langage C :

Fichier « mesfonctions.h »

```
#include <stdio.h>
#ifndef _MESFONCTIONS_H
#define _MESFONCTIONS_H
void mafonction(char * ch);
#endif
```

Fichier « mesfonctions.c»

```
#include "mesfonctions.h"
void mafonction(char * ch)
{
    printf("message : %s \n",ch);
```

Fichier « monprogramme.c »

```
#include <stdio.h>
#include "mesfonctions.h"
int main()
{
    // appel de la fonction
    mafonction("bonjour ");
    return 0;
}
```

Dans chaque module « .h », on doit commencer par les deux directives au préprocesseur #ifndef et #define pour éviter les inclusions multiples, et terminer par #endif.

Dans les modules de définitions de fonctions (ici le fichier « mesfonctions.c »), on inclut (directive #include) le module d'en-tête correspondant (ici « mesfonctions.h »).

Dans le programme principal (ici « monprogramme.c »), on inclut le module d'en-tête.





Langage C : séance 8



Les variables à travers les modules :

• Variable globale :

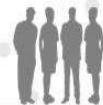
Rappel : - déclarée en dehors de tout bloc.
 - visible de tout le fichier.
 - zone mémoire : zone de données.
 - durée de vie permanente (celle du main).
 - accessible par le main et toutes les fonctions !

• Variable globale static (ou fonction static):

idem que variable globale mais rend celle-ci invisible à l'extérieur du fichier.
 Une fonction static n'est visible que dans le fichier où elle est déclarée.

• Variable globale extern (ou fonction extern):

Indique que l'emplacement réel et la valeur initiale d'une variable, ou
 du corps d'un sous-programme sont définis ailleurs (autre fichier).



Langage C : séance 8



Les variables à travers les modules :

```
//fichier « fic1.c »
...
int vari=1; //variable globale
static int varsj=0 ; //variable globale static
static void f1(); //fonction static
...
void f2() {
...
vari=3; // autorisé
varsj=8; // autorisé
f1(); // autorisé
}
```

```
//fichier « fic2.c »
...
extern int vari; // autorisé
extern void f2(); // autorisé
void f3() {
f2(); // autorisé
f1(); // accès interdit
varsj=5; // accès interdit
vari=10; // autorisé
}
```





Langage C : séance 8



Les variables à travers les modules :

• Variable locale :

- déclarée dans un bloc.
- visible uniquement dans le bloc ou les blocs imbriqués.
- zone mémoire : en pile (stack).
- durée de vie temporaire : détruite à la sortie du bloc.

• Variable locale static :

- Permet de conserver l'état de la variable entre deux exécutions d'une fonction.
- La variable n'est initialisée qu'une seule fois.
- zone mémoire utilisée : celle des variables globales.



Langage C : séance 8



Les variables à travers les modules :

```
// fichier « fic.c »

...
void f1() {
    static int s_i=3; // variable locale static
    printf("s_i = %d\n",s_i);
    s_i++;
}
int main() {
f1();
f1();
f1();
return 0;
}
```

Affichage obtenu :

s_j = 3
s_j = 4
s_j = 5





Langage C : séance 8



Les fichiers de données :

On peut différencier deux types de fichiers de données utilisés par un programme en langage C :

-**Fichier texte** : il contient des octets dont chaque octet correspond à un caractère. Ce type de fichier peut être créé et modifié par un éditeur de texte. Ce sont des fichiers au format «.txt ».

-**Fichier binaire** : il contient des octets dont on ne peut pas à priori connaître la signification. Seul celui qui y a écrit les octets sera capable de récupérer correctement les données.

Exemple : j'écris dans un fichier le contenu d'une variable « int » (4 octets) et le contenu d'un « char » (1 octet). Ce fichier contient donc 5 octets de données. Si je dois relire ces données dans le fichier, je dois d'abord lire 4 octets puis 1 octet pour retrouver correctement les valeurs de mes données.



Langage C : séance 8



Les fichiers « texte »:

On peut accéder aux contenus des fichiers « texte » avec des fonctions de <stdio.h>.

• Type de variables :

On doit utiliser des variables de type : FILE *

```
FILE * fic=NULL; // on appellera « fic » une variable logique de fichier.
```

- Ouverture de fichier : fonction « fopen ».
- Traitement de lecture : fonctions « fscanf », « fgets », « fgetc ».
- Traitement d'écriture : fonctions « fprintf », « fputs », « fputc ».
- Fermeture de fichier : fonction « fclose ».





Langage C : séance 8



Ouverture de fichier (texte ou binaire):

On doit ouvrir le fichier en spécifiant son nom physique (dans l'arborescence des fichiers) et son mode d'ouverture : lecture, écriture, ajout.

La valeur renvoyée par la fonction « fopen » permet d'initialiser la variable logique pour accéder au contenu du fichier.

La fonction retourne NULL en cas d'échec (à tester après chaque ouverture).

- la fonction « fopen ».

FILE * fic=NULL; // car fic doit être un pointeur.

fic=fopen("nomfichier.txt" , "r");

Mode d'ouverture possible :

"r" : lecture, fichier existant.

"w": écriture (écrasement), création si inexistant.

"a": écriture à la fin(sans écrasement), création si inexistant.

"r+": lecture/écriture à la fin, fichier existant.

"w+": lecture/écriture (avec écrasement), création si inexistant.

"a+": lecture/écriture à la fin, création si inexistant.

Nom physique du fichier (avec l'extension !)

Ensuite : if (fic != NULL) // ouverture avec succès !



Langage C : séance 8



Fermeture d'un fichier (texte ou binaire):

- la fonction « fclose ».

Il ne faut pas oublier de fermer un fichier ouvert par un fopen par la fonction fclose.

```
FILE * fic=NULL;
fic=fopen( "nomfichier.txt" , "r" );
if (fic !=NULL)
{
    ... traitement
    fclose(fic);           // fermeture du fichier si ouverture réussie uniquement !
}
else
{
    printf(" echec ouverture fichier \n");
    exit (1);             // la fonction exit permet d'arrêter le programme (→arrêt du main !)
}
```





Langage C : séance 8



Lecture dans un fichier texte:

- la fonction « fscanf ».

Elle permet de lire un ensemble de valeurs du fichier vers la mémoire.

Syntaxe : int fscanf (FILE * stream, char * chaine, liste d'adresses);

Retourne le nombre d'objets lus.

stream : variable fichier.

chaine et liste adresses comme dans un scanf.

Utilisation :

```
int i;
float f;
if (fscanf(fic, "%d %f", &i, &f) == 2)
    printf("%d %f", i,f);
```



Langage C : séance 8



Lecture dans un fichier texte:

- la fonction « fgets ».

Elle permet de lire une ligne sous la forme d'une chaîne de caractères du fichier vers la mémoire.

Syntaxe : char * fgets (char * chaine, int n, FILE * stream);

Lit au plus n-1 caractères (ou jusqu'au '\n') du fichier et les place dans chaine.

Rajoute un '\0' en fin de chaine.

Retourne un pointeur sur chaine ou NULL.

Utilisation :

```
char chaine[21];
if (fgets(chaine, 21, fic) != NULL)
    printf("%s", chaine);
```





Langage C : séance 8



Lecture dans un fichier texte:

- la fonction « fgetc ».

Elle permet de lire un caractère du fichier vers la mémoire.

Syntaxe : int fgetc (FILE * stream);

Lit un caractère du fichier et le retourne ou EOF en cas d'échec.

Utilisation :

```
char car;
car=fgetc(fic);
printf("%c", car);
```



Langage C : séance 8



Ecriture dans un fichier texte:

- la fonction « fprintf ».

Elle permet d'écrire sous la forme d'une chaîne de caractères un ensemble de valeurs de la mémoire vers le fichier.

Syntaxe : int fprintf (FILE * stream, char * chaine, liste de variables);

Retourne le nombre d'objets écrits.

stream : variable fichier.

chaine et liste variables comme dans un printf.

Utilisation :

```
int i=10;
float f=3.14;
if (fprintf(fic, "%d %f", i, f) == 2)
    printf("transfert réussi");
```





Langage C : séance 8



Ecriture dans un fichier texte:

- la fonction « fputs ».

Elle permet d'écrire une chaîne de caractères de la mémoire vers le fichier.

Syntaxe : int fputs (char * chaine, FILE * stream);

Retourne une valeur non négative si succès ou EOF en cas d'erreur.
chaine : doit se terminer par le caractère '\0'.

Utilisation :

```
char chaine[10]= "dupont";
if (fputs(chaine,fic) != EOF)
    printf("transfert réussi");
```



Langage C : séance 8



Ecriture dans un fichier texte :

Elle permet d'écrire un caractère de la mémoire vers le fichier.

- la fonction « fputc ».

Syntaxe : int fputc (char c, FILE * stream);

Retourne le caractère c en cas de succès sinon EOF.

Utilisation :

```
char c='A';
if (fputc(c,fic) != EOF)
    printf("transfert réussi");
```





Langage C : séance 8



Déplacement dans un fichier (texte ou binaire):

- la fonction « ftell ».

Syntaxe : long ftell (FILE * stream);

renvoie la position du pointeur de fichier en nombre d'octets par rapport au début du fichier.

- la fonction « rewind ».

En mode d'accès lecture/écriture sur un fichier, on peut se repositionner en début de fichier avec un appel à la fonction « rewind »:

Syntaxe : void rewind(FILE * stream);



Langage C : séance 8



Déplacement dans un fichier (texte ou binaire):

- la fonction « fseek ».

La fonction « fseek » permet de se positionner sur un octet précis dans un fichier.

Syntaxe : int fseek (FILE * stream, long nb, int position);

Déplace le pointeur de déplacement dans le fichier d'un nombre « nb » d'octets plus loin de l'emplacement mentionné par position.

Retourne 0 si déplacement correct, valeur non nulle si erreur.

« position » doit être une des expressions suivantes :

SEEK_SET : début de fichier.

SEEK_CUR : position courante.

SEEK_END : fin de fichier.

Utilisation :

fseek(fic, 10, SEEK_SET); // se déplace sur le 10 ième octets du fichier.





Langage C : séance 8



Traitement sur les fichiers (texte ou binaire):

- la fonction « remove ».

Le fichier doit être fermé. Cette fonction permet de détruire physiquement un fichier.
On spécifie en paramètre le nom physique du fichier.

```
test=remove("nomfichier.txt");
```

Valeur renournée : 0 si succès, -1 sinon.

- la fonction « rename ».

Le fichier doit être fermé. Cette fonction permet de renommer physiquement un fichier.

```
test=rename("anciennom","nouveaunom");
```

Valeur renournée : 0 si succès, -1 sinon.



Langage C : séance 8



Lecture dans un fichier binaire:

- la fonction « fread ».

Elle permet de lire un groupe d'octets du fichier vers la mémoire.

Syntaxe : size_t fread(void * ptdebut, size_t taille, size_t n, FILE * stream);

Retourne le nombre d'objets lus (valeur du troisième paramètre).

n : nombre d'objets à lire.

taille : dimension en octets d'un objet.

stream : variable fichier.

ptdebut : adresse du bloc mémoire où écrire l'info lue.

Utilisation :

```
int i;
if (fread(&i, sizeof ( int), 1 , fic) == 1)
    printf("%d", i);
```

```
int t[10];
if (fread(t, sizeof ( int), 10 , fic) == 10)
    printf("récupération réussie");
```





Langage C : séance 8



Ecriture dans un fichier binaire :

- la fonction « fwrite ».

Elle permet d'écrire un groupe d'octets de la mémoire vers le fichier.

Syntaxe : size_t fwrite(void * ptdebut, size_t taille, size_t n, FILE * stream);

Retourne le nombre d'objets écrits (valeur du troisième paramètre).

n : nombre d'objets à écrire.

taille : dimension en octets d'un objet.

stream : variable fichier.

ptdebut : adresse du bloc mémoire où lire l'info.

Utilisation :

```
int i=10;  
if (fwrite(&i, sizeof ( int), 1 , fic) == 1)  
    printf("transfert réussi");
```

```
int t[5]={1,2,3,4,5};  
if (fwrite(t, sizeof ( int), 5 , fic) == 5)  
    printf("écriture réussie");
```

