Programmation en C norme ANSI

Référence : L1

Guide de l'étudiant

une division de
Sun Microsystems France S.A.
Service Formation
BP 53
13, avenue Morane-Saulnier
78142 Vélizy Cedex
tél : (1) 30 67 50 50
fax : (1) 30 67 52 35

Révision C, Décembre 1994
Document non révisable

Credits and Trademarks

The OPEN LOOK and the Sun Graphical User Interface were developed by Sun Microsystems, Inc. for its uses and licenses. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licenses.

The products described in this manual may be protected by one or more U.S. patents, foreign patents, and / or pending applications.

Table des Matières

Annexes

Structures Fondamentales d'un Programme en C

Objectifs

- Ecrire des programmes C syntaxiquement corrects.
- Identifier les éléments d'un programme C.
- Faire correspondre opérateurs et opérations.
- Identifier et déclarer les types de base.

Evaluation

Travaux Pratiques 1 et révision de module.

Compilation Simple

- Le code source C doit être mis dans un fichier dont le nom se termine par *.c*
- Utiliser la commande cc avec l'option *-Xc* pour compiler les programmes C ANSI.
- Si la compilation réussit, le fichier exécutable sera appelé a.out par défaut.

```
%  cc -Xc prog.c

%  a.out

<affichage de résultats s'il y en a>
```

Caractéristiques du C

- Bas niveau - haut niveau
- Très Portable
- Mise en forme libre
- Aucune possibilité d'Entrée/Sortie intégrée : beaucoup de fonctions en librairie Standard

Introduction au Source C-La Fonction *main()*

Tout programme C doit contenir la fonction *main()*:

La fonction C universelle *main()*
et une fonction de sortie-écran:

```
/* Voilà la fonction "main"
        et içi des commentaires.
*/

int main (void)
{
    printf("Bienvenue en Programmation C!");
    return 0;
}
```

liste de paramètres entre parenthèses
(pas d'argument dans l'exemple)

corps de fonction (ou bloc) entre 2 accolades

nom de fonction

commentaires entre les marques "/*" et "*/"

Eléments de Base d'un Programme C

Un programme C se compose de *déclarations* et d'*instructions.*

```
/* Ceci montre uniquement un squelette de programme.
   Ce n'est pas un programme compilable */

int main(void)
{
      <data type> <identifier>; /* déclaration */
      <data type> <identifier, identifier, ...>;/* déclaration */
      <data type> <identifier> = <value>;/* initialisation */

      <statements>      /* instructions */
      return 0;
}  /* fin de la fonction main() */
```

Types de Données

- Ci-dessous, la liste des types du C et leurs tailles sur une SPARCstation™:

mot-clef	description	Taille en octets
`char`	caractère	1
`short`	entier court	2
`int`	entier	4
`long`	entier long	4
`float`	réel simple précision	4
`double`	réel double précision	8
`long double`	réel en précision étendue	16
`void`	aucune valeur	0

- Les types `char`, `short`, `int`, et `long` peuvent être précédés des attributs de types `signed` et `unsigned`. Exemples:

```
signed char ch;

unsigned int compteur;

unsigned long nombre;
```

Attributs de Types

- Chaque type peut être préfixé par un ou deux des attributs suivant :

Mot-clef	Description
`const`	L'identifiant représente une constante qui doit être initialisée, mais jamais modifiée.
`volatile`	Le compilateur doit générer des mises à jour de l'identifiant à certains points du programme.

- Exemples :

```
const int constant=10;
volatile double xyz;
const volatile int clock;
```

Identifiants du Langage C

- Exemples d'identifiants légaux :

```
count

num_2

DayOfWeek

IDENT
```

- Exemples d'identifiants illégaux :

```
not#me      /*caractère spécial "#" interdit*/

101notme    /* pas de chiffre en premier*/

-notme      /* "-" erroné pour  "_" */

@%^&*?~+$   /* toutes les ponctuations et
               autres caractères spéciaux */
```

Expressions et Valeurs Constantes

Constantes Entières	Notation	Base
1024	decimale	10
074	octale	8
0xFFFF	hex	16
0xa4d2	hex	16

Constantes en virgule flottante

Notation Décimale	634.5789
Notation Exponentielle	8675309E3
	986e-2
Notation Scientifique	5.76E1
	1.0e3

Constantes caractères et principales Séquences d'Escape

'a' 'b' 'c', 'A' 'B' 'C', etc.	Caractères alphabétiques
'1' '2' '3' '4' '5', etc.	Caractères numériques
'!' '@' '>' '$' '&', etc.	Autres caractères imprimables

'\n'	newline	'\f'	form feed
'\b'	backspace	'\''	single quote
'\t'	tab	'\a'	bell character
'\\'	backslash	'\?'	question mark
'\x4c'	hex digits	'\127'	octal digits
'\v'	vertical tab	'\0'	null character

Suffixes de Constantes et Trigraphes

Suffixes de constantes entières	signification
13*L*	entier long
123*l*	entier long
25*U*	entier non signé
33*u*	entier non signé
12*UL*	entier long non signé
54*lu*	entier long non signé

Suffixes de constantes en virg.flot.	signification
22E3*f*	simple précision (`float`)
43.219*F*	simple précision (`float`)
52e-3*L*	précision étendue (`long double`)
2.124*l*	précision étendue (`long double`)

Trigraphes	Caractères
??=	#
??-	~
??(	[
??)	]
??<	{
??>	}
??’	^
??!	|
??/	\

Expressions et Instructions

- Exemples d'expressions :
  - constantes
  - expressions arithmétiques comme `num * 7`
  - expressions logiques comme `num <= 10`
  - affectations : `num16 = 32767`
  - appels de fonctions
- Exemples d'instructions :
  - structures itératives comme `for` ou `while`
  - structures conditionnelles comme `if` ou `switch`
  - expressions suivies d'un point-virgule
  - affectations suivies d'un point-virgule
  - appels de fonction suivis d'un point-virgule

Introduction aux Fonctions d'Affichage

Sorties

- La fonction *printf()* est utilisée pour l'instant. D'autres fonction seront vues plus loin.

- Tous les *printf()* vont ressembler à :

```
printf(<une chaîne de caractères et/ou une spécification de format>,
<arguments>);
```

- Pour écrire un entier, utiliser *%d* comme spécification de format ; pour un caractère, utiliser *%c*.

- Si aucune spécification de format n'est présente, la liste d'argument doit être vide.

- Exemples :

```
printf("Hello, ceci est juste une chaîne de caractères.\n");

printf("Ceci imprime un entier %d et un retour-chariot.\n", 100);

printf("Ceci imprime un caractère %c et un retour-chariot.\n", 'Z');
```

Introduction aux Fonctions de Saisie

Entrées

- La fonction *scanf()* est utilisée pour l'instant. D'autres fonctions seront vues plus loin.

- Pour l'instant, tous les *scanf()* vont ressembler à :

scanf(<spécification de format>, &<nom_identifiant>);

- Pour lire un entier, utiliser le spécificateur *%d* ; pour lire un caractère, utiliser le spécificateur *%c*.

- Exemples :

```
int intgr;
char ch;
scanf("%d", &intgr);
scanf("%c", &ch);
```

Un Programme Exemple Simple

Ce programme montre plusieurs déclarations et instructions, ainsi que l'utilisation de *scanf()* et *printf()*:

```
/* Ce programme va déclarer quelques variables entières et
   une variable caractère, leur affecter des valeurs (soit
   explicitement soit par scanf()), et enfin imprimer ces valeurs */

int main(void)
{
    int num1;        /* déclaration de num1 */
    int num2 = 10;   /* déclaration et initialisation de num2 */
    char ch;         /* déclaration d'une variable caractère */

        ch = 'a';
        num1 = 5;    /* instructions... */
   printf("les entiers valent %d et %d.\n", num1, num2);
   printf("le caractère est %c.\n", ch);
   return 0;
} /* fin de la fonction main */
```

Blocs

- Les blocs peuvent contenir des déclarations et des instructions.
- Les blocs sont aussi utilisés pour les contrôles de flux et les structures itératives.
- Le corps d'une fonction est contenu dans un bloc "{ }":

```
int main(void)
{
     int num1, num2;
     int sum;

     num1 = 8;        /* instructions d'affectation */
     num2 = 3;

     {    /* autre bloc */
       char letter; /* letter n'est vue que dans ce bloc */
       letter = 'z';
       printf("letter vaut :  %c;inconnue hors ce bloc\n",
          letter);
     }    /* fin du bloc */

     sum = num1 + num2;
     printf ("La somme de %d et %d égale %d.\n",
     num1,num2,sum);
     printf ("Le produit est %d.\n", num1 * num2);
     return 0;
} /* fin du bloc de la fonction main */
```

Opérateurs - Premier Exposé

Opérateurs arithmétiques

+	addition et plus unaire
-	soustraction et moins unaire
*	multiplication
/	division (entière et virgule flottante)
%	modulo (reste)
++, --	incrément et décrément

Opérateurs Relationnels

==	égal
!=	différent
>	strictement supérieur
<	strictement inférieur
<=	inférieur ou égal
>=	supérieur ou égal

Opérateurs d'affectation

=	affectation simple
op=	affectation composée - où *op* est n'importe quel opérateur arithmétique ou sur bits

Opérateurs sur bits

&	*et* bit-à-bit
|	*ou inclusif* bit-à-bit
^	*ou exclusif* bit-à-bit
~	*non unaire* bit-à-bit : complément à un, donc inverse chaque bit
>>	Décalage à droite
<<	Décalage à gauche

Opérateurs logiques

&&	*et* logique
||	*ou* logique
!	*non* logique

Les Opérateurs ++ et --

- L'opérateur ++ peut être utilisé pour pré-incrémenter ou post-incrémenter. Il incrémente toujours son opérande de 1, la différence est le moment où l'opération est effectuée.

```
int x = 4;
int y, z;

y = ++x;          /* x est incrémenté avant
                  l'instruction (une affectation) */

z = x++;          /* x est incrémenté après
                  l'instruction d'affectation */
```

- L'opérateur -- peut être utilisé pour pré-décrémenter ou post-décrémenter. Il décrémente toujours son opérande de 1, mais la différence est le moment où l'opération est effectuée.

```
int x = 4;
int y, z;

y = --x;          /* x est décrémenté avant
                   l'instruction d'affectation */

z = x--;          /* x est décrémenté après
                  l'instruction d'affectation */
```

Opérateurs *op=*

- Les opérateurs *'op='* comprennent : `&=, |=, ^=, ~=, >>=, <<=, +=, -=, *=, /=, %=`.
- Il ne peut-y avoir d'espace entre *"op"* et le signe égal `"="`.
- Les opérateurs *'op= '* sont des raccourcis pour faire un calcul avec une variable et affecter le résultat à cette même variable. La variable est le 1er argument de l'opération.
- Exemples :

```
int x = 4;
int y = 3;

x *= 7;     /* raccourci pour : x = x * 7; */

y -= 6;     /* raccourci pour : y = y - 6;*/

x /= y;     /* raccourci pour : x = x / y;*/

x <<= 2 ;   /* raccourci pour : x = x << 2 ; */
```

Révision Partielle

Cette page montre quelques exemples utilisant certains opérateurs de la page précédente et la déclaration ci-dessous (*Remarque : les exemples sont indépendants les uns des autres*).

Remplir les blancs :

```
int num = 5;/* déclaration et initialisation */
```

L'Expression :	sera évaluée à :
`num = 2`	
`num++`	
`num += 3`	
`num = num * 3`	
`num *= 3`	
`num %= 2`	
`num /= 2`	
`num = num - 7`	
`num -= 7`	

Opérateurs Supplémentaires

Opérateurs Pointeur et Adresse

*	opérateur d'indirection
&	opérateur adresse

Opérateurs Fonction et Structure

()	Appel de Fonction
[]	Référence à un élément de Tableau
.	Référence à un élément de structure
->	Référence par pointeur à un élément de structure

Opérateurs Divers

,	opérateur séquence
sizeof	taille d'un objet en octets
(*type*)	opérateur *cast* - changement de type
? :	opérateur d'expression conditionnelle

Table de Priorité et d'Associativité

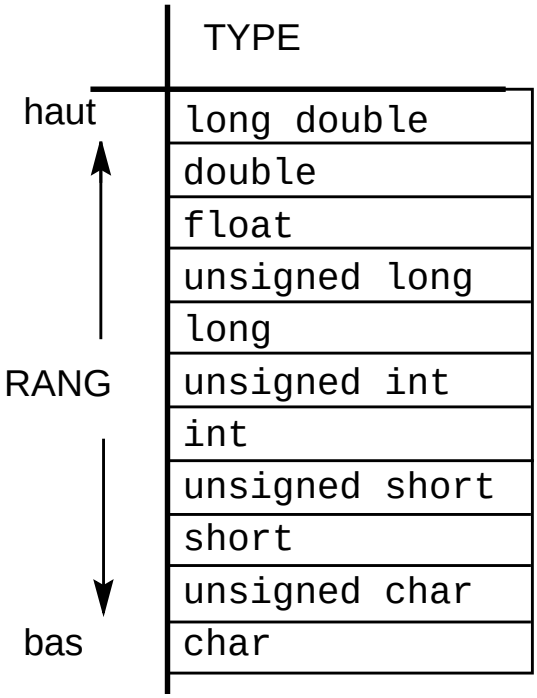
Niveau	Op	Nom	Associativité
1	()	Appel de fonction	Gauche à Droite
1	[]	élément de tableau	Gauche à Droite
1	.	membre de structure	Gauche à Droite
1	->	pointeur sur structure	Gauche à Droite
2	!	Non Logique	**Droite à Gauche**
2	~	Complément à un	**Droite à Gauche**
2	-	moins unaire	**Droite à Gauche**
2	++	Auto Incrément	**Droite à Gauche**
2	--	Auto Décrément	**Droite à Gauche**
2	&	Adresse	**Droite à Gauche**
2	*	Indirection	**Droite à Gauche**
2	(type)	Cast	**Droite à Gauche**
2	sizeof	Taille en octets	**Droite à Gauche**
3	*	Multiplication	Gauche à Droite
3	/	Division	Gauche à Droite
3	%	Modulo	Gauche à Droite
4	+	Addition	Gauche à Droite
4	-	Soustraction	Gauche à Droite
5	<<	Décalage à gauche	Gauche à Droite
5	>>	Décalage à droite	Gauche à Droite
6	<	Inférieur	Gauche à Droite
6	<=	Inférieur ou égal	Gauche à Droite
6	>	Supérieur	Gauche à Droite
6	>=	Supérieur ou égal	Gauche à Droite
7	==	Egalité	Gauche à Droite
7	!=	Différence	Gauche à Droite
8	&	ET binaire	Gauche à Droite
9	^	OU exclusif binaire	Gauche à Droite
10	|	OU inclusif binaire	Gauche à Droite
11	&&	ET logique	Gauche à Droite
12	||	OU logique	Gauche à Droite
13	?:	Conditionnelle	**Droite à Gauche**
14	= *op*=	Affectation	**Droite à Gauche**
15	,	Séquence	Gauche à Droite

Conversions de Types

- Si un opérateur a des opérandes de types différents, le type de rang inférieur sera converti dans celui de rang supérieur (promotion). Par exemple, `int` -> `float`. Dans tous les cas, les opérations sur les types `char` et `short`, en l'absence d'autres types, seront converties en `int`.

- Dans une affectation, le résultat est converti dans le type de la variable à laquelle il est affecté. (Ceci pouvant entraîner une promotion ou une dégradation.)

Hiérarchie des types:



- L'arithmétique préserve les valeurs (*value-preserving*) : les conversions arithmétiques qui impliquent une promotion vont promouvoir vers le plus petit type capable de prendre en compte toutes les valeurs.

Conversions explicites : Cast

Format

(type) <expression>

- La conversion explicite ou *cast* permet au programmeur de forcer la dégradation et / ou la promotion des types.

- Exemples de conversions explicites :

```
int main(void)
{
        float f = 3.875;
        int i, j = 100;

        i = (int) f * j;/* f est dégradé; que vaut i?
*/

        f = (float) i * j / 7;/* la promotion forcée
                va entrainer une division "réelle" */
        return 0;
} /* fin de la fonction main */
```

- Dans le programme ci-dessus, quels auraient été les résultats sans les conversions explicites ?

Révisions de Module

Déclarations

Dans les lignes suivantes, quelles sont les déclarations (légales) ?

```
int i, j=5, limit;
x int;
final_val += (limit/16);
double float;
j++;
int x = 1.0;
double f;
```

Incrément

Soient les déclarations suivantes. Quelles seront les valeurs des *variables* après évaluation des expressions ? (*Remarque : les exemples s'exécutent en séquence*)

```
int x, y;
x = 5;
y = x++;            /*     y ? ____        x ?    */

y = ++x;            /*     y ? ____        x ?    */

int a,b;
a = 10;
b = a--;            /*     a ? ____        b ?    */

b = --a;            /*     a ? ____        b ?    */
```

Révision de Module

Fonction main

Dans la fonction `main()`, déclarer un entier `index`, une variable en virgule flottante `fval`, et une variable caractère `ch`. affecter la somme de 42 et 21 à `index`, une constante en virgule flottante à `fval`, et un caractère ASCII à `ch`. Enfin, utiliser un `printf()` pour imprimer le résultat de l'expression suivante :

$$\frac{2 \times 256 - 12}{14 \ mod \ 6} \times 2$$

Révision de Module

Conversions

Soient les déclarations :

```
char c;
int x;
float y;
```

Indiquer les résultats des différentes expressions ?
***Remarque** : Les exemples s'exécutent en séquence.*

```
y = x = c = 'A';
c = c + 1;
x = y + 2 * c;
y = 2 / c + x;
```

Conversions Explicites

Soient les déclarations :

```
int n;
float x;
int result_one, result_two;
n = 10;
```

Que vont produire les expressions suivantes ?

```
x = (float)n/3;
result_one = 1.6 + 1.8;
result_two = (int)1.6 + (int)1.8;
```

Travaux Pratiques 1 : Les Fondamentaux du C

Présentation

Durant ces travaux pratiques, vous serez initiés à la programmation en C en écrivant quelques programmes C simples.

Exercices

1. **Niveau 1**. Saisir le programme élaboré en Révision de Module. Nommer le fichier `first.c`:

   % **`cc -Xc first.c`** (*L'option -Xc implique le mode ANSI)*
   % **`a.out`** (*Résultat par défaut de la compilation)*

2. **Niveau 2**. Ecrire un programme qui va afficher la taille en octets de chaque type de base. Nommer le fichier `sizes.c`:

   % **`cc -Xc sizes.c`**
   % **`a.out`**

3. **Niveau 3**. Ecrire un programme pour :

   Afficher un caractère comme un `char` puis comme un `int`.

   Affiche un `int` comme un caractère et un entier décimal.

   Conseil : prendre un entier entre 33 et 126 ;

   Nommer le fichier source `printit.c`:

   % **`cc -Xc printit.c`**
   % **`a.out`**

Opérateurs Logiques et Relationnels dans les Expressions Conditionnelles 2

Objectifs

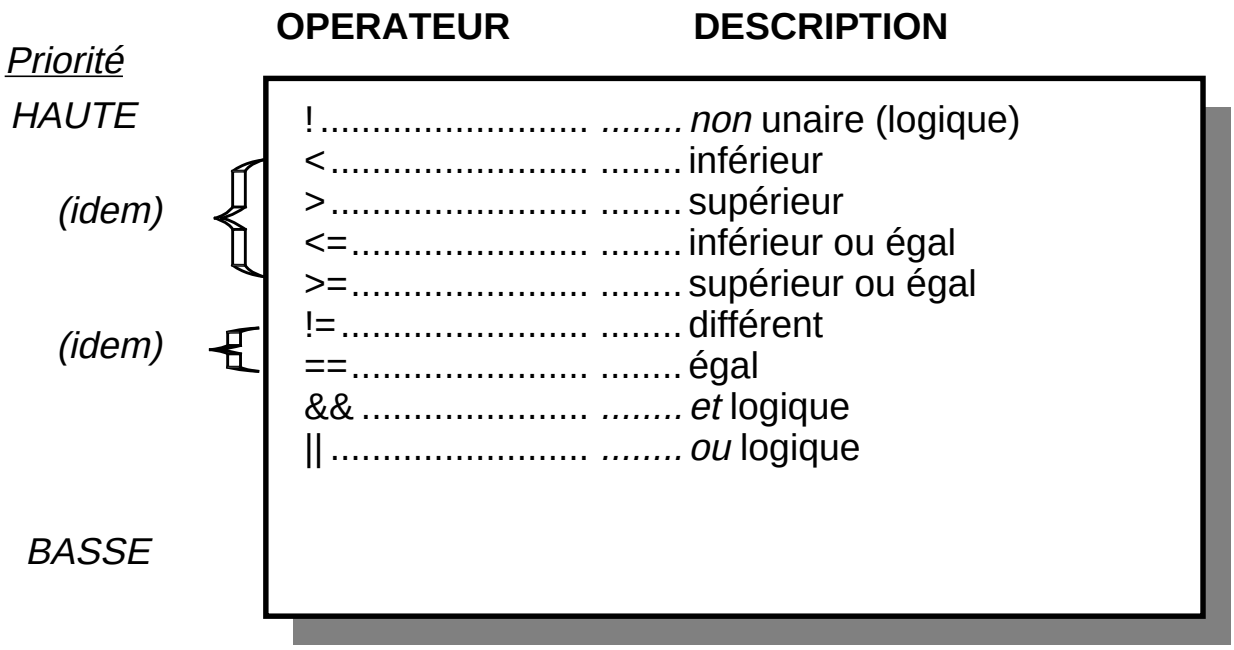
- Utiliser correctement les opérateurs logiques.
- Calculer la valeur d'une expression relationnelle.
- Décrire la différence entre les opérateurs d'égalité et d'affectation.
- Utiliser les opérateurs logiques et relationnels pour programmer des décisions.
- Utiliser les structures `if` et `switch` pour prendre des décisions.

Evaluation

Travaux Pratiques 2 et révisions de module.

Opérateurs Logiques et Relationnels

- Les opérateurs logiques et relationnels sont généralement utilisés dans des structures décisionnelles. L'ensemble de ces opérateurs est :



- Si une expression logique est fausse, sa valeur est 0. Si elle est vraie, sa valeur est 1.
- *Toute expression non-nulle est prise pour vrai.*

L'instruction `if`

L'instruction logique (ou *branchement)* la plus commune en C est l'instruction `if` :

Format 1:

```
if (expression)
      instruction;
```

Format 2:

```
if (expression)
      instruction;
else
      instruction;
```

```
int main(void)
{
    int val;

    printf("Entrez un nombre entier : ");
    scanf("%d",&val);
    if (val != 0)
       printf("La valeur est non-nulle !\n");
    else /* pas de point virgule après un else */
       printf("Vous avez tapé un zéro\n");
    if (( val > 0) && ( val < 10))
       printf("L'entier est un nombre positif à un chiffre\n");
    return 0;
}
```

L'instruction `if`

Exemples d'instructions `if` *imbriquées* :

Exemple 1

```
if (exp1) {
        if (exp2) {
            <instructions>
        }
}
else {
        if (exp3) {
            <instructions>
        }
        else    {
            <instructions>
        }
}
```

L'instruction `if`

Exemple 2

Le flux logique *voulu* est indiqué par l'indentation. Ce n'est *pas* celui exécuté. Un `else` se rapporte au premier `if` possible.

```
if (exp1)
        if (exp2)
            <instruction>
else if (exp3)
            <instruction>
else if (exp4)
        <instruction>
else
        <instruction>
```

Exécution réelle:

```
if (exp1)
        if (exp2)
            <instruction>
        else if (exp3)
            <instruction>
        else if (exp4)
            <instruction>
        else
            <instruction>
```

L'instruction `if`

Une instruction `if` peut contenir une affectation :

```
int main(void)
{
        int val, result;

        printf("Entrez une valeur entre 10 et 100 : ");
        scanf("%d", &val);
        if ((result = val * 42) >= 1024)
            printf("Résultat supérieur ou égal à 1K.\n");
        else
            printf("Résultat plus petit que 1K.\n");
        return 0;
} /* fin de la fonction main */
```

L'instruction `switch`

- L'instruction `switch` est un branchement conditionnel multiple :

```
switch (expression) {
        case constante1:
            <instructions>;
            break;
        case constante2:
            <instructions>;
            break;
        default:
            <instructions>;
            break;
}
```

```
int main(void)
{
        char ch;

        printf("Entrez a ou b:");
        scanf("%c", &ch);
        switch (ch) {
            case 'a':
                printf("Une lettre de valeur !\n");
                break;
            case 'b':
                printf("une lettre qui suit : %c.\n", (ch - 1));
                break;
            default:
                printf("Réponse incorrecte.\n");
        } /* fin du switch */
        return 0;
} /* fin de la fonction main */
```

- un `break` interrompt l'exécution du `switch`, sinon toutes les instructions qui suivent sont exécutés.
- `default` est activé si aucun `case` n'est déclenché. Il peut être placé à n'importe quel endroit dans le `switch`.

L'expression Conditionnelle *?:*

Format

expr_test ? *expr_si_vrai* : *expr_si_faux*

- Si l'*expression test* est vraie, alors l'expression après le *?* est évaluée. Sinon, c'est l'expression après le *:* qui est évaluée. La *valeur* finale de toute l'expression est soit celle de *expr_si_vrai*, soit celle de *expr_si_faux*.
- L'expression conditionnelle *?* et *:* peut remplacer une structure `if else`:

Utiliser :

```
expression_test ? expression_si_vrai : expression_si_faux
```

Au lieu de :

```
if  (expression_test)
        expression_si_vrai;
else
        expression_si_faux;
```

Révision de Module

Opérateurs

Pour les opérateurs suivants, donner le type par groupe et la fonction de chacun :

Type d'opérateur :

<
>
<=
>=

Type d'opérateur :

!=
==

Type d'opérateur :

!
&&
||

Branchements

Q. Quelle structure peut-être utilisée à la place de '*if else*' ?

R. _____

Q. Quelle est la règle d'appariement pour les *if else* imbriqués ?

R. _____

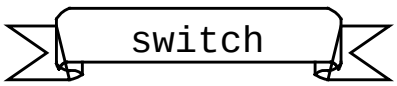
Q. Quel est le résultat de l'omission d'un *break* dans un *switch case*?

R. _____

Q. Quelle est l'utilité de *default*, et quand le code du *default* sera-t-il exécuté?

R. _____

Révision de Module



Ecrire un programme simple qui demande une appréciation (ABCDF) et répond respectivement "Excellent", "Très Bien", "Bien", "Passable" ou "Insuffisant". Donner un message en cas de saisie incorrecte.

Travaux Pratiques 2 : Expressions Conditionnelles

Présentation

Le but de cette séance de TP est de vous amener à créer des structures décisionnelles mettant en jeu différentes instructions de test.

Exercices

**** -** indique que la solution sera reprise dans une prochaine séance.

1. **Niveau 1**. Saisir, compiler et exécuter le programme de la révision du module 2.

   Nommer le fichier source `letters.c`:

   ```
   % cc -Xc letters.c
   % a.out
   ```

2. **Niveau 2**. Ecrire un programme qui va :

   Demander un nombre entre 10 et 100.

   Si le nombre saisi est en dehors de l'intervalle, afficher le message "*hors-limites*".

   Si le nombre est dans l'intervalle, vérifier sa divisibilité par 2, 3, 5, ou 7.

   Afficher un message pour chaque diviseur. Note : plusieurs peuvent être bons - comme pour 10 qui est divisible par 2 et 5.

   Afficher un message si le nombre n'est divisible par aucun des diviseurs proposés (exemple un nombre premier).

   Conseil : L'opérateur % peut être utilisé pour déterminer la divisibilité.

- voir page suivante -

Nommer le fichier source `diviz.c`:

```
% cc -Xc diviz.c
% a.out
```

3. ****Niveau 3**. Ecrire un programme qui va :

Demander une année de naissance.

Vérifier que l'année est plausible (en l'occurence entre 1900 et 1975), pour éviter une faute de frappe. En cas de saisie erronée, sortir avec un message d'erreur.

Dans le cas d'une date correcte, soit *1945*, en déduire la valeur de chaque chiffre séparément : *1*, *9*, *4* et *5*.

Afficher un menu à 3 options :

1) SOMME,
2) PRODUIT,
3) Age actuel,

Faire le calcul indiqué en traitant le choix à l'aide d'un `switch`.

Implémenter un cas `default` pour un message d'erreur en cas de choix incorrect.

Nommer le fichier source `ages.c`:

```
% cc -Xc ages.c
% a.out
```

Conseil :

```
int main(void)/* extraction des chiffres */
{
     int x = 1947;
     int unites, dizaines, centaines, milliers;
     unites = x % 10;
     x = x / 10;
     dizaines = x % 10;
     x = x / 10;
     centaines = x % 10;
     x = x / 10;
     milliers = x % 10:
     printf("%d\n%d\n%d\n%d\n", milliers, centaines,
          dizaines,unites);
}
```

Fonctions et `<stdio.h>`

Objectifs

- Ecrire des programmes simples avec plusieurs fonctions.
- Utiliser le prototypage des fonctions pour les déclarations et définitions.
- Décrire l'interfaçage des fonctions.
- Identifier le mécanisme de transfert de contrôle au programme appelant.
- Utiliser correctement la fonction *`printf()`* pour afficher entiers, caractères et nombres en virgule flottante.
- Réaliser des conversions à la saisie avec la fonction *`scanf()`*.
- Lire sur l'entrée standard caractère par caractère avec *`getchar()`*.
- Ecrire caractère par caractère sur la sortie standard en utilisant *`putchar()`*.

Evaluation

Travaux Pratiques 3 et révision de Module.

Fonctions

- Les fonctions sont utilisées pour réaliser une petite partie d'un travail.
- La fonction *main()* a été utilisée dans tous les exemples jusqu'à maintenant. Elle est obligatoire dans tout programme C.
- Les définitions de fonctions ne peuvent être imbriquées entre elles.

Format

Syntaxe ANSI C pour la définition de fonction (sans correspondance en C traditionnel) :

```
<type> <id_de_fonction> (<liste-de-types_param>)
{
        <déclarations>;
        <instructions>;
}
```

Exemple de Définition de Fonction

```
int main(void)
{
        double dnum, rad = 5.67;
        double circum(double);/* décla.*/

        dnum = circum(rad);

        /* suite du programme */
        return 0;
}
double circum(double r) /* définition */
{
        double pi = 3.14159;
        return ( 2 * pi * r);
}
```

Interface de Fonction

- Une fonction dont le type retourné est différent de `int` doit être déclarée au niveau de la fonction appelante. Déclarer également les fonctions qui retournent un `int` est une bonne habitude.

Format

<type> *<idfonct>*(*<liste des types des param.>*);

- Dans l'appelant, l'appel de la fonction lui-même est une expression. Les arguments passés doivent avoir le même type que ceux définis pour la fonction.

- Le contrôle revient à l'*appelant* lorsque l'on rencontre soit le "}" de fin dans l'*appelée*, soit une instruction `return`.

```
int main(void)
{
        int cube(int);  /*déclaration de la fonction cube*/
        int result;
        int val;

        printf("Entrer un entier : ");
        scanf("%d", &val);
        result = cube(val);
        printf("Le cube de %d est %d.\n", val, result);
        return 0;
} /* fin de la fonction main */

int cube(int n) /* définition de la fonction cube */
{
        return(n * n * n);
} /* fin de la fonction cube */
```

Interface de Fonction - `return`

- Une déclaration de fonction précédée du mot-clef `void` informe le compilateur qu'aucune valeur n'est retournée. Si la fonction retourne un type autre que `int`, celui-ci doit être indiqué. Mais il est bon de déclarer également les fonctions retournant un `int`.
- L'instruction `return` ne peut passer qu'*une seule valeur* à l'appelant.
- L'expression évaluée dans l'instruction `return` devient la valeur de l'expression appel de fonction dans l'appelant.

```
int main(void)
{
        int cube(int);
        int result;
        int val;

        printf("Entrer un entier: ");
        scanf("%d", &val);
        result = cube(val);
        printf("Le cube de %d est %d.\n", val, result);
        return 0;
} /* fin de la fonction main */

int cube(int n) /* définition de la fonction cube */
{
        return(n * n * n);
} /* fin de la fonction cube */
```

Fin de Programme - *exit()*

Format

```
void exit(int status);
```

- Par convention, *status* vaut 0 pour les retours sans erreur et 1 si une erreur est survenue.
- *exit()* peut être utilisée pour sortir d'un programme avant la rencontre de l'accolade de fin "}".
- *exit()* est une sortie propre d'un programme. On ne revient pas d'un appel à la fonction *exit()*.

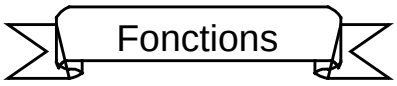
```
int main(void)
{
        void error(void);/*déclaration de la fonction error*/
        int num;

        printf("Entrer un entier entre -25 et 25:  ");
        scanf("%d", &num);
        if ((num < -25) || (num > 25))
            error( );
        else
            printf("Le nombre entré est %d\n", num);
        return 0;
} /* fin de main */

void error(void)/*définition de la fonction error*/
{
        printf("Entier hors intervalle, fin.\n");
        exit(1);

 } /* fin de la fonction error */
```

Arguments

- Les arguments sont *passés par valeur* à l'appel des fonctions.
- La valeur de chaque argument à l'appel est affectée au paramètre correspondant dans la fonction :

```
int main(void)
{
        int num = 5;
        void func(int);

        func(num);
        printf("Main: num = %d.\n", num);
        return 0;
} /* fin de la fonction main */

void  func(int number)
{
        number += 2;
        printf("Func: number = %d.\n", number);
} /* fin de la fonction func */
```

```
% a.out
Func: number = 7.
Main: num = 5.
%
```



Ecrire un programme comportant une fonction *`addum()`*, prenant 2 arguments `int`, et retournant un `int` égal à leur somme :

Inclusion de fichiers

- La directive *#include* est utilisée pour inclure le contenu d'autres fichiers dans un source. Par exemple,

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

- Les signes < et > indiquent que le fichier à inclure doit être recherché dans une liste de répertoires dépendant de l'implémentation du compilateur. `cc` recherchera d'abord dans (cas de la version 2.0 du Sun C) */installation_dir*`/SUNWste/SC2.0/include/cc` pour les fichiers header. En cas d'échec la recherche se poursuivra dans `/usr/include`.

- */installation_dir*`/SUNWste/SC2.0/include/cc` contient les fichiers suivants :

floatingpoint.h	stab.h	sys
math.h	sunmath.h	

- Par convention, les guillemets désignent le chemin de fichiers *non* trouvés dans les répertoires standards, comme pour *"mydefs.h"*, et *"myincludes/defs.h"*. Dans ce cas, le chemin indiqué est prioritaire sur les répertoires standards.

<stdio.h> et <stddef.h>

- <stdio.h> est le fichier /usr/include/stdio.h.

  Il définit les constantes et les fonctions utilisées fréquemment dans les entrées/sorties. Il est appelé le fichier header des entrées/sorties standard.

- <stdio.h> définit *EOF* :

  La valeur de cette constante est habituellement définie de telle sorte qu'elle ne corresponde à aucun *caractère existant*. Elle est retournée pour une fin de fichier (*End Of File*).

- <stddef.h> définit *NULL* :

  Cette constante est décrite comme un pointeur de type void égal à 0. Beaucoup de fonctions retournent cette valeur en cas d'erreur.

Fonction d'écriture de base - *printf()*

Format

```
#include <stdio.h>
int printf(const char *format, ...);
```

- Le code retourné par *printf()* est un entier. Normalement égal au nombre de caractères affichés (transmis), il est négatif en cas d'erreur. (La plupart des programmeurs ignorent le code retourné par *printf()*.)
- *format* indique le formatage des sorties :

1. chaîne de caractères
2. spécification de format

- Les autres arguments (en nombre variable) représentent les éléments à formater et à imprimer.

Spécifications de format

- Les spécifications de format indiquent à *printf()* (et *scanf()*) comment écrire (lire) la valeur d'une expression. Elles comprennent :

%c	Conversion des `int` en `unsigned char`
%d	Valeur entière décimale
%i	Valeur entière décimale (l'argument correspondant détermine la base)
%o	Valeur octale non signée
%s	Chaîne de caractères (Tableau de `char`)
%u	Valeur entière décimale non signée
%x	Valeur héxadécimale non signée
%e, *f* ou *g*	Valeur en virgule flottante
*%***m.n***f*	**m** est la taille du champ et **n** est le nombre de chiffres après la virgule ('.' içi)
*%***-m.n***f*	le signe moins force la justification à gauche dans le champ
*%h*i	short int (normalement avec *scanf()*)
%li	long int (normalement avec *scanf()*)
%lf	double (normalement avec *scanf()*)
%Lf	long double

- L'indication de la taille du champ (**m**) et de la justification à gauche (**-**) peut compléter toute spécification de format.

Utilisation de *printf()*

Exemples de *printf()* montrant des spécifications de format :

```
#include <stdio.h>
int main(void)
{
        int x = 10;
        char c = 'q';
        float f = 1.23;

        printf("Simplement une chaine de caractères.\n");
        printf("hexa:%x,octal: %o, flottant: %f\n",x,x,f);
        printf("caractère c = %c\n",c);
        return 0;
} /* fin de la fonction main */
```

Conversion en entrée avec format - *scanf()*

Format

```
#include <stdio.h>
int scanf(const char *format, ...);
```

- Normalement, scanf() retourne le nombre d'items correctement saisis. En cas d'erreur de lecture avant toute conversion, *scanf()* retourne *EOF*.
- Utilisation de *scanf()*pour lire sur l'entrée standard :

```
#include <stdio.h>
int main(void)
{
    int ival, num;
    float fval;
    double dval;
    printf("Entrer un entier et deux réels:  ");
    num = scanf("%d%f%lf",&ival, &fval, &dval);
    if (num < 3) {
     printf("Erreur dans scanf()\n");
     exit(1);
    }
    else {
     printf("Vous avez saisi %d items, convertis en :\n",num);
     printf("L'entier: %d,les réels: %f %f.\n", ival, fval, dval);
    }
    return 0;
} /* fin de la fonction main */
```

Saisie de caractères - *getchar()*

Format

```
#include <stdio.h>
int getchar(void);
```

- En cas de succès, *getchar()* renvoie le caractère suivant du flux *entrée standard*. Sur une fin de fichier ou une erreur de lecture, *getchar()* renvoie *EOF*.
- Exemple de lecture d'un caractère utilisant *getchar()*:

```
#include <stdio.h>
int main(void)
{
    int ch; /* ATTENTION : type int pour getchar() */

    printf("Entrer un caractère :  ");
    ch = getchar( );
    printf("Vous avez saisi %c, exact ?\n", ch);
    printf("Code ASCII %d décimal, %x hexa, %o octal.\n",
            ch,ch,ch);
    return 0;
} /* fin de la fonction main */
```

Sortie d'un caractère - *putchar()*

Format

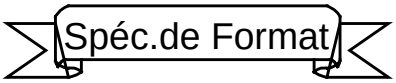
```
#include <stdio.h>
int putchar(int c);
```

- En cas de réussite, *putchar()* renvoie le caractère transmis à *sortie standard*. Sur erreur, *putchar()* renvoie *EOF*.

- Le programme suivant répète ce qui est saisi en utilisant *getchar()* et *putchar()*:

```
#include <stdio.h>
int main(void)
{
        int ch;

        printf("Saisir un caractère :  ");

        ch = getchar( );
        printf("caractère saisi : ");
        putchar(ch);/* écrit le caractère */
        return 0;
} /* fin de la fonction main */
```

Révision de Module



Soient les déclarations suivantes, écrire un ou plusieurs *printf()* pour imprimer les valeurs et indiquer le résultat attendu :

```
int num1 = 16 * 2;
int num2 = 0xFF;
int num3 = 0777;
float fnum = 42.0 + num1;
char ch1 = 'a';
char ch2 = ch1 + 1;
```

Affichage :

_____

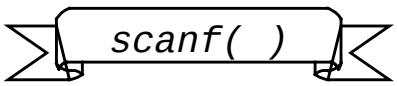
_____

_____

Après les déclarations suivantes, écrire deux instructions, utilisant des fonctions différentes, pour lire un caractère sur l' *entrée standard* :

```
char chr;
int  ch;
```

Révision de Module



Soient les *scanf()* et *printf()* et le jeu d'essai suivant. Quel est le résultat attendu ?

```
int result, x;
float f;
x = 0;
f = 0.0;
printf("Entrer un entier puis un réel #:  ");
result = scanf("%d%f",&x,&f);
printf("nb items lus : %d, x = %d, f = %f.\n", result,x,f);
```

saisie :
```
        25    54.32E-1
```
impression :

_____

saisie :
```
        127   2+5
```
impression :

_____

saisie :
```
        12.5E2   17
```
impression :

_____

saisie :
```
        a   12.5
```
impression :

_____

Travaux Pratiques 3 : Fonctions et `<stdio.h>`

Présentation

Ces Travaux Pratiques permettent la prise en main des fonctions et des appels de fonction, ainsi que la mise en œuvre de quelques fonctions de la librairie standard d'Entrée/Sortie.

Exercices

1. **Niveau 1**. *Vérifier* les résultats des questions de la révision en fin de chapitre, en écrivant un programme faisant les mêmes déclarations, les mêmes traitements d'E/S, et imprimant les résultats. Nommer le programme source `review_io.c`:

   ```
   % cc -Xc review_io.c
   % a.out
   ```

2. **Niveau 2**. Ecrire un programme pour :

   Demander un caractère *alphabétique*.

   Tester la saisie (Est-ce bien un caractère alphabétique ?).

   Imprimer le caractère ou afficher le message "ERREUR" .

   Demander un entier.

   Afficher cet entier en base 10, en base 8 et en base 16.

   Demander un réel (virgule flottante).

   Imprimer le produit de l'entier et du réel avec une précision de 5 chiffres après la virgule.

   Imprimer ce produit en notation exponentielle scientifique - *ie.* 4.2000000E+1.

   Afficher le message "*Où tu iras, je serai.* \ *n*".

- voir page suivante -

Nommer le fichier source `mixed_io.c`:

```
% cc -Xc mixed_io.c
% a.out
```

3. **Niveau 3.** Ecrire un programme pour :

   Demander le prix de quelque chose (lire un `double`).

   Appeller une fonction *`tva()`* pour calculer et retourner une TVA à 18,6%.

   Imprimer le hors taxe, la TVA et le montant TTC.

   Conseil : Exemples de déclaration et définitions :

```
     /* définition de fonction... */
double tva(double valeur, double taux)
/* valeur est le montant hors-taxe et taux le
     taux de TVA */
```

   Nommer le fichier source `tva.c`:

```
% cc -Xc tva.c
% a.out
```

Introduction au Compilateur C et au Préprocesseur

4

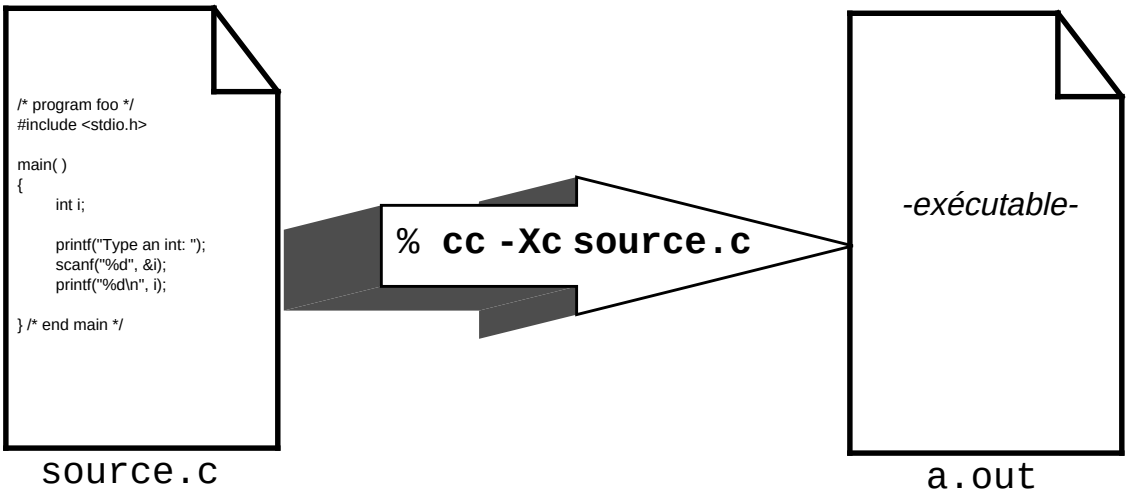
Objectif

- Utiliser les options de compilation pour obtenir différents niveaux de conformance ANSI.
- Générer des fichiers *.i*, *.s*, et *.o* en utilisant les options de `cc` adéquates.
- Utiliser les options de spécification de chemin de recherche pour les fichiers header et les librairies.
- Utiliser les directives du préprocesseur pour définir des constantes et des fichiers d'include.
- Utiliser `lint` pour vérifier des sources C.

Evaluation

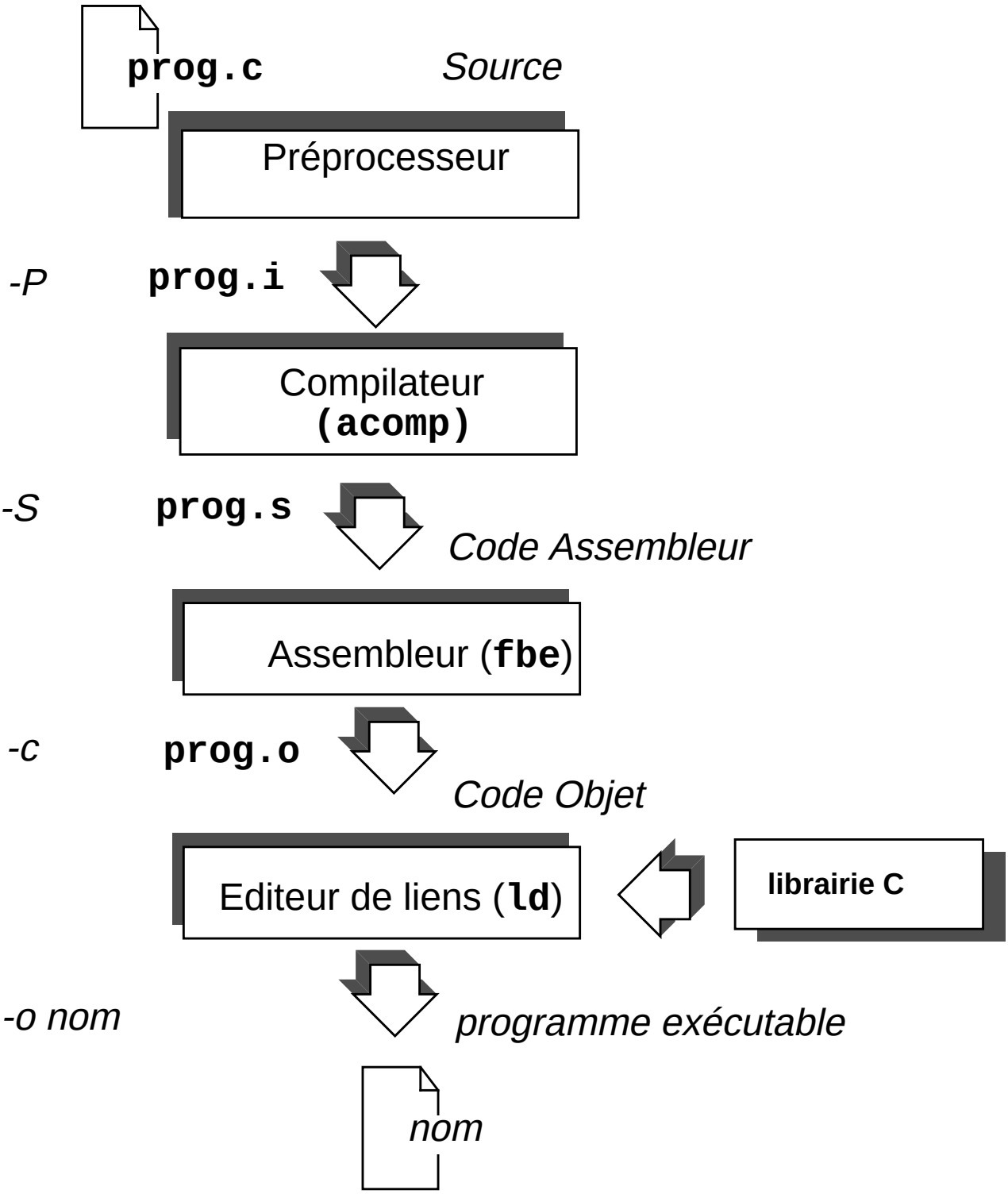
Travaux Pratiques 4 et révision de module.

Compilation de Programmes C



- Les fichiers sources C doivent avoir l'extension `.c`
- La commande `cc` appelle le compilateur C ANSI `acomp` (qui contient le préprocesseur et le compilateur proprement dit), l'assembleur `fbe`, et le linker `ld` pour créer le fichier executable, `a.out`.
- Le résultat de `cc` est `a.out` par defaut, l'option *`-o`* permettant d'indiquer le nom voulu.

Phases de Compilation



Options de Conformance ANSI C

- Le degré de conformance au standard ANSI du Langage C peut être indiqué par les options de compilation suivantes:

-Xt (transition) Cette option donne une compatibilité ANSI C plus K&R C, *sans* les changements sémantiques imposés par le C ANSI. Ceci est l'option par défaut.

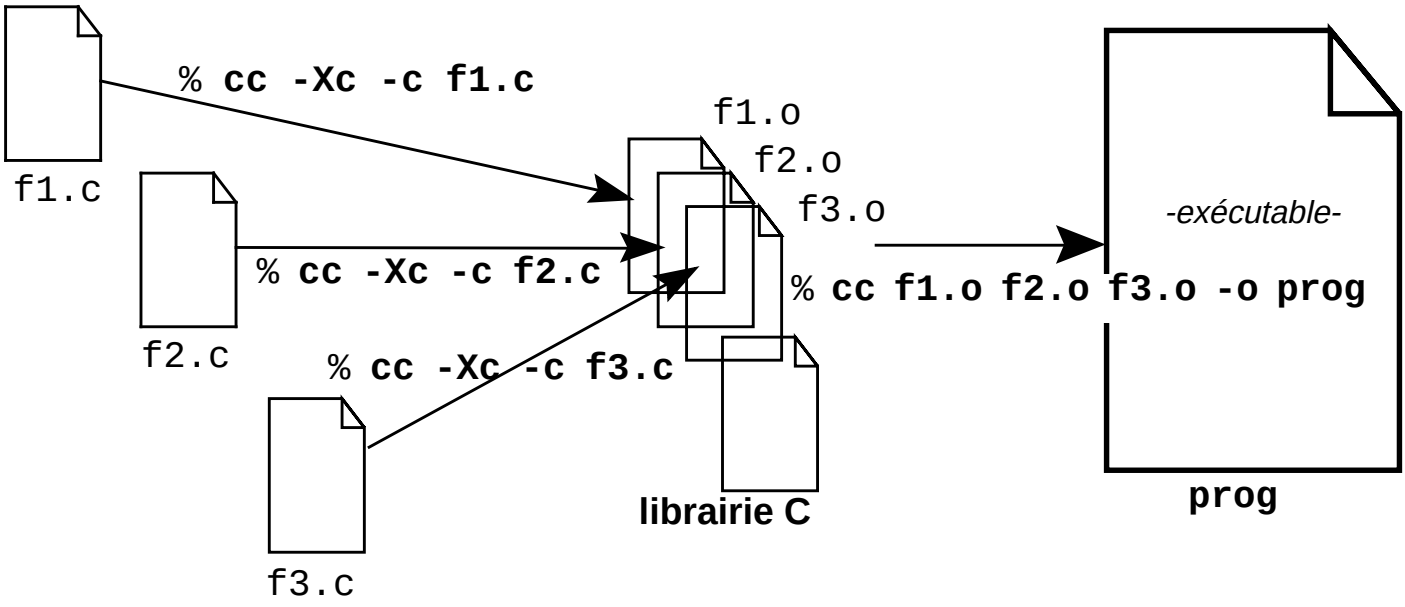
-Xa (ANSI) Cette option donne une compatibilité ANSI C plus K&R C, *avec* les changements sémantiques imposés par le C ANSI.

-Xc (conformance) Avec cette options, les sources et les header se conforment au C ANSI, sans aucune extension K&R.

-Xs (senescent)(*"devenant vieux"*, option Sun C) Le langage compilé inclut les possibilités pré-ANSI K&R. Le compilateur signale les constructions ayant un comportement différent entre le C ANSI et le C K&R.

- La macro prédéfinie `__STDC__` prend la valeur 1 avec l'option `-Xc` ou la valeur 0 autrement. Le standard ANSI du C définit `__STDC__` comme étant à 1.

Compilation Séparée



- Si vous avez séparé vos fonctions en les écrivant dans des fichiers différents, vous pourrez les compiler individuellement.
- Pour compiler sans linker, utiliser l'option *-c*. Le fichier généré aura la même racine que le source mais avec l'extension *.o*
- Utiliser cc pour linker les fichiers objets (et la librairie C) et créer ainsi l'exécutable final.

Compilation avec des Librairies d'Application

- Quand le programme appelle des fonctions d'une librairie autre que la librairie C, le programmeur doit linker avec la librairie contenant les fonctions.
- Voici les options de la commande *cc* requises généralement pour le link avec les librairies d'application :

Options et Syntaxe de `cc`	Signification
`-l<librairie>`	Link avec la librairie indiquée.
`-I<repertoire>`	Répertoire de recherche pour les fichiers `#include` dont le nom ne commence pas par /, avant de chercher dans les répertoires standards.
`-L<repertoire>`	Répertoire de recherche pour les librairies, avant de poursuivre dans les répertoires standards.

- L'option `-l<librairie>` doit suivre l'argument fichier source.
- La recherche des fichiers `#include` est la suivante :
  ① le répertoire du source (si le nom du fichier est entre " "),
  ② les repertoires indiqués par `-I<repertoire>`,
  ③ `/install_dir/SUNWste/SC2.0/include`, (si version 2.0)
  ④ `/usr/include`.

Utilisation des Librairies d'Application

Ce programme a été compilé sur une SPARCStation 1+. Le compilateur C ANSI était installé dans /usr/opt.

trig.c

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

int main(void)
{
     double val = -1.0;
     printf("l'arc sinus de %f est %f.\n", val, asin(val));
     return 0;
}
```

```
sun%  cc -Xc trig.c -o trig
Undefined                       first referenced
   symbol                          in file
asin                               trig.o
ld: fatal: Symbol referencing errors. No output written to trig
sun% cc -Xc -# trig.c -o trig -lm
/usr/opt/SUNWste/bin/../SC2.0/acomp -i trig.c -o
/var/tmp/ctm1BAAa000BU -Qy -Xc
-I/usr/opt/SUNWste/bin/../SC2.0/include/cc
/usr/opt/SUNWste/bin/../SC2.0/fbe -o trig.o -s -q -Qy
/var/tmp/ctm1BAAa000BU
/usr/ccs/bin/ld -dy /usr/opt/SUNWste/bin/../SC2.0/crti.o
/usr/opt/SUNWste/bin/../SC2.0/crt1.o
/usr/opt/SUNWste/bin/../SC2.0/__fstd.o  /usr/ccs/lib/values-Xc.o  -o  trig
trig.o -lm -Y
P,/usr/opt/SUNWste/bin/../SC2.0:/usr/ccs/lib:/usr/lib -Qy -lc
/usr/opt/SUNWste/bin/../SC2.0/crtn.o
sun% trig
L'arc sinus de -1.000000 est -1.570796.
sun%
```

Directive du Préprocesseur : Définition de Constantes

- Une possibilité de la directive `#define` est de créer des *constantes symboliques.*
- La valeur est substituée chaque fois que l'identifiant apparaît dans le source :

my_header.h

```
#define BUF_SIZE 512
#define MESSAGE "Programme avec constantes symboliques."
```

prog.c

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

int main(void)
{
        printf("Taille de tampon : %d.\n", BUF_SIZE);
        printf("%s\n",MESSAGE);
        printf("MESSAGE\n");  /* affiche:  MESSAGE
                et non la valeur de MESSAGE */
        return 0;
} /* fin de la fonction main */
```

Vérification de Programmes C - `lint`

- `lint` traite les fichiers sources.

- `lint` réagit principalement dans 3 catégories de cas :

1. Usage inconsistant
2. Code non portable
3. Structures suspectes

- `lint` travaille en deux passes :

  Première passe : Vérification des erreurs possibles internes au source.

  Seconde passe : Vérification de l'intégrité entre plusieurs sources.

- `lint -Xc` demande à `lint` de vérifier la conformance au standard ANSI.

- `lint -p` pousse `lint` à vérifier plus précisément la portabilité.

Vérification de Programmes C - `lint`

- `lint` signalera les inconsistances telles que :

1. Incohérence entre le type et/ou le nombre des arguments passés aux fonctions. Le prototypage des interfaces de fonctions l'aide dans cette tâche.
2. Mauvaise utilisation de pointeurs (voir plus loin).
3. Variables et fonctions définies mais non-utilisées.
4. Ignorance des codes de retour des fonctions.

- Exemple de programme et rapport de `lint` :

```
#include <stdio.h>
main(void)
{
        int x;
        char c;
        float f;
        x = 16;
        c = 'R';
        f = 1.23;
        printf("x = %d, c = %c, f = %f\n",x,c,f);
        printf("octal x = %o, hexa x = %x\n",x,x);
}
```

```
sun% lint -Xc types.c
(7) error: syntax error before or at: float
(10) error: undefined symbol: f
(11) error: newline in string literal
(12) error: syntax error before or at: printf

set but not used in function
    (10) f in main
    (6) c in main
    (5) x in main

implicitly declared to return int
    (11) printf

declaration unused in block
    (11) printf
lint: errors in types.c; no output created
lint: pass2 not run - errors in types.c
```

Vérification de Programmes C - `lint`

- Le programme suivant se compile mais `lint` se plaint encore :

```
#include <stdio.h>
int main(void)
{
        int x;
        char c;
        float f;
        x = 16;
        c = 'R';
        f = 1.23;
        printf("x = %d, c = %c, f = %f\n", x, c, f);
        printf("octal x = %o, hexa x = %x\n", x, x);
        return 0;
}
```

```
sun% lint -Xc types.c
function returns value which is always ignored
    printf

        <output from compatibility check with llib-lc.ln>
sun%
```

- Pour éviter d'avoir un message de `lint` pour les fonctions dont le code de retour est ignoré, forcer leur type à `void`, si ce code est *vraiment* inutilisé.

Révision de Module

compilation

Pour chacune des options de `cc` ci-dessous, indiquer le type et l'extension du fichier généré:

	type du fichier généré	suffixe
`% cc -Xc -P mumble.c`	_____	_____
`% cc -Xc -S mumble.c`	_____	_____
`% cc -Xc -c mumble.c`	_____	_____
`% cc -Xc -o enunciate mumble.c`	_____	_____

compilation séparée

Soient les 4 fichiers suivants (tous issus de sources C), quelle sera la commande pour les linker tous en un exécutable nommé `concasseur`: `sel.c`, `poivre.o`, `cumin.c`, `thym.s`:

_____

préprocesseur

Ecrire une directive pour créer une constante symbolique valant 42, une deuxième directive pour une constante égale à la précédente plus un entier, et une dernière pour une constante représentant la chaîne de caractère "*La Légèreté s'oppose à la Gravité*":

_____
_____
_____

Travaux Pratiques 4 : cc, Préprocesseur et lint

Présentation

Ceci est une introduction à l'utilisation de quelques options de compilation, à la programmation des directives du préprocesseur, et à la vérification de programmes à l'aide de `lint`.

Exercices

1. **Niveau 1**. Recompiler les programmes des TP précédents en utilisant les options de compilations suivantes :

   -P, -S, -c, -o.

   Noter le type et le nom des fichiers générés. Utiliser la commande `file` :

   ```
   % file <fichier-généré>
   ```

   pour déterminer le type de fichier. Si le contenu est du texte ascii, le visualiser :

   ```
   % cc -Xc -P mixed_io.c
   % more mixed_io.i
   ```

2. **Niveau 2**. Faire tourner `lint` sur tous les sources créés jusqu'à présent, rediriger les sorties vers un fichier de rapport de `lint` :

   ```
   % lint -Xc review_io.c > lint.log
   % more lint.log
   ```

Structures Itératives

Objectifs

- Utiliser correctement les structures itératives du langage C.
- Repérer les similitudes entre `while` et `for`.
- Décider quand et comment placer des `goto` en C.

Evaluation

Travaux Pratiques 5 et révision de module.

L'instruction `for`

Format

`for (`*`<expression1>`*`;`*`<expression2>`*`;`*`<expression3>`*`)`
`<instruction>` `;`

- *`expression1`* est appelée aussi *initialisation*.
- *`expression2`* est le "test". Si le test est omis, une boucle infinie est déclenchée car il est pris comme *vrai*.
- *`expression3`* est le *pas*.
- Toutes les trois sont optionnelles.
- La *virgule* est utilisée dans l'une ou l'autre des expressions pour séparer plusieurs instructions.
- Les points-virgules sont syntaxiquement indispensables :

```
 #include <stdio.h>

int main(void)
{
     int index;
     int y;

     for (index=0; index <= 19; index++)
        printf("%d\n",index); /* boucle à une seule instruction */
     printf("Fin de boucle.\n");

     for (index = 5, y = 1;(index > 0) && (y < 10);index--, y += 3) {
        printf("index = %d\n", index);
        printf("y = %d\n", y);
     } /* boucle sur un bloc */
     return 0;
} /* fin de la fonction main */
```

L'instruction `while`

Format

while (*<expression>*)
    <instruction> ;

- Si l' *expression* est vraie, *instruction* est exécutée. La boucle se termine quand et si l'*expression* devient fausse.

- Si l'*expression* est fausse avant le premier passage dans la boucle, l'*instruction* n'est pas exécutée du tout.

```
#include <stdio.h>
int main(void)
{
        int index = 0;

        while (index <= 10)
            printf("%d\n",index++);
        printf("Fin de boucle.\n");
        return 0;
} /* fin de la fonction main */
```

`for` **contre** `while`

- En général, une boucle `for` est équivalente à une boucle `while`.
- L'instruction `for` aide à garder le contrôle de la boucle à un seul endroit.
- L'instruction `for` est habituellement utilisée lorsque les valeurs initales, la condition de boucle, et la condition de fin sont contrôlées par la même variable. Elle est utilisée aussi lorsqu'on connaît le nombre d'itérations à l'avance.

```
#include <stdio.h>
int main(void)
{
        int index;

        index = 0;
        while (index < 100) {
            /* corps de boucle */
            index++;
        } /* fin du while */

        /* for équivalent... */

        for (index = 0; index < 100; index++) {
            /* corps de boucle */
        } /* fin du for */
        return 0;
} /* fin de la fonction main */
```

L'instruction do while

Format

```
do
  <instruction> ;
while ( <expression> );
```

- Dans cette structure, l'expression est évaluée à la *fin* de la boucle.

- L'instruction est toujours exécutée au moins une fois.

```
#include <stdio.h>

int main(void)
{
     int val;

     do {
        printf("Une valeur entre 1 et 10: ");
        scanf("%d", &val);
     } while (val < 1 || val > 10);

     printf("fin de boucle.\n");
     return 0;
} /* fin de la fonction main */
```

Contrôle de boucle - `break`

`break` force la sortie de la boucle où il apparaît:

```
#include <stdio.h>
int main(void)
{
        int val;

        for (;;) { /* boucle infinie */
            printf("Une valeur entre 1 et 10 : ");
            scanf("%d", &val);
            if (val >= 1 && val <= 10)
                break;  /* Sortie de boucle sur saisie correcte */
        }   /* fin du for */

        /* suite du programme */

        return 0;
} /* fin de la fonction main */
```

Contrôle de Boucle - `continue`

- `continue` saute à l'itération suivante de la boucle.
- Dans les `do` et les `while`, ceci correspond à l'évaluation de la condition de boucle.
- Pour les `for`, ceci correspond à l'évaluation de la portion *pas* du `for`:

```
#include <stdio.h>
int main(void)
{
        int num;

        while (1) { /* boucle infinie */
            printf("Une valeur entre 1 et 10 : ");
            scanf("%d", &num);
            if (num < 1 || num > 10) {
                printf("Hors intervalle!\n");
                continue;
            }
            printf("Vous savez lire !\n");
            break;
        }
        return 0;
} /* fin de la fonction main */
```

Contrôle de Boucles Imbriquées

Exemples de contrôles de boucles *imbriquées* :

```
#include <stdio.h>
int main(void)
{
        int val, i;

        for (i = 0; i < 10; i++) {
            while (1) {
                printf("Entrez un entier : ");
                scanf("%d", &val);
                if (val % 2) {
                    printf("Nombre impair.\n");
                    continue;
                }
                printf("Nombre pair.\n");
                break;
            } /* Fin du while */
        }  /* fin du for */
        return 0;
} /* fin de main */
```

Traverser des boucles imbriquées - goto

Format

goto *etiquette*;
etiquette: *instruction*;

- Les structures itératives peuvent être aussi implémentées en utilisant l'instruction de saut goto:

```
#include <stdio.h>
int main(void)
{
        int a, b;

        for (a = 0; a < 5; a++){
            for (b = 0; b < 5; b++) {
                printf("a=%d;  b=%d\n", a, b);
                if ((a == 3) && (b == 3))
                    goto fin_de_boucle;
            }
        }

fin_de_boucle:
        printf("C'est tout pour aujourd'hui\n");

        return 0;
} /* fin de la fonction main */
```

- Prenez garde à l'effet *Spaghetti*, très fréquent avec les goto.

Révision de Module

boucles `while`

Q. Quelle est la principale différence entre une boucle `while` et une boucle `do-while`?

R. _____

Q. Quelles sont les conditions de sortie d'une boucle `while` ?

R. _____

Q. Quelles règles peuvent aider à déterminer si une boucle `for` est plus adéquate qu'une boucle `while` ?

R. _____

Ecrire une boucle `while` qui s'exécute 10 fois, et à chaque itération imprime la valeur de l'index et le résultat de la somme de l'index avec une constante. Prendre les instructions de boucle ainsi que l'incrément dans un bloc :

boucles `for`

Q. Qu'est-ce que provoque l'omission de la deuxième expression dans une instruction `for` ?

R. _____

Q. Combien de fois une boucle `for` est-elle exécutée ?

Réécrire la boucle `while` précédente avec un `for` :

Travaux Pratiques 5 : Itération

Présentation

Introduction aux structures itératives du Langage C.

Exercices

****** La solution sera utilisée dans les TP à venir.

1. **Niveau 1**. *Vérifier* Les résultats des deux programmes des révisions en les compilant et en les exécutant.

2. **Niveau 2**. Ecrire un programme pour :

   Afficher à l'écran un triangle rectangle fait d'astérisques "*" , et avec une base de 40 caractères. *Exemple d'affichage* :
   *
   **
   ***
   ****
   ***** (etc…)

   Facultatif : Laisser l'utilisateur définir le caractère à afficher et la taille de la base. Si la base est en dehors de l'intervalle [1-80], la forcer à 40.

   Nommer le fichier source `triangle.c`:
   `% cc -Xc triangle.c -o triangle`

3. ****Niveau 3**. Ecrire un programme pour :

   Demander une minuscule à l'utilisateur, tant qu'il n'en rentre pas une.

Remarque : Quand l'utilisateur entre un caractère, *`scanf()`* et *getchar()* laissent un caractère *newline* dans le flux d'entrée standard. Ce '\n' doit être pris en compte d'une manière ou d'une autre.

- voir page suivante -

Déterminer si la majeure partie de l'alphabet minuscule est avant ou après ce *char*. *'n'* est considéré comme le milieu.

si le *char* est au début de l'alphabet, alors afficher les caractères jusqu'à la fin en ordre croissant, *char* compris. Si le *char* est dans la deuxième moitié, afficher l'alphabet à l'envers, à partir du *char* jusqu'au début.

Nommer le fichier source `loops.c`:

```
% cc -Xc loops.c -o loops
```

Tableaux

Objectifs

- Déclarer et utiliser correctement les tableaux.
- Traiter les tableaux avec des structures itératives.
- Manipuler des tableaux à plusieurs dimensions.
- Initialiser tous les types de tableaux.

Evaluation

Travaux Pratiques 6 et révision de module.

Déclarer un Tableau

Format

<type> *<identifiant>*[<nb d'éléments>];

- Un tableau est un ensemble ordonné de données de même type, référencé par un nom unique.
- Un tableau se définit plutôt comme une variable scalaire plus une spécification de taille de tableau.
- Les constantes symboliques sont très utiles lorsqu'on déclare et manipule les tableaux :

```
#define MAX_INDEX 10
int main(void)
{
        int int_array[25];/* à éviter */
        float float_array[MAX_INDEX];/* ok */
        char char_array[MAX_INDEX * 4];

        /* ici le code du programme */
        return 0;
}
```

Référencer les Eléments de Tableaux

Format

<identifiant>[indice]

- L'accès à un élément de tableau se fait en indiquant le nom du tableau suivi de l'*indice* entre crochets.
- La plage des indices valides d'un tableau est de *zéro* à (*nb éléments - 1*) :

```
#define MAX 10
int main(void)
{
        int int_array[MAX];

        int_array[0] = 5;
        int_array[1] = 10;
              .  .  .
        int_array[MAX - 1] = 50;
              .  .  .
}
```

Manipuler les Tableaux

- Puisqu'un tableau est un ensemble ordonné, l'accès à ses éléments peut s'automatiser avec une boucle.
- Il n'existe pas d'opération sur les tableaux - chaque élément doit être manipulé séparément :

```
#include <stdio.h>
#define MAX 26
int main(void)
{
        int counts[MAX], index, ret;
        char ch;

        /* init des comptes à 0... */
        for (index = 0; index < MAX; index++)
            counts[index] = 0;
        /* saisie des valeurs... */
        printf("Entrer des caracteres. Finir par ^D :  ");
        while ((ch = getchar()) != EOF)
                            counts[ch - 'A']++;
        printf ("Liste des comptes :\n");
        for (index = 0; index < MAX; index++)
            printf ("counts[%d]=%d\n", index, counts[index]);
        return 0;
} /* fin de la fonction main */
```

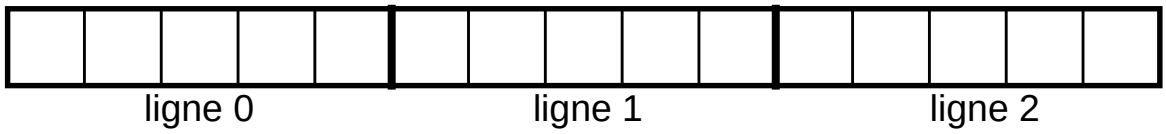
Tableaux à Plusieurs Dimensions

Exemple

```
int matrix[3][5];
```

Cette définition génère un tableau à deux dimensions :

réellement rangé en mémoire comme suit :



Manipuler un Tableau Multi-dimensionnel

La méthode standard pour traiter les tableaux à plusieurs dimensions est d'utiliser des boucles imbriquées, à raison d'une boucle par dimension.

```
#define ROWS 10/* Indice maxi de ligne + 1 */
#define COLS 2/* Indice maxi de colonne + 1 */
#include <stdio.h>
int main(void)
{
     int array[ROWS][COLS], row, col;

/* chargement du tableau avec des valeurs... */
     for (row = 0; row < ROWS; row++) {
        for (col = 0; col < COLS; col++) {
           array[row][col] = row + col;
           printf("Ligne = %d, Colonne = %d.\n", row, col);
        } /* fin du for  */
     } /* fin du for */

/* affichage des valeurs du tableau... */
     for (row = 0; row < ROWS; row++) {
        for (col = 0; col < COLS; col++) {
           printf("Array[%d][%d] = %d.\n", row, col, array[row][col]);
        }
     } /* fin du for */
     return 0;
} /* fin de la fonction main */
```

Initialisation des Tableaux

- On peut affecter des valeurs aux tableaux au moment de leur déclaration. C'est l'*initialisation* des tableaux.
- Dans une initialisation de tableau, sa taille peut être indiquée ou pas. Si elle ne l'est pas, la taille du tableau dépend du nombre de valeurs.
- Syntaxe de l'initialisation :

  `<type> <nom du tableau>[taille opt.]={listes de valeurs};`

  Les valeurs de la liste sont séparées par des virgules.
- Si la taille est indiquée, et qu'il n'y a pas autant de valeurs, le reste du tableau est rempli par des 0.
- Exemples:

```
int arr1[] = {1, 2, 3, 4, 5};/* tableau d'entiers à une dimension */

int arr2[20] = {0}; /* tableau de 20 entiers, 1er élément à 0, les autres
non initialisés*/

char arr3[] = {'a', 'b', 'c', 'd', 'e'};/* tableau de 5 caractères */

double arr4[3][4] = {
            {4.3, 1.2, 5.6, 8.7},  /* row 1 */
            {1.3, 2.4, 5.7, 6.8}  /* row 2 */
};      /* tableau de double à 2 dimension, 3eme ligne non initialisée*/

float arr5[3][4] = { {1.2}, {3.4}, {5.6}};/* tableau de float à 2 dim,
                col. 2, 3, et 4 non initialisées. Col. 1 a les valeurs :
                arr5[0][0] = 1.2, arr5[1][0] = 3.4, et
                arr5[2][0] = 5.6 */
```

Révision de Module

Déclarer des tableaux

Remplir les blancs en se basant sur les 2 déclarations suivantes :

```
int arr1[14];
float arr2[5][5];
```

1. Le premier élément de *arr1* est référencé par . . . . . .

2. En pensant à la disposition en mémoire des tableaux à plusieurs dimensions, le troisième élément à partir du début de la position en mémoire de *arr2* est . . . . . .

3. . . . . référence le dernier élément de *arr2*.

Tableaux

Q. En quoi un tableau diffère-t-il d'une variable *ordinaire* ?

R. _____

Q. Comment accède-t-on aux éléments d'un tableau ?

R. _____

Q. Pourquoi est-il préférable de spécifier la taille des tableaux à l'aide de constantes symboliques ?

R. _____

Révision de Module

Manipulation de Tableau

Ecrire un programme qui déclare un tableau de 5 entiers, demande une valeur pour chaque élément, charge la valeur avec le bon indice, et enfin imprime tout le contenu du tableau.

Travaux Pratiques 6 : Tableaux

Présentation

Introduction à la manipulation des tableaux, liée aux concepts vus dans les TP précédents.

Exercices

****** - La solution sera utilisée dans des TP à venir.

1. **Niveau 1.** *Vérifier* les résultats des programmes vus en révision en les compilant et en les exécutant.

2. ****Niveau 2**. Ecrire un programme pour :

   Déclarer un tableau de 15 entiers et le remplir avec des valeurs de 10 à 150.

   Afficher le contenu du tableau.

   Après affichage, inverser l'ordre des valeurs (array[0] aura 150 et array[14] aura 0). Ne pas utiliser de 2ème tableau pour la transformation.

   Afficher le tableau après inversion.

   Nommer le fichier source `reverse.c`:
   `% cc -Xc reverse.c -o reverse`

3. **Niveau 3**. Ecrire un programme `dimension2.c` pour :

   Déclarer un tableau d'entier bi-dimensionnel de 10 lignes et 2 colonnes. Utiliser des macros pour les dimension des lignes et des colonnes.

   Demander une valeur pour chaque élément du tableau.

   Afficher tout le tableau.

   `% cc -Xc dimension2.c -o dimension2`

Classes d'Allocation

Objectifs

- Décrire la configuration mémoire d'un programme tournant sous SunOS.
- Donner la liste des types d'allocation.
- Identifier les particularités de chaque classe.

Evaluation

Révision de module.

Définition et Déclaration

- **Définition** - La définition (ou *déclaration de définition*) d'un identifiant provoque l'allocation de la mémoire correspondante et l'association du nom avec ce bout de mémoire. Les variables et les fonctions ne sont définies qu'une fois.

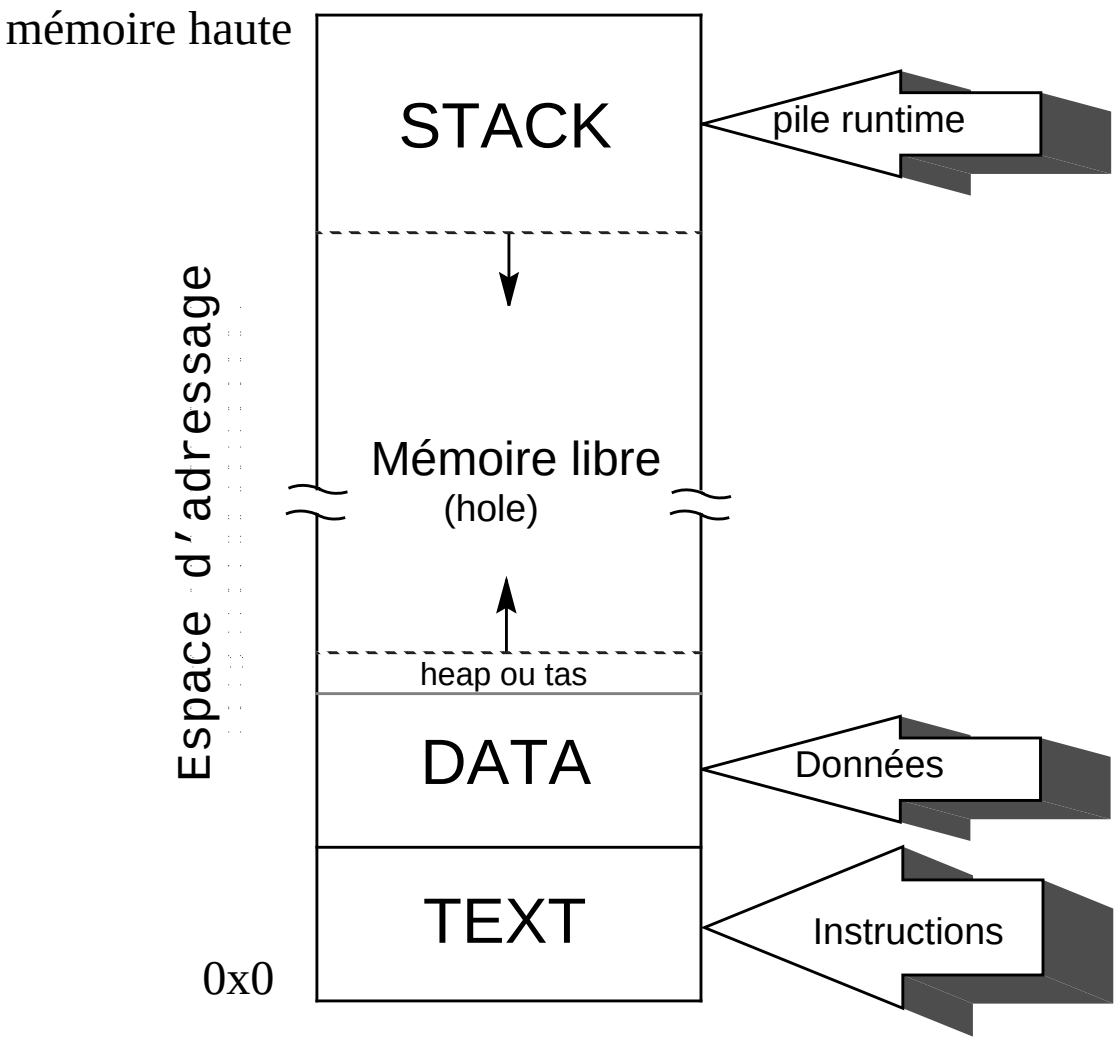
```
/* définition de la variable c3 */
char c3 ; /* allocation d'un octet */

/* définition de my_func */
char my_func(char c1, char c2)
{
        return((c1 < c2) ? c1 : c2);
}
```

- **Déclaration** - Une déclaration (ou *déclaration de référence*) décrit un identifiant en termes de type et de durée de vie. On suppose qu'une définition a été faite ailleurs. Cette déclaration donne les indications nécessaires au compilateur pour lui permettre de faire une interprétation correcte du code.
  Pour les fonctions en particulier, le fait de déclarer aussi les types des arguments s'appelle un *prototypage* et permet au compilateur, entre autre, de faire une vérification du type des arguments passés par l'appelant.

```
int some_func(void)
{
        char my_func(char, char);
        /* suite de la fonction */
}
```

Configuration Mémoire d'un Programme C

Un programme C tournant en mémoire s'appelle un *process* et comporte 3 segments, *stack, data* et *text* :



Classes d'Allocation des Variables du C

- Il y a deux façons de caractériser des variables - *type* et *classe d'allocation*.

- La classe d'allocation détermine :

  la valeur initiale - *quelle est la valeur avant utilisation?*

  durée de vie - *à quel moment la variable existe ?*

  visibilité - *où la variable est-elle connue (accessible) ?*

Allocation `auto`

- **Valeur Initiale** - indéfinie.
- **Durée de vie** - Apparaît au moment de la déclaration et cesse d'exister à la sortie du bloc.
- **Visibilité** - Les variables automatiques sont connues seulement dans le bloc où elles sont définies et dans les blocs inclus dans le bloc de définition.

```
int main(void)
{
       auto int index, value;
    /* est équivalent à...
       int index, value;       */

} /* fin de la fonction main */
```

Allocation en Registre (`register`)

- Les propriétés de l'allocation `register` sont les mêmes que l'allocation automatique (essentiellement parce que `register` est une sous-classe d' `auto`) à une exception près - les registres de la machine n'ayant pas d'adresse mémoire, vous ne pouvez appliquer l'opérateur Adresse-de (&) à une variable de la classe `register`.
- La plupart des machines sont limitées en nombre de registres utilisables pour les besoins de vos programmes.
- En général, compteurs de boucles, indices, pointeurs et autres variables d'utilisation intensive sont de bons candidats à l'allocation `register`.

```
int main(void)
{
        register int index;


} /* fin de la fonction main */
```

Allocation `static`

- **Valeur Initiale** - Une donnée `static` est garantie d'être initalisée à zéro.
- **Durée de vie** - L'espace d'allocation statique est attribué au process au début du programme, et lui est retiré à la terminaison du programme.
- **Visibilité** - Si la variable est définie dans une fonction (locale), alors la variable n'est visible qu'à l'intérieur de cette fonction. Si la variable est définie à l'extérieur d'une fonction (globale), alors elle est visible dans toutes les fonctions qui *suivent* sa définition, mais nulle part en dehors du fichier source de définition.

```
int main(void)
{
        static int index;


} /* fin de la fonction main */
```

- Les fonctions définies avec le mot-clef `static` ne sont pas connues non-plus à l'extérieur de leur fichier de définition.

```
static int func(int cnt, double z)
{

}
```

Allocation `extern`

- **Valeur Initiale** - Toute donnée en allocation `extern` est garantie d'être initialisée à zéro.
- **Durée de vie** - La donnée `extern` a une espérance de vie importante. Elle est allouée au commencement du programme et désallouée à la fin.
- **Visibilité** - Une donnée `extern` est visible dans les fonctions qui suivent sa définition. Une déclaration permet de l'utiliser dans un source différent.

```
%  more source1.c

int index = 4;/*déf.globale*/
extern int display(void);/*décla.*/

int main(void)
{
        float value;
        int result;

        result = display();

} /* fin de main */

%
```

```
%  more source2.c

int display(void)
{
        extern int index;



} /* fin de display */

%
```

- Les fonctions peuvent avoir une spécification d'allocation `static` ou `extern`. `extern` est la spécification par défaut.

Exemple : Initialisations et valeurs initiales

Relever les initialisations et les valeurs initiales que l'on peut supposer dans un programme comme ci-dessous :

```
/* initialisation d'un tableau externe bi-dimensionnel */
int matrix[3][5] = {
        { 1, 2, 3, 4, 5 },
        { 6, 7, 8, 9, 10 },
        { 11, 12, 13, 14, 15 }
};

float x;  /* valeur initale garantie à 0.0 */

int main(void)
{
     /* initialisation de tableaux uni-dim. static et auto */
     static int digits[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
     int int_array[20] = { -1, -2, -3, -4}; /* suite non
initialisée*/

     float y;/* variable auto : val.initiale indéfinie */
     static float z;/* static : garantie de 0.0 inital */

     /* corps du programme... */

} /* fin de la fonction main */
```

Table Récapitulative

mot-clef d'allocation	valeur Initiale	durée-de vie	Visibilité	Localisation
`auto`	Indéfinie	bloc	bloc	la pile (STACK)
`register`	Indéfinie	bloc	bloc	registre(si possible sinon STACK)
`static`	0	prog	bloc (ou fichier source)	segment DATA
`extern`	0	prog	fichier source (ou bloc)	segment DATA

Allocation

Q. Quelles sont les principales parties ou *segments* de l'image mémoire d'un process ?

R. _____ _____ _____

Q. Quelles différences entre *définition* et *déclaration* ?

R. _____

_____

Q. Quelles sont les 4 classes d'allocation ?

R. ____ _____ _____ _____

Q. Quels sont les 3 attributs d'une variable déterminés par sa classe d'allocation?

R. _____ _____ _____

Q. Quel segment est utilisé pour l'allocation des variables *automatiques*?

R. _____

Q. Quel effet la déclaration `static` d'une variable dans une fonction produit-elle?

R. _____

Q. Quelle classe d'allocation permet le partage de variables entre fonctions ?

R. _____

Pointeurs et Adresses

Objectifs

- Ecrire des expressions qui donnent des adresses.
- Référencer indirectement des données à l'aide de variables pointeurs.
- Utiliser les pointeurs vers des variables.
- Passer des pointeurs vers des variables à des fonctions - *appel par référence*.
- Utiliser l'arithmétique des pointeurs.

Evaluation

Travaux Pratiques 8 et révision de module.

L'opérateur *Adresse De*

Format

& *<lvalue>*

- L'opérateur adresse est utilisé pour récupérer l'adresse d'une variable - son emplacement en mémoire :

```
#include <stdio.h>
int main(void)
{
        int number;

        printf("Entrer un entier :  ");
        scanf("%d", &number);
        printf("L'adresse de \"number\" est 0x%x.\n", &number);
        printf("La valeur de \"number\" est %d.\n", number);

        return 0;
} /* Fin de la fonction main */
```

- Sur les Stations de Travail Sun, les pointeurs sont sur 32 bits, quelle que soit le type de la variable pointée.

Principes des Pointeurs et Déclarations

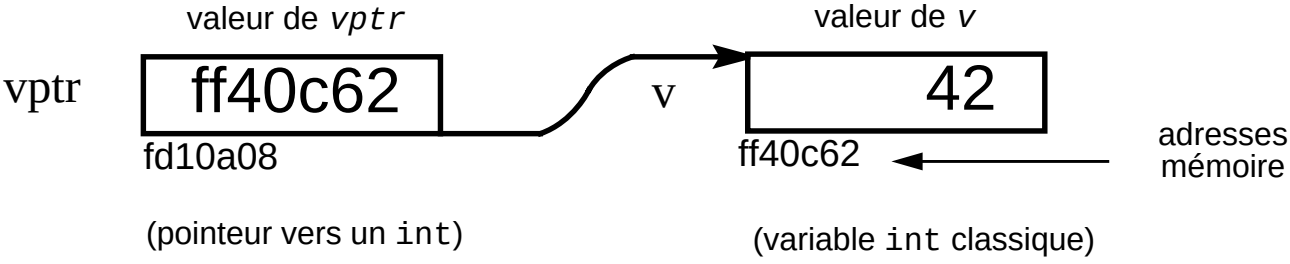
- Une donnée élémentaire est stockée dans une *cellule mémoire* à une *adresse* précise.

- Les variables pointeurs réprésentent l'*emplacement* d'une donnée plutôt que sa *valeur* - ainsi, les pointeurs contiennent l'*adresse* d'une donnée :

Format

```
<type> *identifiant;
```

```
int v = 42;       /* variable entière */
int *vptr; /*pointeur vers une variable entière*/
vptr = &v;

/* qui peut aussi s'écrire... */
int v = 42, *vptr = &v;
```



Nomenclature Conventionnelle des Pointeurs

Par convention, le nom d'une variable peut indiquer que c'est un pointeur.

```
#include <stdio.h>
int main(void)
{
     int  *iptr, val;
     float  *fptr, real;

     iptr = &val;
     printf("Entrer un entier :  ");
     scanf("%d", iptr);
     fptr = &real;
     printf("Entrer un réel :  ");
     scanf("%f", fptr);

     printf("valeur de val: %d, real: %f.\n", val, real);
     printf("Adresse de val: 0x%x, de real: 0x%x.\n", iptr, fptr);
     return 0;
} /* fin de main */
```

Opérateur d'indirection

Format

* *<expression pointeur>*

- L'*Indirection* ou *adressage indirect* s'utilise avec une variable qui, plutôt que contenir des données, détient l'adresse de ces données.
- L'opérateur d'*indirection* peut être vu comme le complément de l'opérateur *adresse de* - & donne l'adresse de la variable concernée, * donne les données pointées.

```
#include <stdio.h>
int main(void)
{
     static int val1 = 42;
     int  *ptr, val2;
     /* affecte l'adresse de val1 à ptr... */
     ptr = &val1;
     /* Indirection: donne la valeur pointée par ptr à val2... */
     val2 = *ptr;
     /* val1, val2, et *ptr sont maintenant égales... */
     printf("val1=%d;  val2=%d; *ptr=%d\n", val1, val2, *ptr);
     return 0;
} /* fin de la fonction main */
```

- La déclaration `void *ptr` (pointeur vers void) est légale. `void *` est un pointeur générique capable de pointer tout type de donnée sans restriction. Il a la même représentation et le même alignement qu'un pointeur sur `char`, et il *ne peut pas* être déréférencé. On utilise la convertion explicite (*cast*) dans ce cas.

Résumé sur les Opérateurs & et *

Opérateur *Esperluette* &

- Utilisé pour renvoyer l'*adresse* d'une variable :

```
int index;
printf("Adresse de index == %x", &index);
```

Opérateur Asterisque *

- Utilisé pour *déclarer* une variable pointeur :

```
int    *iptr;
int    **ipp; /* pointeur sur pointeur */
char   *cptr;
double *dptr;
```

- Utilisé pour déréférencer une variable ou une expression pointeur - c'est-à-dire, accéder à la valeur pointée par la variable ou l'expression :

```
/* Supposons que iptr contient l'adresse d'une
   variable index déclarée comme un entier : */
/* iptr = &index;... */
*iptr = 5; /* modification indirecte de la valeur */
printf("valeur chargée dans index = %d\n", index);
```

Révision Partielle

opérateurs & et *

A partir des déclarations suivantes, quelles sont les instructions ci-dessous valides ? Justifier les réponses et indiquer les résultats dans les cas valides. (Remarque : le résultat d'une instruction - valide - dépend de la précédente.)

```
int *numptr, num;
char *chptr;
float *realptr;
float real;

1) *chptr = 'A';

2) numptr = &num;

3) *numptr = 5;

4) printf ("%d", num);

5) real = 7;

6) *realptr = real;

7) realptr = &real;

8) *realptr = num++;

9) printf ("num = %d\n", num);

10) *chptr = &'A';
```

Fonctions : Appel par Référence

- Les pointeurs sont passés aux fonctions lorsque les arguments doivent être modifiés par celles-ci - *appel par référence*

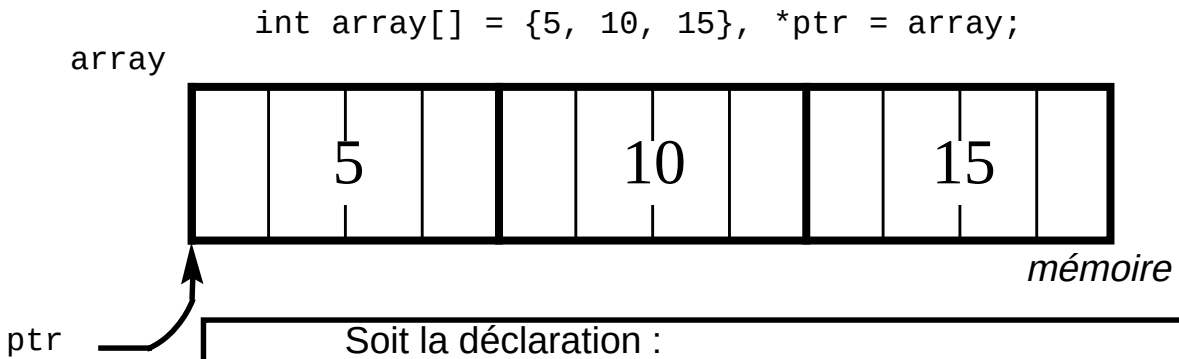
- L'opérateur d'indirection doit être utilisé dans la fonction pour accéder aux données :

```
#include <stdio.h>
int main(void)
{
    int num = 5;
    void func(int *);

    printf("Main: Avant l'appel, num = %d.\n", num);
    func(&num);
    printf("Main: Après l'appel, num = %d.\n", num);
    return 0;
} /* fin de main */

void  func(int *ptr)
{
    *ptr += 2;    /* plus 2 à travers le pointeur */
    printf("Func: num = %d.\n", *ptr);
} /* fin de func */
```

```
% a.out
Main: Avant l'appel, num = 5.
Func: num = 7.
Main: Après l'appel, num = 7.
%
```

Pointeurs et Tableaux

- Le nom d'une variable tableau est une constante de type pointeur vers le premier élément du tableau :



Soit la déclaration :

```
#define MAX 12
#define LAST (MAX - 1)
int i, array[MAX], *ptr;
```

ce qui suit est équivalent :

```
ptr = array;
ptr = &array[0];
```

et `array[i]` est équivalent à `*(ptr + i)`

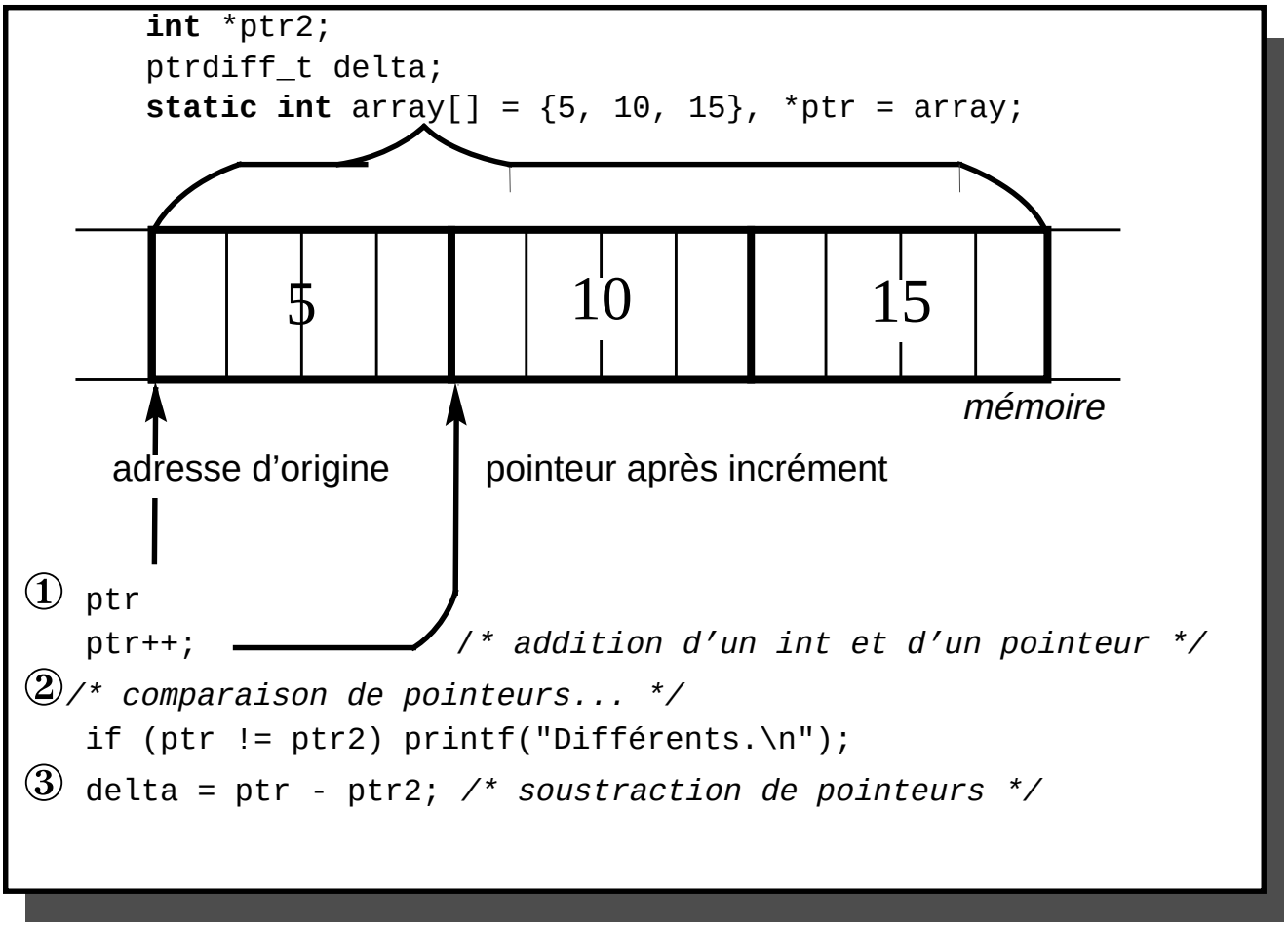
et toutes les propositions suivantes sont vraies :

```
array == ptr
&array[0] == ptr
array[0] == *ptr
array[4] == *(ptr + 4)
&array[7] == ptr + 7
array[LAST] == *(ptr + LAST)
&array[LAST] == ptr + LAST
```

Arithmétique des Pointeurs

- Ensemble d'opérations définies sur les pointeurs :

1. L'addition ou la soustraction d'un entier avec un pointeur donne un pointeur.
2. Comparaison de pointeurs.
3. Soustraction de Pointeurs. Le résultat est le nombre d'objets entre les deux adresses. Le type de l'entier signé résultat est *ptrdiff_t* défini dans `<stddef.h>`.



Révision Partielle

Calculs de pointeurs

Soient les déclarations suivantes, quels sont les résultats des instructions ?
Remarque : Les exemples sont inter-dépendants.

```
int index, numbers[5], *nptr = numbers;
numbers[0] = 2;
numbers[1] = 4;
numbers[2] = 6;
numbers[3] = 8;
numbers[4] = 10;
```

```
1) index = *nptr;

2) index = *(nptr + 2);

3) index = *(nptr++);

4) printf("*nptr = %d\n",*nptr);

5) index = nptr - numbers;

6) index = *nptr++;

7) printf("nptr pointe l'élément %d.\n", nptr - numbers);

8) index = ++(*nptr);
```

Les Notations Pointeur et Indice de Tableau

Les notations *pointeur* et *indice de tableau* sont équivalentes :

```
#include <stddef.h>
#include <stdio.h>
#define MAX 42
int main(void)
{
/* déclaration de tableau, pointeur et pointeur de fin...*/
    int index, array[MAX], *ptr, *end = &array[MAX-1];
    ptrdiff_t  delta;/* ptrdiff_t déf. dans stddef.h */

    /* Notation indice... */
    for (index = 0; index < MAX; index++)
        array[index] = index;

    /* notation pointeur équivalente */
    for (ptr = array; ptr <= end; ptr++) {
        delta = ptr - array;
        printf("Array[%d] = %d.\n", delta, *ptr);
    } /* fin de for  */
    return 0;
} /* fin de main */
```

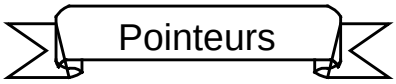
Passage de Tableaux aux Fonctions

Le contenu d'un tableau passé en argument à une fonction peut être modifié, car le nom du tableau est un pointeur sur le premier élément. Ainsi les tableaux sont toujours passés en référence

:

```
#include <stdio.h>
#define SIZE 15
int main(void)
{
        double arr[SIZE], *ptr;
        void load_array(double array[]);

        load_array(arr);/* Remplissage du tableau */
        for (ptr = arr; ptr < &arr[SIZE]; ptr++){
            printf("Array[%d] = %f\n", ptr-arr, *ptr);
        } /* fin de for
}   /* fin de main */

void load_array(double array[])
{
        int whole = 1;
        double *pos;
        for (pos=array ; pos <= &array[SIZE]-1; pos++,whole++){
            *pos = whole / 3.0 ;
        } /* fin de for */
}   /* fin de load_array
```

Révision de Module



Q. En quoi un pointeur diffère-t-il d'une variable *ordinaire* ?

R. _____

Q. Quelle est la définition du terme *lvalue* ?

R. _____

Q. Quel est l'opérateur *adresse de* et à quelles expressions s'applique-t-il ?

R. _____

Q. Quel opérateur est utilisé pour déclarer un pointeur ?

R. _____

Q. Pourquoi est-il important d'initialiser un pointeur avant de l'utiliser ?

R. _____

Q. Qu'est-ce que l'indirection, quand s'en sert-on et avec quel opérateur ?

R._____

Q. Comment une fonction peut-elle modifier un argument dans l'appelant ?

R._____

Q. Quelles sont les opérations autorisées sur les pointeurs ?

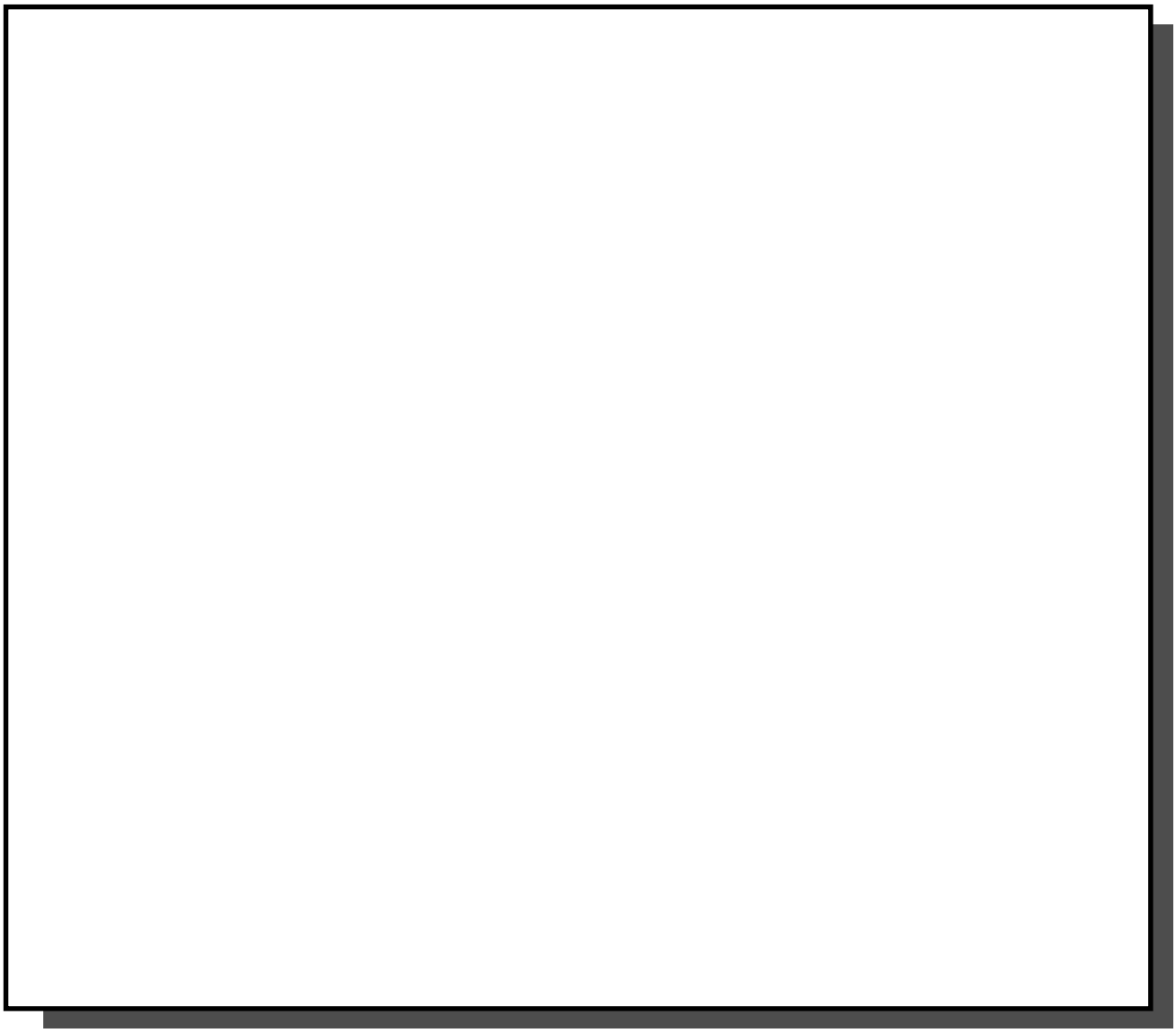
R. _____

Q. Décrire les relations entre la notation pointeur et indice dans le contexte de l'accès aux tableaux. Donnes quelques exemples.

R._____

Appel par référence

Ecrire un programme contenant 2 fonctions *main()* et *swap()*. La fonction *swap()* échange deux valeurs de variables de la fonction *main()*.
Conseil : la fonction doit utiliser pointeurs et indirection pour modifier les variables dans l'appelant



Travaux Pratiques 8 : Pointeurs et Adresses

Présentation

Introduction à la programmation des pointeurs et de l'indirection.

Exercices

1. **Niveau 1**. *Vérifier* les résultats du programme swap de la révision du module.

2. **Niveau 2**. Ecrire un programme pour :

   Déclarer et initialiser un tableau d'`int` de 10 éléments avec des valeurs quelconques.

   Appeler une fonction qui trouve la moyenne des valeurs, change les valeurs du tableau en *(valeur initiale * moyenne)*, et retourne la moyenne calculée - appeler cette fonction *`average()`*: utiliser la notation pointeur.

   Remarque: La fonction retournera un `double` et doit recevoir le début et la fin du tableau.

   Appeler une fonction qui affiche les valeurs du tableau, *avant* et *après* l'appel à *`average()`*. Appeler cette fonction *`print_array()`*.

   Faire afficher par la fonction *`main`* la moyenne après l'appel à *`average()`*. Pour tous les accès au tableau utiliser la notation pointeur, *jamais* la notation indicée. Nommer le fichier source *`average.c`* :

   ```
   % cc -Xc average.c -o average
   ```

3. **Niveau 3**. Réécrire le programme des TP 6 `reverse.c` en utilisant la notation pointeur au lieu de la notation indicée. Le nommer `preverse.c`.

   En plus, écrire une fonction *`print_array()`* pour prendre en charge tous les afffichages du contenu du tableau.

- voir page suivante -

4. **Niveau 4**. *(Facultatif)* : Ecrire un programme pour :

   Déclarer une variable pointeur de fonction retournant un `int`.

   Affecter à la variable l'adresse de la fonction ***`printf()`***.

   Afficher un message en utilisant la variable pour effectuer l'appel à la fonction.

   Nommer le programme `funcptr.c` :

   ```
   % cc -Xc funcptr.c -o funcptr
   ```

Chaînes et Caractères

Objectifs

- Définir les chaînes dans le contexte de SunOS et du Langage C.
- Inclure le package String dans un programme C.
- Utiliser le package String pour manipuler les chaînes.
- Reconnaître et utiliser les macros de classification et de traitement des caractères dans un programme C.

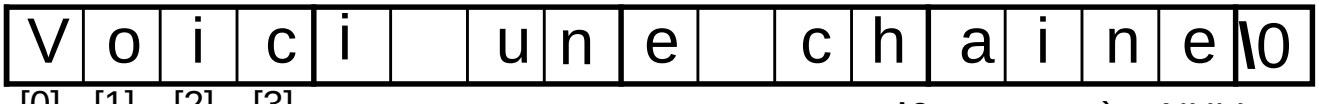
Evaluation

Travaux Pratiques 9 et révision de module.

Introduction aux Chaînes de Caractères

- Les chaînes de caractères et les constantes (chaînes littérales) sont des séquences de caractères (octets ou *bytes*) entre guillemets (") , terminées par le caractère *null* -'\0' :

```
char str[] = "Voici une chaine";
```



Exemples de déclaration de chaînes :

```
char *str2 = "Chaine deux"; /* en lecture/écriture */
char str3[ ] = "Chaine trois"; /* lecture seule */
```

- Les constantes chaînes adjacentes sont automatiquement concaténées :

```
printf("Cette chaîne est concaténée avec \n"
  "cette chaîne pour qu'on puisse imprimer \n"
  "plusieurs lignes avec printf() sans \n"
  "aucun problème.\n");
```

Saisie de Chaînes : *scanf()*

- La spécification de format pour la lecture de chaînes avec *scanf()* est **%s**, l'argument correspondant devant *pointer* vers un tableau de caractères.
- La valeur retournée par *scanf()* est un entier. En cas de réussite, c'est le nombre d'items correctement entrés. En cas d'erreur avant toute conversion, *scanf()* renvoie *EOF*.
- Le même %s est utilisé avec *printf()* pour imprimer les chaînes :

```
#include <stdio.h>
#define MSG "Amusez-vous bien !"
#define ASIZE 40

int main(void)
{
        char first_name[ASIZE], last_name[ASIZE];

        printf("Entrer vos nom et prenom :  ");
        scanf("%s%s", first_name,last_name);
        printf("Merci %s %s,\n", first_name, last_name);
        printf("%s\n", MSG);
        return 0;
} /* fin de main */
```

Lecture et Ecriture des Chaînes : *gets()* et *puts()*

Format

```
#include <stdio.h>
char *gets(char *s);
int puts(const char *s);
```

- La fonction *gets()* renvoie le pointeur que vous lui passez en argument (assurez vous que l'allocation est bien faite), ou un pointeur *NULL* si aucune entrée n'a pu être réalisée sur l'*entrée standard*.
- *gets()* n'a pas besoin d'indication de format.
- *puts()* affiche une chaîne sur la *sortie standard* et *rajoute* un caractère *newline*.
- Normalement, *puts()* renvoie une valeur non-négative, et *EOF* si une erreur d'écriture arrive.

```
#include <stdio.h>
#define PSIZE 256

int main(void)
{
     char input[PSIZE];/* espace alloué en lecture/écriture */

     printf("Taper une ligne suivie de <Retour-Chariot>:  ");
     gets(input);
     printf("\n\nVous avez saisi la ligne suivante :\n");
     puts(input);/* puts rajoute un newline - '\n' */

     return 0;
} /* fin de main */
```

Package Chaînes de Caractères (String)

- Les fonctions de manipulation des chaînes de caractères sont incluses dans la librairie standard du C.

- Pour éviter d'avoir à prototyper les fonctions de manipulation de chaînes dans vos programmes, vous devez inclure le fichier `<string.h>`.

Copie et Concaténation de Chaînes

Format

```
#include <string.h>
 /* copie s2 dans s1 (écrase s1) */
char *strcpy(char *s1, const char *s2);

  /* concatène s2 après s1 */
char *strcat(char *s1, const char *s2);
```

- *strcpy()* et *strcat()* renvoient toutes les deux le pointeur *s1*.
- Exemples d'utilisation de *strcpy()* et *strcat()*:

```
#include <stdio.h>
#include <string.h>
#define MAX 256
#define QUOTE "Le langage : une forme de bégaiement organisé."

int main(void)
{
    char str1[MAX], str2[MAX], both[2*MAX];

    printf("Entrer votre citation préférée <%d caractères:  ",
         MAX);
    gets(str1);
    strcpy(str2, QUOTE);/* copie QUOTE dans str2 */
    strcpy(both, str1); /* copie la 1ere chaîne dans both */
    strcat(both, str2); /*concatène la 2eme chaîne sur both */
    printf("1ere : \"%s\"", str1);
    printf("2eme : \"%s\"", str2);
    printf("ensemble : \"%s\"", both);

    return 0;
} /* fin de main */
```

Comparaison et Longueur de Chaînes

Format

```
#include <string.h>
size_t strlen(const char *s1);
int strcmp(const char *s1, const char *s2);
```

- *strlen()* renvoie la longueur de *s1*, sans compter le caractère *nul* de fin.
- *strcmp()* renvoie un entier négatif nul ou positif selon que la première chaîne est respectivement inférieure, égale ou supérieure, selon l'ordre *lexicographique*, à la deuxième chaîne.

```
#include <stdio.h>
#include <string.h>
#define MAX 80

int main(void)
{
        char str1[MAX], str2[MAX], *ptr, ch;
        int len = 0, result = 0;
        printf("Entrer une chaîne <%d caractères:  ", MAX);
        gets(str1);
        len = strlen(str1);     /* compte les octets de str1 */
        printf("Vous avez tape %d caractères.\n", len);
        printf("Entrer une autre chaîne <%d caractères :  ", MAX);
        gets(str2);
        result = strcmp(str1, str2);/*compare str1 et str2 */
        if (result < 0)
            printf("Alphabétiquement, %s < %s.\n",str1,str2);
        else if (result > 0)
            printf("Alphabétiquement, %s > %s.\n", str1, str2);
        else
            printf("Les chaînes sont égales !.\n");
        return 0;
} /* fin de main */
```

Recherche de Caractères dans les Chaînes

Format

```
#include <string.h>
char *strchr(const char *s, int c);
```

- *strchr()* renvoie le pointeur sur la première occurence de *c* dans *s*, ou le pointeur *NULL* si non-trouvé.

```
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#define MAX 40

int main(void)
{
     char play[MAX], *cptr;
     int ch, pos;

     printf("Entrer votre pièce de Molière préférée :  ");
     gets(play);
     printf("Entrer un seul caractère :  ");
     ch = getchar( );
     cptr = strchr(play, ch);/* ch est-il dans play ? */
     if (cptr == NULL)
        printf("Le caractère '%c' n'est pas dans [%s]\n",ch,play);
     else {
        pos = cptr - play;
        printf("Nom tronqué : %s\n", cptr);
        printf("play[%d] est '%c'.\n", pos, play[pos]);
     } /* fin de if */
     return 0;
} /* fin de main */
```

Convertir les Chaînes en Nombres

- Les fonctions de conversions de chaînes sont utilisées pour convertir des chaînes *représentant* des nombres dans leur *véritable* valeur.
- Les fonctions ***atoi()***, ***atof()*** et ***atol()*** s'utilisent obligatoirement en incluant `<stdlib.h>`.
- La valeur retournée par ***atoi()***, ***atof()*** et ***atol()*** est le résultat de la conversion. Le code de retour de ***sscanf()*** est ***EOF*** si une erreur de lecture arrive avant toute conversion ; sinon, c'est le nombre d'items saisis qui est retourné.

```
#include <stdlib.h>
int atoi(const char *nptr) - conversion ASCII en entier. Exemple:
                num = atoi("47");

double atof(const char *nptr) - conversion ASCII en double.Exemple:
                dnum = atof("47");

long atol(const char *nptr)- conversion en long int. Exemple
                lnum = atol("47");


#include <stdio.h>
int sscanf(const char *s, const char *format, ...) - lit un nombre
                 depuis une chaîne s dans une variable selon format.
Exemple:
                sscanf ("47", "%f", &fnum)
Lit un réel depuis la chaîne "47" dans la variable fnum.
```

Remarque : atoi("abc") retourne 0 mais atoi("0") retourne aussi 0. On ne peut pas faire la différence.

La Fonction *sprintf()*

Format

```
#include <stdio.h>
int sprintf(char *s, const char *format, ...);
```

- La fonction ***sprintf()*** est la fonction d'affichage correspondant à ***sscanf()***:
- ***sprintf()***est similaire à ***printf()***, mais au lieu d'écrire sur la *sortie standard*, le résultat est écrit dans le tableau pointé par *s*.
- ***sprintf()*** renvoie le nombre de caractères écrits, non-compris le caractère '*\o*' qui est ajouté à la fin.

```
...
char str[256], *mess = "Utiliser sprintf";
int num = 13;
...
sprintf(str, "Nouvelle chaîne avec les valeurs de\n"
       " num (%d) et mess (%s).\n\n", num, mess);
...
```

Conversions de Chaînes en Nombres

Exemple de programme utilisant *atoi()* et *sscanf()*:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 80

int main(void)
{
        char string1[MAX], string2[MAX];
        int ival;
        double dval;

        printf("Entrer un entier :  ");
        gets(string1);
        ival = atoi(string1);
        printf("Entrer un réel :  ");
        gets(string2);
        sscanf(string2, "%lf", &dval);
        printf("%f plus %d = %f.\n", dval, ival, ival+dval);
        return 0;
} /* fin de main */
```

Opérations sur les Caractères

- Macros de classification et de conversion :

```
#include <ctype.h>
```

`int isalpha(int c);`	*c* est-il un caractère alphabétique ?
`int isupper(int c);`	*c* est-il une majuscule ?
`int islower(int c);`	*c* est-il une minuscule ?
`int isdigit(int c);`	*c* est-il un chiffre (0-9) ?
`int isxdigit(int c);`	*c* est-il un chiffre hexadécimal (0-9, a-f or A-F) ?
`int isalnum(int c);`	*c* est-il un alphanumérique (a-zA-Z or 0-9) ?
`int isspace(int c);`	*c* est-il un caractère d'espacement ? (par exemple : espace, tabulation)
`int ispunct(int c);`	*c* est-il une ponctuation ? (par exemple : ?, !)
`int isprint(int c);`	*c* est-il un caractère imprimable ? (y-compris l'espace)
`int iscntrl(int c);`	*c* est-il un caractère de *contrôle* ? (par exemple : le caractère delete ou un caractère de contrôle classique)
`int isgraph(int c);`	*c* est-il un caractère dessinable ? (sans l'espace)
`int toupper(int c);`	conversion en majuscule. (vérifie *`islower()`*)(fonction)
`int tolower(int c);`	conversion en minuscule. (vérifie *`isupper()`*)(fonction)

- Les macros ci-dessus prennent un `int` en paramètre, si celui-ci peut être représenté comme un `unsigned char` ou est égal à *`EOF`*.

- La valeur retournée par les fonctions *`isxxx()`* est soit vraie (non nulle) soit fausse (nulle) . La valeur retournée par les fonctions *`toxxx()`* est soit le caractère converti si la conversion a pu être faite, soit le paramètre inchangé.

Opérations sur les Caractères (suite)

Exemple d'utilisation de *isalpha()* et *toupper()* :

```
#include <stdio.h>
#include <ctype.h>
#define MAX 80

int main(void)
{
      char *aptr, *optr, alphas[MAX], other[MAX];
      int ch;

      aptr = alphas;/* charge l'adresse du tableau... */
      optr = other;
      printf("Entrer une série de caractères :  ");

      while ((ch = getchar( )) != '\n') {
         if (isalpha(ch)) {
            *aptr = toupper(ch);
            aptr++;
            if (aptr == &alphas[MAX-1])
               break;
         }
         else {
            *optr = ch;
            optr++;
            if (optr == &other[MAX-1])
               break;
         }
      } /* fin de while */

      *aptr = '\0';/* finir par un caractère nul... */
      *optr = '\0';
      printf("Caractères alpha (en Maj.): %s\n", alphas);
      printf("Autres caractères : %s\n", other);
      return 0;
} /* fin de main */
```

Révision de Module

Chaînes

Q. Quelles sont les différences entre une chaîne et un tableau de `char` ?
R. _____
Q. Quelle fonction peut-on utiliser pour compter les caractères d'une chaîne ?
R. _____

Q.Quelle est la fonction qui concatène deux chaînes ?
R. _____
Q. Quelles sont les valeurs possibles en comparant 2 chaînes avec *`strcmp()`* ?
R. _____ _____

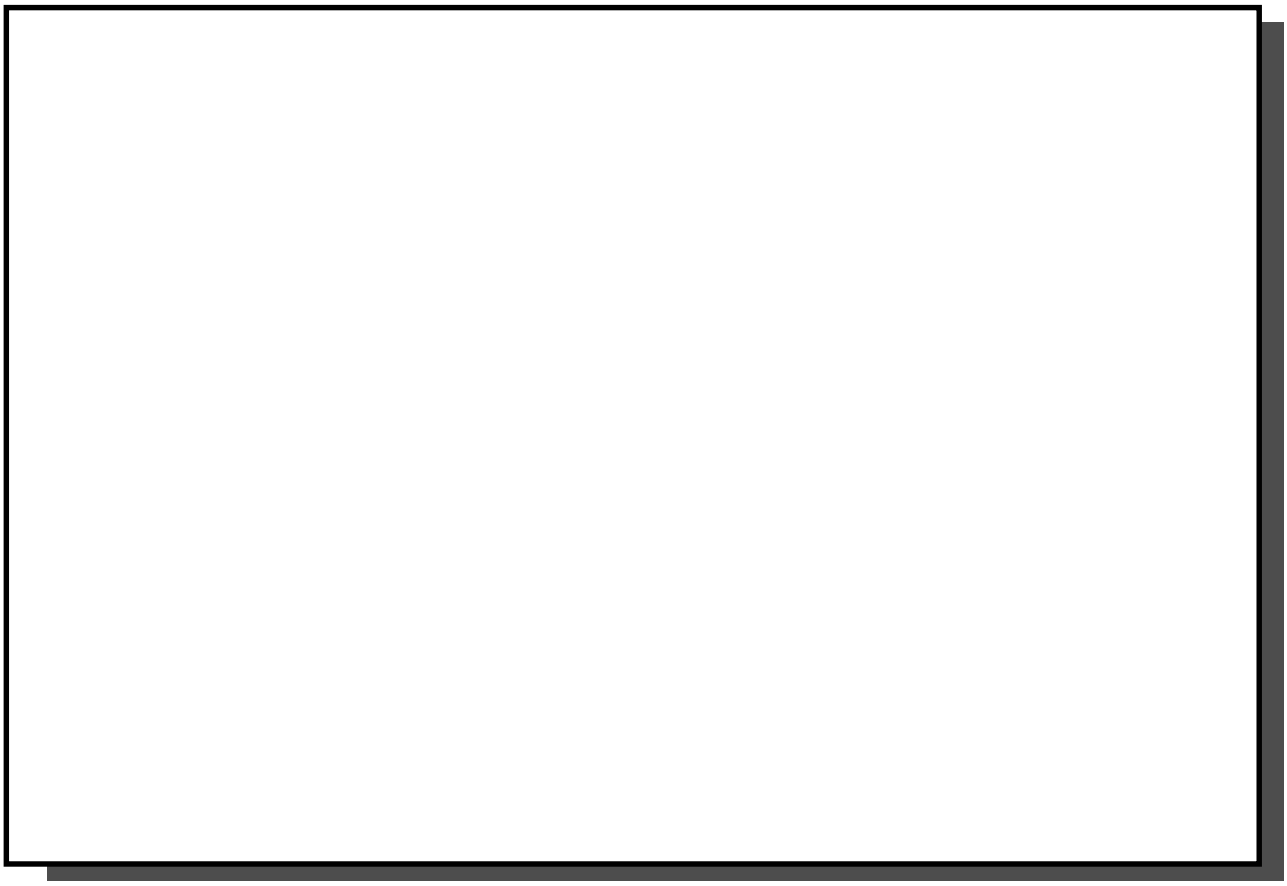
Caractères

Q. Quel fichier doit-on inclure (`#include`) pour utiliser les macros de conversion de caractères ?
R. _____

Q. A part les guillemets, quelles sont les différences entre 'a' et "a" ?
R. _____
Q. Quelle macro de classification vérifie qu'un caractère est *imprimable* ?
R. _____

Conversion

Ecrire un programme qui déclare une chaîne de 40 `char`, demande des caractères au hasard, charge la saisie dans le tableau de `char`, et affiche tous les caractères non-alphabétiques rencontrés.



Travaux Pratiques 9 : Chaînes et Caractères

Présentation

Familiarisation avec les chaînes et les fonctions de manipulation, ainsi qu'avec les macros de classification et de conversion de caractères.

Exercices

1. **Niveau 1**. *Vérifier* Les résultats du programme de révision en le compilant et en l'exécutant.

2. **Niveau 2**. Réécrire le programme `loops.c` des TP 5 en utilisant les macros de classification *`isalpha()`* et *`islower()`*, plutôt qu'en testant le code ASCII directement.

   Nommer le programme `lupes.c`.

   facultatif : Convertir toutes les majuscules saisies en minuscules en utilisant la macro `tolower()`. Ne pas oublier `<ctype.h>`.

3. **Niveau 3**. Ecrire une fonction qui émule *`strlen()`* - l'appeler *`slen()`*. Le programme doit :

   Demander 2 chaînes à l'utilisateur.

   Appeler *`slen()`* pour compter les caractères dans les chaînes, mais sans le caractère *nul* de fin - utiliser la notation pointeur.

   Afficher la longueur de chaque chaîne.

   En utilisant les fonctions *`strcpy()`* et *`strcat()`*, concaténer dans une troisième chaîne, les 2 chaînes saisies en insérant " *** " entre les deux. S'assurer que la chaîne de destination est assez grande pour accueillir les deux chaînes.

   Afficher la chaîne résultat.

   Ne pas oublier `<string.h>`. Nommer le programme `stringy.c`:

   ```
   % cc -Xc stringy.c -o stringy
   ```

Structures, Unions, Définition de Type et Type Enumerés

Objectifs

- Déclarer des variables structures.
- Initialiser des variables structures.
- Ecrire des expressions qui référencent des membres de structures.
- Utiliser la macro *`offsetof()`*.
- Utiliser des structures imbriquées.
- Définir des pointeurs de structures et référencer des membres à travers les pointeurs.
- Créer et utiliser des *unions* dans un programme C.
- Déclarer des données de type énuméré.
- Créer de nouveaux types en utilisant `typedef`.

Evaluation

Travaux Pratiques 10 et révision de module

Principes des Structures

- une structure est un agrégat, dont les éléments peuvent être de différents types :

```
struct {
     int a;
     int b;
} xyz;   /* Similaire à int xyz[2]; */


struct {
     char name[40];
     float salary;/* Contrairement aux tableaux,
                     les structures peuvent regrouper*/
     int paygrade;/* des éléments de différents types.  */
} employee;
```

- Les éléments d'une structure sont des *membres* ou *champs*.
- Chaque *membre* d'une structure a un identifiant unique.

Déclaration de Structure

Format

```
struct <id-de-structure> {
   <membre 1>
   <membre 2>
   ...
   <membre n>
}  <identifiant_de_variable>;
```

- Quand l'*id-de-structure* est utilisé, un nouveau type de donnée est ajouté au programme. Ce nouveau type s'appelle `struct <id-de-structure>` et peut être utiliser par la suite pour déclarer des variables.
- L' *`identifiant_de_variable`* provoque l'allocation de mémoire correspondant. S'il est omis, aucun espace mémoire n'est alloué.

```
#include <stdio.h>
int main(void)
{
        struct db {
            int entry_code;
            short age;
            char first_name[16];
            char last_name[32];
        }; /* fin de struct db, pas d'allocation */

        /* déclaration de 3 variables,la mémoire est allouée */
        struct db first, second, third;

        /* ... suite du programme ... */
} /* fin de main */
```

Forme et Taille d'une Structure

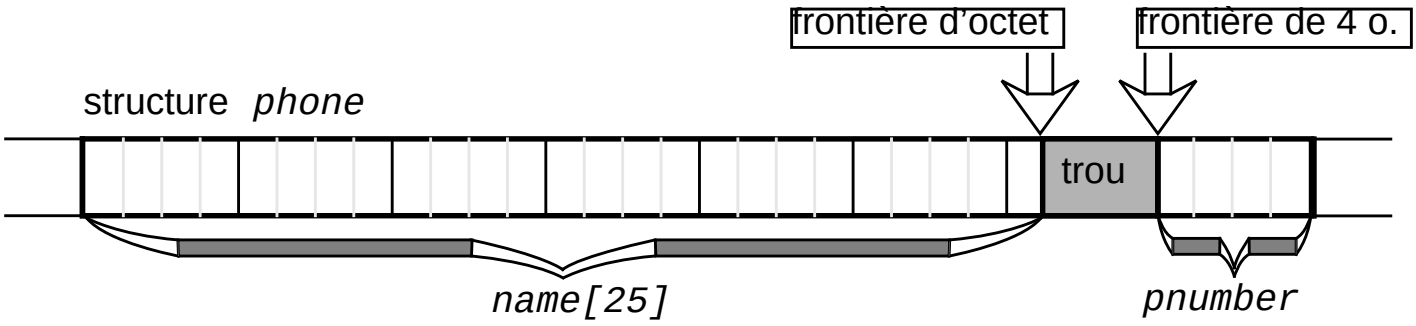
- Toutes les machines / architectures ont leurs règles :

  Architectures Sun4 (SPARC) : *alignement aux frontières naturelles des types : 8 octets pour double, 4 octets pour float, 4 octets pour int, 2 octets pour short, 1 octet pour char.*

- La taille d'une structure est donnée par `sizeof(struct_identifier)`. (Attention : la taille d'une structure n'est pas forcément égale à la somme des tailles de ses membres.)

- Dans l'exemple suivant sur Sun4 , il y a un *trou* de trois octets non-utilisés.

```
...
struct phone {
     char name[25];
     int pnumber;
} xphone;
...
```



Référencer des Membres de Structure

Format

<nom_de_struture> . <nom_de_membre>

- Les membres de structures sont référencés à l'aide de l'opérateur "." .
- L'opérande de gauche de l'opérateur "." doit être un nom de variable structure.
- L'opérande à droite de l'opérateur "." doit être un nom de membre de structure :

```
#include <stdio.h>
int main(void)
{
        struct record {
            int key;
            char first[16];
            char last[32];
        } rec;

        printf("Entrer votre nom :  ");
        gets(rec.first);
        printf("Entrer votre prénom :  ");
        gets(rec.last);
        printf("Entrer un entier :  ");
        scanf("%d", &rec.key);
        printf("Le nom est %s %s.\n", rec.first, rec.last);
        printf("Le 1er car. du nom est %c.\n", rec.first[0]);
        printf("Le code est %d.\n", rec.key);
        return 0;
} /* fin de main */
```

Utilisation de la macro *offsetof()*

Format

```
#include <stddef.h>
size_t offsetof(type, nom_de_membre);
```

- La macro ***offsetof()*** renvoie l'offset en octets du membre de structure, ***nom_de_membre***, depuis le début de la structure, référencé par ***type***.

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

int main(void)
{
      struct address {
         char first_name[32];
         char last_name[32];
         char street[128];
         char city[26];
         char state[3];
         int zip_code;
      };

      printf("l'offset du membre zip_code dans struct address = %d\n",
       offsetof(struct address, zip_code));
      return 0;
} /* fin de main */
```

Imbrication de Structures et Accès

- `struct` est un type de membre de structure valide, ainsi on peut imbriquer des structures.
- Il n'y a pas de limite à la profondeur d'imbrication des structures, si ce n'est la capacité du programmeur à s'y retrouver.

```
#include <stdio.h>
int main(void)
{
        struct xname {
            char first[20];
            char last[30];
        };
        struct record {
            int key;
            struct xname name;
        } rec;

        printf("Entrer un nom de famille :  ");
        gets(rec.name.first);
        printf("Entrer un prénom :  ");
        gets(rec.name.last);
        printf("Entrer un entier :  ");
        scanf("%d", &rec.key);
        printf("Nom=%s %s.\n",
            rec.name.first,rec.name.first);
        printf("Code = %d.\n", rec.key);
        return 0;
} /* fin de main */
```

Initialisation de Structure

- Les structures `static`, `extern`, et `auto` sont initialisées de la même manière que les tableaux, en utilisant des expressions constantes.
- L'ordre dans lequel on place les constantes doit correspondre à l'ordre d'apparition des membres :

```
#include <stdio.h>
int main(void)
{
        struct database {
            char name[40];
            int key;
        };

/*      initialisation de rec.name et rec.key... */
        static struct database rec = {"Michel Martin", 42};
        struct database rec1 = { "Alain Dupont", 24};

        printf("Champ nom : %s, %s.\n", rec.name, rec1.name);
        printf("Code : %d, %d.\n", rec.key, rec1.key);
        return 0;
} /* fin de main */
```

Tableaux de Structures

Les éléments de tableaux peuvent être des structures :

```
#include <stdio.h>
#define MAXITEMS 3
#define ITEM 20

int main(void)
{
     int index;
     float total;
     struct info {
        char item[ITEM];
        float cost;
        int count;
     } stock[MAXITEMS];

     for (index = 0; index < MAXITEMS; index++) {
        printf("Entrer le nom numero %d: ", index + 1);
        scanf("%s", stock[index].item);
        printf("Entrer son coût : ");
        scanf("%f", &stock[index].cost);
        printf("Entrer la quantité commandée : ");
        scanf("%d", &stock[index].count);
        total = stock[index].cost * stock[index].count;
        printf("Coût total pour %s: %.2f F\n\n",
            stock[index].item, total);
     } /* fin de for */

     return 0;
} /* fin de main */
```

Initialisation des Tableaux de Structures

Attention à l'ordre des données dans l'initialisation des tableaux de structures :

```
#include <stdio.h>
int main(void)
{
        struct database {
            char name[40];
            int key;
        };

        struct database list[3] = { "Marc Dupas", 42,
                                    "Michel Dufour", 256,
                                    "Henri Dupont", 0 };

        printf("Premier nom = %s.\n", list[0].name);
        printf("Le 3eme entier = %d.\n", list[2].key);
        return 0;
} /* fin de main */
```

Pointeurs de Structures

- On peut déclarer un pointeur vers un type structure.
- Les membres d'une structure sont référencés à travers un pointeur par l'opérateur `->` :

```
#include <stdio.h>
int main(void)
{
        struct employee {
            char first_name[40];
            char last_name[40];
            int age;
            float salary;
        };
        struct employee person[5], *emptr;
        int i;

        emptr = person;
        for (i = 0; i < 5; ++i, ++emptr) {
            printf("Nom et prénom de l'employé : ");
            scanf("%s%s", emptr->first_name, emptr->last_name);

            printf("Age de l'employé : ");
            scanf("%d", &emptr->age);

            printf("Salaire : ");
            scanf("%f", &emptr->salary);

            while (getchar() != '\n');/* raz du clavier */
        }
        return 0;
} /* fin de main */
```

Révision Partielle

Structure

Soient les déclarations suivantes, donner l'instruction satisfaisant à chaque question :

```
struct inventory {
        char model[20];
        float cost;
        int count;
} cars[50];
struct inventory *mycar = &cars[0];
```

1) Copier *jaguar* dans le membre *model* de la 5ème voiture du tableau *cars*.

_____

2) Copier *jeep* dans le membre *model* de *mycar*.

_____

3) Lire un coût dans le membre *cost* de la 3ème voiture du tableau *cars*.

_____

4) Afficher le coût du dernier élément du tableau.

_____

5) Affecter 75000 F au coût de *mycar*.

_____

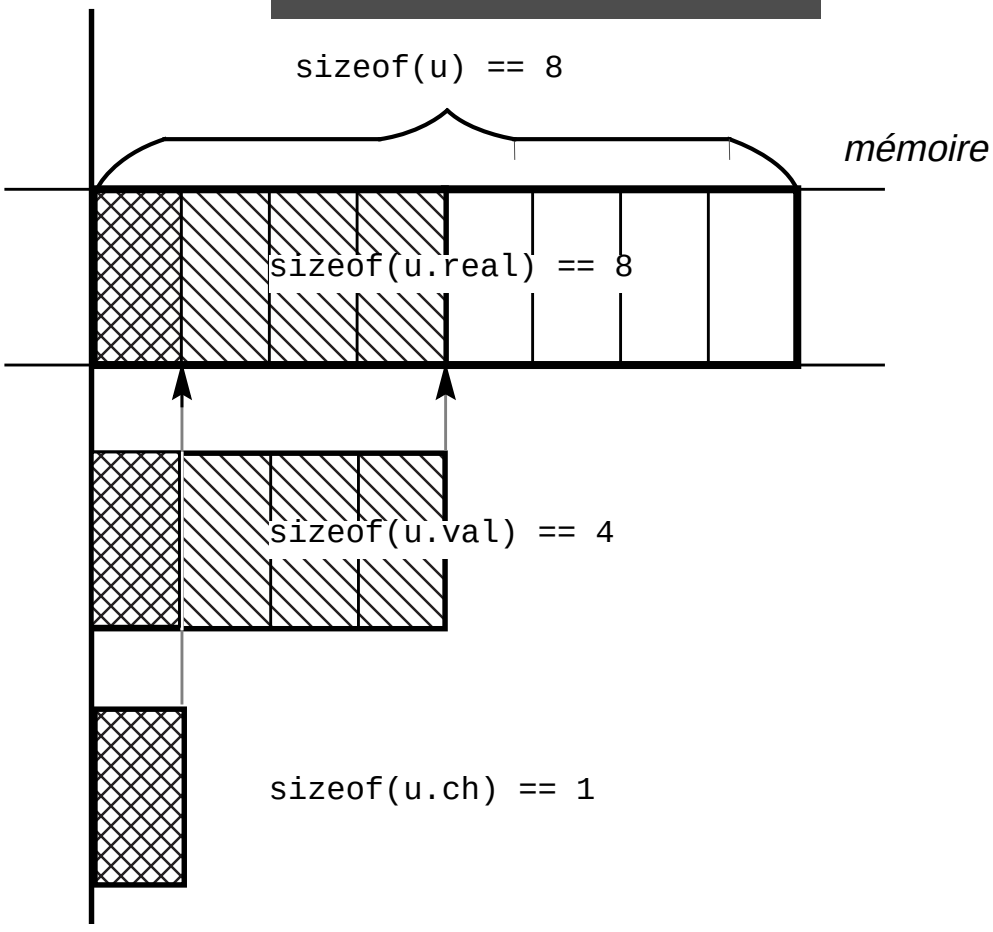
6) Saisir le nom du modèle de la 8ème voiture du tableau.

_____

Introduction aux Unions

- Une `union` est un espace mémoire partagé par plusieurs objets, généralement de types *différent*s.
- L'information stockée dans une `union` est vue du même *type* que le membre référencé, et la référence est faite de la *même* manière que pour un membre de *structure*.
- Une union est *seulement* aussi grande que son plus grand élément :

```
union hypothetical {
        double real;
        int val;
        char ch;
}u;
```



Introduction aux Enumérations

Format

enum *<id-type>* {*membre1,…, membre_n*} *<variables>;*
ou
enum *<id-type> {membre1, ..., membre_n};*
enum *<id-type> <vars>;*

- Un *type énumérés* ou *énumération* est défini par le mot-clef enum.
- Une *énumération* est un ensemble nommé de constantes entières.
- *Les Enumérations* peuvent être employées pour améliorer la lisibilité et la clarté d'un programme :

```
#include <stdio.h>
int main(void)
{
        enum fruit {pomme = 1, orange, poire, kiwi, raisin};
        enum fruit fruit_choice;

        printf("Entrer votre choix:\t1 - pomme\n\t\t\t2 - orange\n"
            "\t\t\t3 - poire\n\t\t\t4 - kiwi\n\t\t\t5 - raisin\n"
            "Choix :______\b\b\b");
        scanf("%d", &fruit_choice);
        switch(fruit_choice) {
            case pomme:
                printf("Votre choix correspond à une pomme.\n");
                break;
            case orange:
                printf("Votre choix correspond à une orange.\n");
                break;
            /* ... suites des case */
        }  /* fin de switch */
        return 0;
} /* fin de main */
```

Introduction à `typedef`

Format

`typedef` *<type_existant> <nouveau_type>*;

`typedef` permet de donner un nouveau nom (créer un synonyme) pour un type de donnée déjà existant :

```
#include <stdio.h>
#define MAX 80
int main(void)
{
        typedef char String[MAX];
        typedef union {
            int word32;
            short word16[2];
            char bytes[4];
        } Mask;
        typedef struct {
            int num;
            String text;
            Mask flags;
        } Record;
        Record rec1, rec2;/* <- déclaration simplifiée ! */

        /* déroulement simulé... */
        printf("Entrer une chaîne :  ");
        gets(rec1.text);
        printf("Entrer votre tour de tête :  ");
        scanf("%d", &(rec1.flags.word32));
        printf("Votre QI est %d.\n", rec1.flags.word32 * 2);
        return 0;
} /* fin de main */
```

Révision de Module

Structures

Q. Quelle est la principale propriété qui rend une structure différente d'un tableau ?

R. _____

Q. Quelles sont les types qui peuvent être membres de `struct` ?

R. _____

Q. Combien de niveaux d'imbrication de structure sont permis ?

R. _____

Q. Comment déterminer la taille d'une structure ?

R. _____

Q. Qu'est-ce que l'identifiant de structure et quelle est son importance ?

R. _____

Soient les déclarations suivantes, décrire les structures et la manière d'accéder à leurs membres :

```
struct record{
        int data;
        char name[16];
} rec1;
```

```
struct record rec2[2] = {128, "chaîne1", 256, "chaîne2"};
```

```
struct record *rptr = &rec1;
```

Structures

Ecrire un programme qui déclare une structure à 3 membres : une chaîne de 40 caractères nommée *data*, un entier *key* et un double *dub.* Afficher la taille de la structure, faire des affectations de valeur à chaque membre, et afficher chaque membre :

Conseil : vous pouvez utiliser *strcpy()* pour charger une valeur dans *data*.

Travaux Pratiques 10 : Structures

Présentation

Introduction à l'utilisation des *structure*s, *union*s, *typedef*s, et *enum*s dans un programme C.

Exercices

****** - La solution sera utilisée dans les TP à venir.

1. **Niveau 1**. *Vérifier* les résultats du programme de révision en le compilant et en le lançant.

2. ****Niveau 2**. Reprendre le programme `ages.c` des TP 2. Le programme doit déclarer une structure (*avec identifiant de structure*) comportant les membres suivants :

```
char name[20]
int birth_year;
short age;
short sum;
short product;
```

   Le programme doit demander la valeur de chaque champ, et stocker la valeur dans une structure après avoir vérifié sa validité. Une fois la structure entièrement renseignée, proposer un menu pour le choix d'un champ. Après choix de l'utilisateur, afficher le contenu du champ demandé.

- voir page suivante -

3. ****Niveau 3**. Ecrire un programme pour :

   Déclarer une structure avec les membres suivants :

   ```
   char first[20]
   char last[20]
   short age;
   ```

   Déclarer un tableau de 4 éléments du type structure.

   Définir une fonction *`print_strux()`* qui affiche une structure comme ci-dessus.

   Dans une boucle, demander à l'utilisateur des valeurs pour les 4 structures du tableau. Saisir l'âge comme une chaîne de caractère et la convertir avec *`atoi()`*.

   Faire une fonction qui affiche une des structures à partir d'un indice utilisateur (c'est-à-dire entre 1 et 4 et non entre 0 et 3).

   Facultatif : Faire une boucle autour de la dernière fonction d'impression pour que l'utilisateur puisse visualiser autant de structures qu'il veut et non pas une seule.

   Nommer le fichier source `strux.c`.

   Conseil : Définir la structure avant la fonction *`main()`*.

Opérateurs sur Bits

Objectifs

- Utiliser l’opérateur sur bits "&" pour masquer des bits à 0 dans un entier.
- Utiliser l’opérateur sur bits "|" pour positionner des bits à 1 dans un entier.
- Utiliser les opérateurs de décalage de bits sur des entiers.
- Manipuler individuellement les bits d’un entier avec les champs de bits des structures.

Evaluation

Travaux Pratiques 11 et révision de module.

Tables de Vérité Logiques.

- Les tables de vérité : *P* et *Q* représentent des bits :

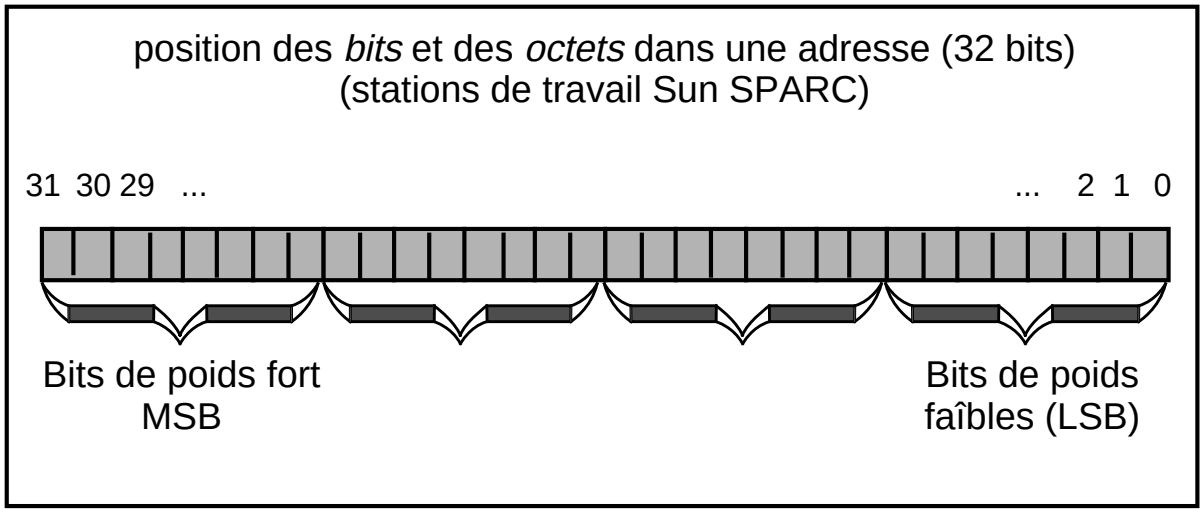
P	Q	P&Q	P|Q	P^Q	~P	~Q
0	0	0	0	0	1	1
1	0	0	1	1	0	1
0	1	0	1	1	1	0
1	1	1	1	0	0	0

& . . . . . *et* bit à bit (AND)
|. . . . . *ou* bit à bit (OR)
^. . . . . *ou exclusif* bit à bit (XOR)
~ . . . . . *non* bit à bit unaire : inversion de chaque bit (NOT)

- "&" donne le bit 1 si et seulement si les deux bits opérandes sont à 1, et 0 dans les autres configurations.
- "|" donne le bit 0 si et seulement si les deux bits opérandes sont à 0, et 1 dans les autres cas.
- "^" donne le bit 1 si et seulement si un seul des deux bits est à 1, sinon 0.
- "~" donne le complément à 1 de chaque bit, autrement dit si le bit est 1 le résultat est 0, si le bit est 0 le résultat est 1.

Définitions Diverses

- *least significant bit* (LSB)- (*bit de poids faîble*) le bit le plus à droite dans l'octet ou le champ de bits.
- *most significant bit* (MSB)- (*bit de poids fort)* le bit le plus à gauche dans l'octet ou le champ de bits.
- *masque* - configuration de bits utilisée pour remettre à zéro (raz) ou forcer à 1 certains bits dans un octet ou un champ de bit, en vue de modification ou de test.
- *masquer*- utiliser une configuration de bits pour retenir ou éliminer certains bits d'un octet.



Opérateurs sur Bits - *et &*

Vue de détail sur l'opérateur logique sur bits *et &* - forcer à 0 les 3 octets de poids fort :

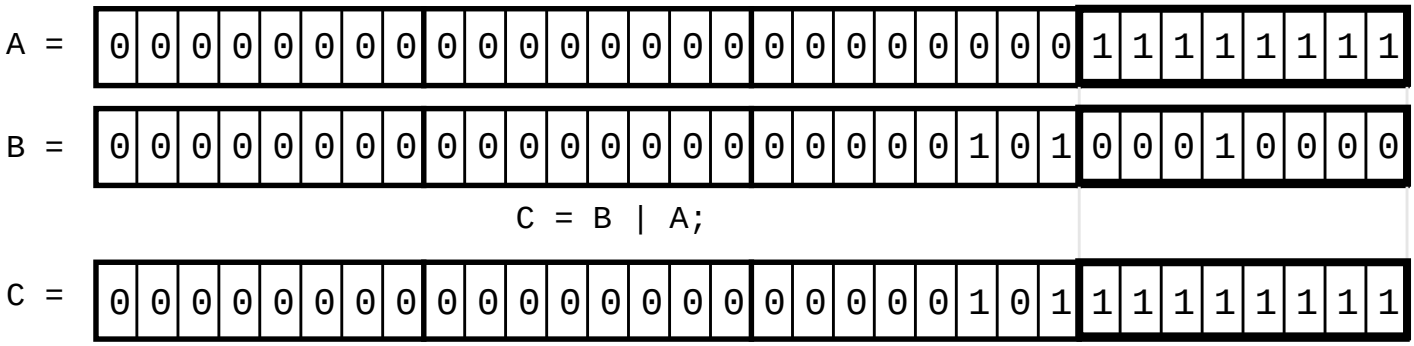
```
                unsigned int A = 0xFF, B = 0x339ddb59, C;

A =  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
B =  0 0 1 1 0 0 1 1 1 0 0 1 1 1 0 1 1 1 0 1 1 0 1 1 0 1 0 1 1 0 0 1
                C = B & A;
C =  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 1
```

```
#include <stdio.h>
#define READY 0x1       /* périphérique prêt */
#define RESET 0x2       /* périphérique initialisé */
#define DATA 0x4        /* données dispo sur périphérique */
#define BROKEN 0x8      /* périphérique coupé */
int main(void)
{
        unsigned char devstatus(char devname[]);
        unsigned char status;
        status = devstatus("/dev/printer_one");
        if (status & READY)
            printf("le bit READY est présent dans status.\n");
        if (status & RESET)
            printf("le bit RESET est présent dans le status.\n");
        /* tests similaires pour DATA et BROKEN */
}
unsigned char devstatus(char devname[])
{
        /* retourner la configuration 5 == '0101'
            bit 0 == 1 READY
            bit 1 == 0 pas de RESET
            bit 2 == 1 DATA (données disponibles)
            bit 3 == 0 pas de bit BROKEN (ligne ok)
        /*
        return 0x5;
} /* fin de main */
```

Opérateurs sur Bits - *ou* |

Démonstration de ou logique sur bit, pour positionner à 1 l'octet de poids faible d'un entier :

```
                    unsigned int A = 0xFF, B=0x510, C;
```




```
#include <stdio.h>
#define PRIZE1 0x001
#define PRIZE2 0x100
int main(void)
{
        int some_function(int val), num, bflags = 0;
        printf("Entrer un entier positif : ");
        scanf("%d", &num);
        bflags = some_function(num);
        if (bflags & PRIZE1)
            printf("Vous gagnez le premier prix !\n");
        if (bflags & PRIZE2)
            printf("Vous gagnez le deuxième prix !\n");
} /* fin de main */

int some_function(int val)
{
        if (val < 256)      /* 256=0x100 */
            return (val | PRIZE1);
        else if (val >= 256)
            return(val | PRIZE2);
        else
         return 0;
} /* fin de some_function */
```

Opérateurs sur Bits - *xor* et *complément à un*

Voyons de plus près l'opérateur logique sur bits *xor* et *le complément à un* :

```
unsigned int A = 0xFF, B = 0x339ddb59, C;
```

A = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

B = 0 0 1 1 0 0 1 1 1 0 0 1 1 1 0 1 1 1 0 1 1 0 1 1 0 1 0 1 1 0 0 1

```
C = B ^ A;
```

C = 0 0 1 1 0 0 1 1 1 0 0 1 1 1 0 1 1 1 0 1 1 0 1 1 1 0 1 0 0 1 1 0

```
C = ~C;
```

C = 1 1 0 0 1 1 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 1 1 0 0 1

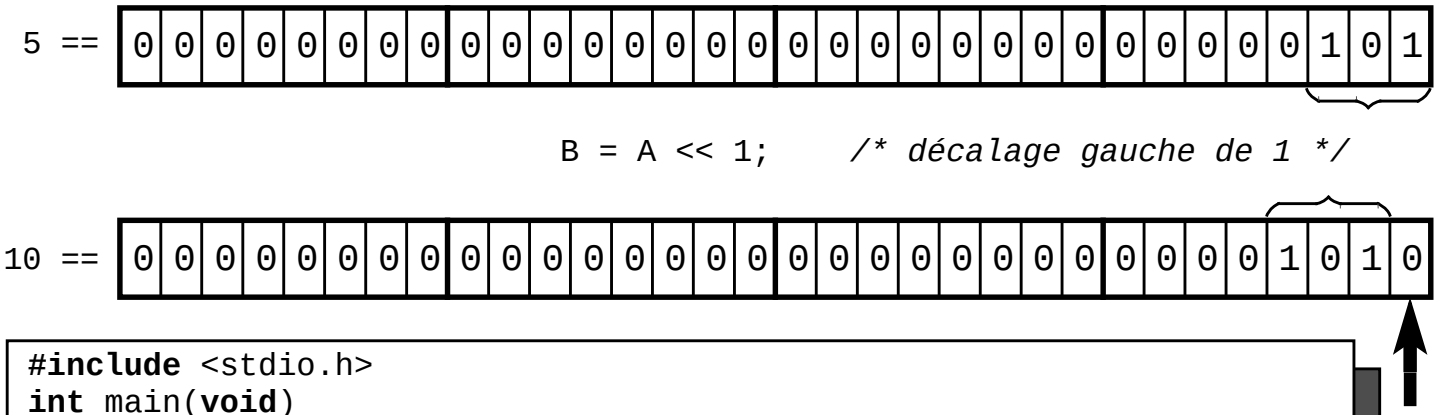
Opérateurs sur Bits - ">> et <<" Décalage

Format

```
variable << nombre_de_positions
variable >> nombre_de_positions
```

Exemples d'utilisation d'opérateur décalage à gauche et à droite "<<" et ">>" - pour un décalage à gauche de 1 bit :

```
unsigned int A = 5, B;
```




```
#include <stdio.h>
int main(void)
{
        int i, num;
        unsigned int mask;
        printf("Entrer un entier en base 10 : ");
        scanf("%d", &num);

        printf("Octal:\t0%o\n", num);
        printf("Hexa:\t0x%x\n", num);
        printf("Binaire:\t0b");
        mask = 1 << ( sizeof(int) * 8 - 1 ) ;

        for (i = sizeof(int) << 3 ; i > 0; i-- , mask >>= 1 ) {
            putchar(num & mask ? '1': '0');
        }
        putchar('\n');
        return 0;
}  /* fin de main */
```

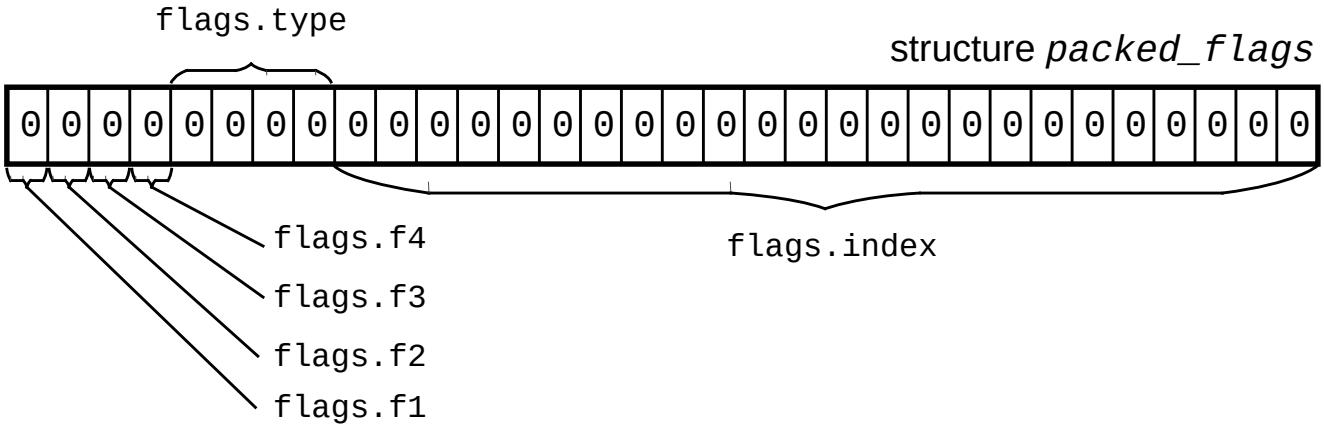
Retour sur les Structures - Les Champs de Bits

Format

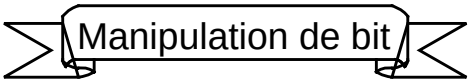
```
struct <id_de_structure> {
   <type_d'int> <identifiant>:<nb de bits>
   ...
} <id_de_variable>;
```

Les champs de bits des structures fournissent un moyen commode d'accéder individuellement aux bits, et permet une utilisation plus efficace de la mémoire - les indicateurs binaires (flags ou drapeaux) peuvent être regroupés dans des entiers :

```
struct packed_flags {
        unsigned int f1:1;
        unsigned int f2:1;
        unsigned int f3:1;
        unsigned int f4:1;
        unsigned int type:4;
        unsigned int index:24;
} flags;
```



Révision de Module



Q. Quel opérateur met les bits/octets à 0 ? Lequel les met à 1 ?

R._____ _____

Q. Décrire l'effet de l'opérateur sur bits *non-unaire* (complément à un) :

R. _____

Q. Décrire les effets des opérateurs de décalage à droite >> et à gauche << :

R. _____

Ecrire une déclaration qui fasse l' `union` d'un `unsigned int` et d'une `struct` comprenant un tampon de 3 octets suivi de 8 *champs d'1 bit* (`unsigned int`) .

Travaux Pratiques 11 : Manipulation de Bit

Présentation

Introduction aux opérateurs sur bits et aux champs de bits des structures du C.

Exercices

1. **Niveau 1**. Ecrire un programme qui, à partir des déclarations de la révision du module, affecte des 1 à chaque champ de bit (dans le sens décroissant : 8-1). Afficher la taille de la structure et de l'union, puis la valeur de chaque champ et du mot entier à chaque affectation (en hexa et/ou décimal).

2. **Niveau 2**. Ecrire un programme pour :

   `#define` un masque d'un octet pour l'octet de poids faîble et un pour le second octet de poids faîble (LSB).

   Déclarer un `short int`.

   Positionner et tester à l'aide des masques, les deux octets de l'entier court.

   Facultatif : (**Niveau 3**)
   Ecrire une fonction dans le programme qui positionne le bit 0 ou le bit 8 à 1 selon que la première lettre du nom de l'utilisateur est respectivement dans la première ou la deuxième partie de l'alphabet. La fonction `main` doit afficher un message différent selon le bit positionné par la fonction.

   Nommer le fichier source `masks.c`

Passage d'Arguments à main()

Objectifs

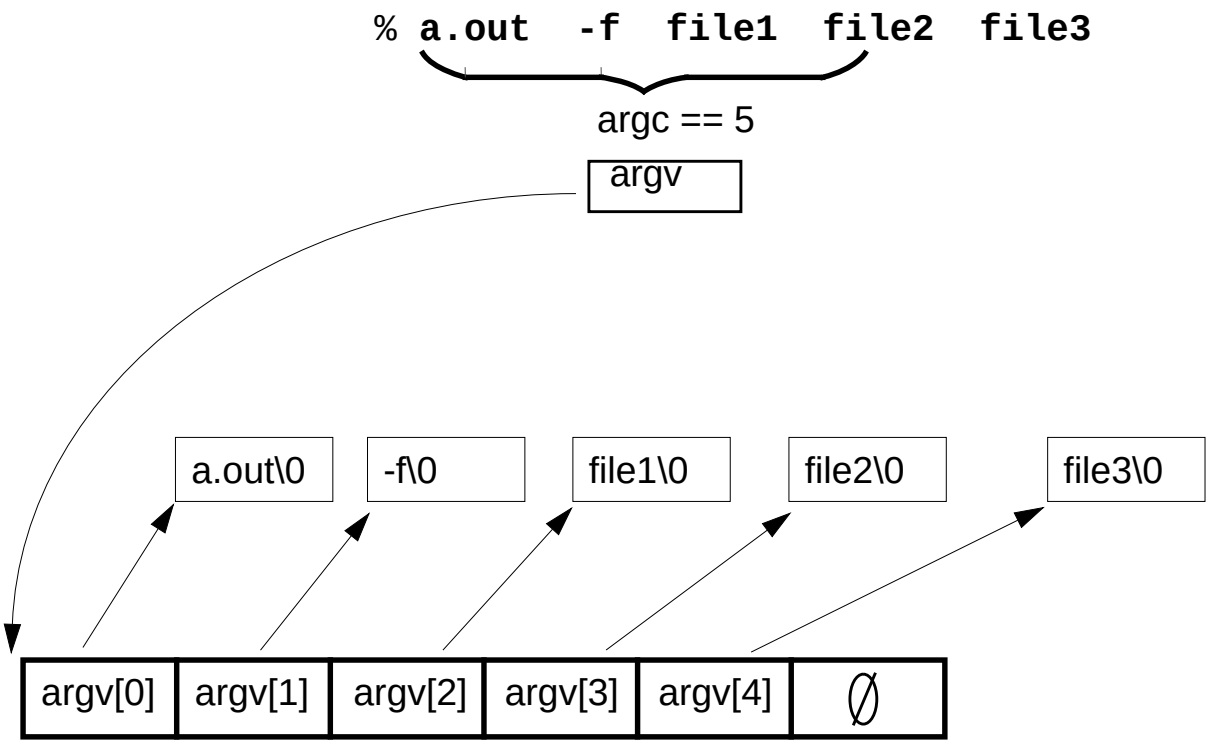
- Retrouver et traiter des arguments de la ligne de commande dans un programme C.

Evaluation

Travaux Pratiques 12 et révision de module.

Récupérer les Arguments de la Ligne de Commande

- Le compte d'argument est dans la variable *argc*.
- L'argument *argv* pointe vers un tableau de pointeurs sur `char` (les chaînes de caractères des arguments) , dont le dernier est le pointeur *NULL*.



Récupérer les Arguments de la Ligne de Commande

Exemple de traitement des arguments de main avec une notation indicée :

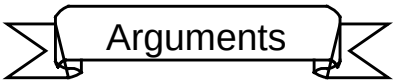
```
#include <stdio.h>
int main (int argc, char *argv[])
{
       int index;

       if (argc > 1) {
           printf("%d arguments:\n", argc - 1);
           for (index=1; index<argc; index++)
               printf("%s\n",argv[index]);
       }
       else
           printf("Pas d'arguments.\n");
       return 0;
} /* fin de main */
```

Récupérer les Arguments de la Ligne de Commande

Autre exemple utilisant la notation pointeur :

```
#include <stdio.h>
#define MIN_ARGS 2    /* 1 argument minimum */

int main (int argc, char **argv)
{
       int index = 0;
       char **temp = argv;  /* sauvegarde de argv */
       void  usage(char *);/* déclaration de la fonction usage */

       if (argc < MIN_ARGS)/*1 argument au moins, quoi ! */
          usage(*argv);
       while (*temp)    /* tant que différent de NULL... */
          /* affiche tous les arguments... */
          printf("L'argument #%d est \"%s\"\n",index++, *temp++);
       return 0;
} /* fin de main */

void usage(char *prog_name)
{
       printf("\nUsage:\n");
       printf("\t%s <argument> [<argument>...]\n\n", prog_name);
       exit(1); /* fin du programme après le message */
} /* fin de usage */
```

Révision de Module



Q. Quel est le nom conventionnel des arguments de main() et leur ordre ?

R. _____

Q. Quelles sont les deux manières de déclarer les arguments de main ?

R. _____

En utilisant la notation indicée, écrire un programme `add` qui additionne 2 entiers passés en paramètre suivant la syntaxe :

`add <n1> <n2>`.

Le programme doit vérifier qu'on lui passe 3 arguments et afficher un message d'aide sinon.

Travaux Pratiques 12 : Arguments de la Ligne de Commande

Présentation

Familiarisation avec la déclaration et l'utilisation des arguments de la ligne de commande dans un programme C.

Exercices

1. **Niveau 1**. *Vérifier* les résultats du programme de révision en le compilant et en l'exécutant.

2. **Niveau 2**. Ecrire un programme pour :

   Utiliser un pointeur pour argv et un indice pour les chaînes.

   Afficher les arguments caractères par caractères avec *`putchar()`*.

   Si pas d'argument, afficher un message et sortir.

3. **Niveau 3**. (*facultatif*) Le programme doit prendre 2 arguments. Comparer le 1er avec `-n` (*normal*), ou `-i` (*inverse*). Si le 1er argument n'est aucun des 2 précédents, donner un message d'usage et sortir. Si l'option est -n afficher le 2eme argument normalement, sinon l'afficher en inversant l'ordre des lettres, caractère par caractère avec *`putchar()`*.Une fonction peut gérer l'affichage de l'argument. Ne pas coder en dur le nom du programme dans le message d'usage, mais utiliser plutôt *`argv[0]`*.

   Appeler le programme `argmanip.c`

   Conseil : Le message d'usage ressemblera à :

   Usage :
   `argmanip [-f | -r] <chaîne_de_caractère>`

   Assurez vous de bien inclure `<string.h>`.

Entrées/Sorties Fichiers Standard

Objectifs

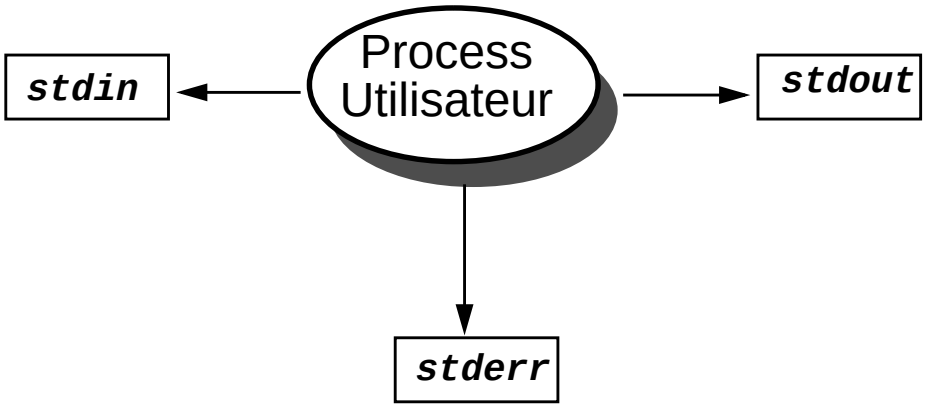
- Réaliser des E/S (Entrées/Sorties) en utilisant la librairie standard.
- Ouvrir un fichier avec *`fopen()`*.
- Lire dans un fichier avec *`fread()`*, *`fgets()`*, *`fgetc()`* et *`fscanf()`*.
- Ecrire dans un fichier avec *`fwrite()`*, *`fputs()`*, *`fputc()`* et *`fprintf()`*.
- Positionner le pointeur de lecture/écriture avec *`fseek()`*.
- Récupérer la position courante avec *`ftell()`*.
- Fermer un fichier avec *`fclose()`*.
- Vider un tampon de sortie avec *`fflush()`*.

Evaluation

Travaux Pratiques 13 et révisions de module.

Définition des Entrées/Sorties niveau User

- *FILE* est la structure de contrôle des fichiers.
- Les objets *stdin*, *stdout* et *stderr*, du type *FILE* *, peuvent être utilisés comme arguments des fonctions standard d'E/S fichiers (à voir plus loin) :



stdin	"entrée standard"
stdout	"sortie standard"
stderr	"sortie erreur standard"

Ouvrir un fichier avec *fopen()*

Format

```
#include <stdio.h>
FILE *fopen(const char *nomfic, const char *mode)
```

- Les routines de lecture, écriture, déplacement, etc, sur fichiers, sont fournies dans la librairie standard du C (voir `man -s 3s function`)
- *fopen()* renvoie un pointeur sur une structure de contrôle *FILE* qui mémorise les informations sur le fichier ouvert. Ce pointeur est souvent stocké dans une variable nommée *fp*.
- *fopen()* renvoie le pointeur *NULL* sur erreur d'ouverture
- Le *mode* fourni à *fopen()* indique comment se fera l'accès au fichier après l'ouverture :

"**r**"		fichier texte pour lecture.
"**w**"		vider ou créer un fichier texte pour écriture.
"**a**"		append ; ouvrir ou créer un fichier texte pour écriture en fin de fichier
"**rb**"		fichier binaire pour lecture.
"**wb**"		vider ou créer un fichier binaire pour écriture.
"**ab**"		ouvrir ou créer un fichier binaire pour écriture en fin de fichier.
"**r+**"		ouvrir un fichier texte en lecture/écriture.
"**w+**"		vider ou créer un fichier texte en lecture/écriture.
"**a+**"		ouvrir ou créer un fichier texte en lecture/écriture, pointeur en fin de fichier.
"**rb+**"	ou "**r+b**"	ouvrir un fichier binaire en lecture/écriture.
"**wb+**"	ou "**w+b**"	vider ou créer un fichier binaire en lecture/écriture
"**ab+**"	ou "**a+b**"	ouvrir ou créer un fichier binaire en lecture/écriture pointeur d'écriture en fin de fichier.

Fermer un Fichier avec *fclose()*

Format

```
#include <stdio.h>
int fclose(FILE *fp);
```

- *fclose()* finit d'écrire toutes les donnée bufferisées et ferme le flux (*stream)* (une source ou une destination de données) associé à *fp*.
- *fclose()* renvoie 0 en cas de réussite, et *EOF* sur erreur.
- *fclose()* est implicite si *exit()* est appellée ou si le programme se termine. Il vaut tout de même mieux fermer explicitement les fichiers avant la fin d'un programme.

```
#include <stdio.h>
int main(void)
{
        FILE *fp;

        if ((fp = fopen("/etc/passwd", "r")) == NULL) {
            printf("Ouverture impossible /etc/passwd.\n");
            exit(1);
        } /* fin de if */

        /* traitement des données du fichier... */

        fclose(fp);/* fermer le fichier */
        return 0;
} /* fin de la fonction main */
```

E/S Formatées et Pointeurs de Fichiers

Format

```
#include <stdio.h>
int fscanf(FILE *fp, const char *format, ...);
```

- *fscanf()* est similaire à *scanf()*, sauf que les données sont lues à partir du fichier associé à *fp* plutôt que sur l'entrée standard.
- En cas de succés, *fscanf()* retourne le nombre d'items correctement saisis. Si une erreur survient avant toute conversion, alors *fscanf()* renvoie *EOF*.

Format

```
#include <stdio.h>
int fprintf(FILE *fp, const char *format, ...);
```

- *fprintf()* est similaire à *printf()*, sauf que la sortie se fait sur le fichier pointé par *fp* plutôt que sur la sortie standard.
- Normalement, *fprintf()* renvoie le nombre de caractères écrits. Sur erreur d'écriture, une valeur négative est retournée.

Lecture Simple sur un Fichier

Format

```
#include <stdio.h>
int fgetc(FILE *fp);
```

- Sans erreur, *fgetc()* renvoie le prochain caractère à lire sur le fichier pointé par *fp*. Sur fin-de-fichier ou sur erreur de lecture, *fgetc()* renvoie *EOF*.
- Programme démontrant l'utilisation de *fopen()*pour l'ouverture d'un fichier, de la fonction d'entrée *fgetc()*, et de la fonction de sortie *fprintf()*:

```
#include <stddef.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
        FILE *fp;
        int   ch;
        void usage(char *);

        if (argc != 2)
            usage(argv[0]);
        if ((fp = fopen(argv[1], "r")) == NULL) { /*afficher sur stderr
*/
            fprintf(stderr, "Erreur d'ouverture de %s\n", argv[1]);
            exit(1);
        } /* fin de if */
        while((ch = fgetc(fp)) != EOF)/*lecture car/car sur le fichier */
            putchar(ch);    /* sortie sur stdout */
        fclose(fp);         /* fermeture */
        return 0;
} /* fin de main */

void usage(char *prog_name)
{
        fprintf(stderr, "\nUsage:\n");
        fprintf(stderr, "\t%s <chemin>\n\n", prog_name);
        exit(1);
```

Lecture Simple sur Fichier

Format

```
#include <stdio.h>
char *fgets(char *s, int n, FILE *fp);
```

- En cas de réussite, *fgets()* renvoie *s*. Si *la fin de fichier* est rencontrée ou qu'une erreur de lecture survient, le pointeur *NULL* est renvoyé.
- Exemple de programme illustrant l'usage de *fopen()* et de *fgets()*:

```
#include <stddef.h>
#include <stdio.h>
#define BUFSIZE 256

int main(int argc, char *argv[])
{
     FILE *fp;
     char str[BUFSIZE];
     void usage(char *prog_name);

     if (argc != 2) usage(argv[0]);
     if ((fp = fopen(argv[1], "r")) == NULL) {
       fprintf(stderr, "Erreur d'ouverture de %s\n", argv[1]);
       exit(1);
     } /* fin de if */
     while (fgets(str, BUFSIZE, fp) != NULL)   /*lecture/fichier */
       printf("%s", str);/* affichage sur stdout */
     fclose(fp);    /* fermeture */
     return 0;
} /* fin de main */

void usage(char *prog_name)
{
     fprintf(stderr, "\nUsage:\n");
     fprintf(stderr, "\t%s <chemin>\n\n", prog_name);
     exit(1);
} /* fin de usage */
```

Ecriture Simple sur Fichier

Format

```
#include <stdio.h>
int fputc(int c, FILE *fp);
```

- *fputc()* renvoie le caractère écrit. Sur erreur, c'est *EOF* qui est retournée.
- Exemple de programme pour l'utilisation de *fopen()*, *fgetc()* et *fputc()*:

```
#include <stddef.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
     FILE *fpin, *fpout;
     int ch;
     void usage(char *), err(char *);
     if (argc != 3)
       usage(argv[0]);
     if ((fpin = fopen(argv[1], "r")) == NULL)
       err(argv[1]);
     if ((fpout = fopen(argv[2], "a+")) == NULL)
       err(argv[2]);
     while ((ch = fgetc(fpin)) != EOF)
       fputc(ch, fpout);
     fclose(fpin);  /* fermeture des fichiers */
     fclose(fpout);
     return 0;
} /* fin de main */
void usage(char *prog_name)
{
     fprintf(stderr, "\nUsage:\n");
     fprintf(stderr, "\t%s <f_entree> <f_sortie>\n\n", prog_name);
     exit(1);
} /* fin de usage */
void err(char *file)
{
     fprintf(stderr, "Ouverture impossible de %s\n", file);
     exit(1);
} /* fin de err */
```

Ecriture Simple sur Fichier

Format

```
#include <stdio.h>
int fputs(const char *s, FILE *fp);
```

- En cas de succès, *fputs()* renvoie une valeur non-négative. La fonction renvoie *EOF* en cas d'erreur.
- Exemple de programme utilisant *fopen()*, *fgets()* en entrée, et *fputs()* en sortie :

```
#include <stddef.h>
#include <stdio.h>
#define BUFSIZE 256
int main(int argc, char *argv[])
{
        FILE *fpin, *fpout;
        char  str[BUFSIZE];
        void usage(char *), err(char *);
        if (argc != 3)
            usage(argv[0]);
        if ((fpin = fopen(argv[1], "r")) == NULL)
            err(argv[1]);
        if ((fpout = fopen(argv[2], "a+")) == NULL)
            err(argv[2]);
        while (fgets(str, BUFSIZE, fpin) != NULL)
            fputs(str, fpout);
        fclose(fpin);     /* fermeture des fichiers */
        fclose(fpout);
        return 0;
} /* fin de main */
void usage(char *prog_name)
{
        fprintf(stderr, "\nUsage:\n");
        fprintf(stderr, "\t%s <f_entree> <f_sortie>\n\n", prog_name);
        exit(1);
} /* fin de usage */
void err(char *file)
{
        fprintf(stderr, "Erreur d'ouverture de %s\n", file);
        exit(1);
} /* fin de err */
```

Lecture de Données depuis un Fichier.

Format

```
#include <stdio.h>
size_t fread(void *ptr,size_t taille,size_t nitems,FILE *fp);
```

- *fread()* peut lire de grandes quantités de données, comme indiqué par *taille*. Cette fonction est utilisée pour des fichiers binaires dont on connaît la taille des enregistrements, créés avec *fwrite()*.
- *fread()* renvoie le nombre d'items lus. Ce nombre est différent de *nitems* sur erreur ou fin de fichier (*EOF*).
- Exemple de l'utilisation de *fopen()*, et de la fonction d'entrée fread*()*:

```
#include <stddef.h>
#include <stdio.h>
#define ITEMSIZE 25
#define INVCOUNT 3
typedef struct {
            char item[ITEMSIZE];
            float cost;
} Data;
int main(void)
{
        FILE *indev;
        Data inventory[INVCOUNT];
        int c, itemsread;
        if ((indev = fopen("inv.dat", "r")) == NULL){
            fprintf(stderr, "Erreur d'ouv. de 'inv.dat'en lecture\n");
            exit(1);
        }
        itemsread = fread(inventory, sizeof(Data), INVCOUNT, indev);
        fclose(indev);
        for (c = 0; c < itemsread; c++){
            printf("Item %2d:  %25s - %.2f F\n", c, inventory[c].item,
                    inventory[c].cost);
        }
        return 0;
} /* fin de main */
```

Ecriture dans un Fichier - *fwrite()*

Format

```
#include <stdio.h>
size_t fwrite(const void *ptr,size_t size,size_t nitems,FILE *fp);
```

- *fwrite()* renvoie le nombre d'éléments écrits. Ce nombre est inférieur à *nitems* en cas d'erreur.
- Exemple de programme utilisant *fopen()*, et *fwrite()* en sortie :

```
#include <stddef.h>
#include <stdio.h>
#define ITEMSIZE 25
#define INVCOUNT 3

typedef struct {
     char item[ITEMSIZE];
     float cost;
} Data;
int main(void)
{
     FILE *outdev;
     static Data inventory[INVCOUNT] = {
        "crayon",  0.15,
        "stylo",    0.49,
        "surligneur", 1.25
     };

     if ((outdev = fopen("inv.dat", "w")) == NULL){
        fprintf(stderr, "Ouv. impossile de 'inv.dat' en écriture\n");
        exit(1);
     } /* fin de if */
     fwrite(inventory, sizeof(Data), INVCOUNT, outdev);
     fclose(outdev);
     return 0;
}  /* fin de main */
```

Trouver la Position Courante du Pointeur d'un Fichier

Format

```
#include <stdio.h>
long ftell(FILE *fp);
```

- Le tampon pointé au travers de *fp* contient un pointeur relatif au début du fichier, sur le prochain caractère lu ou écrit dans le fichier. Au fur et à mesure des lectures et/ou écritures, ce pointeur de fichier est déplacé en avant dans le fichier.
- La fonction *ftell()* renvoie la position courante en octets dans le fichier représenté par *fp*. En cas d'erreur, *ftell()* renvoie *-1L*.

```
#include <stddef.h>
#include <stdio.h>
int main(void)
{
        FILE *fp;
        long pos;

        ... /* autres déclarations ... */

        if ((fp = fopen(argv[1], "r")) == NULL){
            fprintf(stderr, "fopen raté\n");
            exit(1);
        }

        ... /* après quelques lectures dans le fichier */

        pos = ftell(fp);

        ... /* suite du programme */
}  /* fin de main */
```

Déplacer le Pointeur de Fichier

Format

```
#include <stdio.h>
void rewind(FILE *fp);
```

- Le pointeur de fichier peut être remis au début avec *rewind()*. *rewind()* ne renvoie aucune valeur :

```
#include <stddef.h>
#include <stdio.h>
#define MAX 4
#define BUFSIZE 40
int main(int argc, char *argv[])
{
     FILE *fp;
     struct db_record {
       char name[BUFSIZE];
       short age;
     } rex[MAX], rec;
     int size = sizeof(rec), i;
     char temp[BUFSIZE];
     void usage(char *), err(char *); /*fonctions définies ailleurs*/
     if (argc != 2)
       usage(argv[0]);
     if ((fp = fopen(argv[1], "a+")) == NULL)
       err(argv[1]);
     for (i = 0; i < MAX; i++) {
       printf("Entrer le nom numéro %d: ", i + 1);
       fgets(rex[i].name, BUFSIZE, stdin);
       printf("Entrer l'age du %s :  ", rex[i].name);
       rex[i].age = (short) atoi(fgets(temp, BUFSIZE, stdin));
       if (!(fwrite(&rex[i], size, 1, fp)))
          err(argv[1]);
     } /* fin du for */
     rewind(fp);/* pointeur de fichier au début  */
     for (i = 0; i < MAX; i++) {
       if (!(fread(&rec, size, 1, fp)))
          err(argv[1]);
       printf("Nom n°%d: %s, age: %d\n",i+1,  rec.name, rec.age);
     } /* fin de for */
     fclose (fp);/* fermer le fichier */
     return 0;
} /* fin de main */
```

Déplacer le Pointeur de Fichier

Format

```
#include <stdio.h>
int fseek(FILE *fp, long int deplacement, int mode);
```

- *fseek()* est utilisée pour déplacer le pointeur du fichier *fp*. La nouvelle position est à *deplacement* de la position spécifiée par *mode*.
- *mode* prend 3 valeurs : *SEEK_SET* (début), *SEEK_CUR* (courant), et *SEEK_END* (fin). (Inclure <stddef.h> pour les *SEEK_**)
- *fseek()* renvoie 0 normalement, et non-nul en cas d'erreur :

```
#include <stddef.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
     FILE *fp;
     struct db_record {
        char name[40];
        short age;
     } rec;
     int size = sizeof(rec), recnum;
     char str[6];
     void usage(char *), err(char *); /*fonctions définies ailleurs*/
     if (argc != 2)
        usage(argv[0]);
     if ((fp = fopen(argv[1], "a+")) == NULL)
         err(argv[1]);
     while (1) {
        printf("Quel enregistrement (1-4) ? (0 pour fin):");
        if (!(recnum = atoi(fgets(str, 6, stdin))))
           break;
        if ((recnum < 0) || (recnum > 4))
           continue;
        fseek(fp, (long int)(size*(recnum - 1)), SEEK_SET);
        if (!(fread(&rec, size, 1, fp)))
           err(argv[1]);
        printf("Nom n°%d: %s, age: %d\n", recnum, rec.name, rec.age);
     } /* fin de while */
     fclose (fp);/* fermer le fichier */
     return 0;
} /* fin de main */
```

Vider la Mémoire Tampon associée à *FILE*

Format

```
#include <stdio.h>
int fflush(FILE *fp);
```

- *fflush()* force l'écriture sur disque des données présentes dans le tampon ; souvent utilisée pour rendre immédiat l'affichage d'un message, ou toute autre opération d'entrée/sortie.
- Le buffer de *stdout* est vidé à chaque newline. Les buffers de fichiers sont vidés lorsqu'un bloc est complet. Le buffer de *stderr* est vidé à chaque caractère écrit.
- Sans erreur, *fflush()* renvoie 0 ; sinon *EOF*.

```
#include <stdlib.h>
#include <stdio.h>
#define MIN 50
#define BUFSIZE 80
int main(void) {
     int num_crunch(void);/*fonction faisant tout le travail */
     int index, iq;
     char str[BUFSIZE];
     while (1) {
        printf("Entrer votre QI :  ");
        fflush(stdout);
        iq = atoi(fgets(str, BUFSIZE, stdin));
        if (iq < MIN)
           fprintf(stderr,"Quittez le navire, un idiot à la barre!");
        index = num_crunch();
        if ((iq + index) < (2 * MIN))
           break;
     }
     return 0;
} /* fin de main */
```

Résumé des Fonctions

Fonction	Valeur retournée sans erreur	Code D'erreur	Code de Fin de fichier
fopen()	*FILE**	*NULL*	sans objet
fclose()	0	*EOF*	sans objet
scanf()	nb d'affectations	*EOF* est retournée si une erreur intervient avant toute conversion.	
sscanf()	nb d'affectations		
fscanf()	nb d'affectations		
getchar()	caractère lu	*EOF*	*EOF*
fgetc()	caractère lu	*EOF*	*EOF*
fgets()	ptr sur chaîne lue	*NULL*	*NULL*
fread()	Nb d'items lus dans tous les cas		
printf()	nb car.écrits	code < 0	sans objet
sprintf()	l'adresse du premier argument dans tous les cas		
fprintf()	nb car.écrits	code < 0	sans objet
putchar()	car.écrit	*EOF*	sans objet
fputc()	car.écrit	*EOF*	sans objet
fputs()	code >= 0	*EOF*	sans objet
fwrite()	nb items écrits	nb items écrits	sans objet
ftell()	position en octets	-1L	sans objet
fseek()	0	code != 0	sans objet
rewind()	pas de valeur retournée		
fflush()	0	*EOF*	sans objet

Révision de Module

E/S Fichiers

Q. Quels sont les noms des pointeurs *`FILE`* pour les 3 fichiers ouverts par défaut ?

R.

Q. Quel fichier .h faut-il inclure pour utiliser ces identifiants ?

R.

Q.Quelle est la fonction d'affichage formatée sur un *`fp`* donné ?

R.

Ecrire un programme utilisant *`fgets()`* pour lire le fichier `/etc/motd` et *`printf()`* pour l'afficher à l'écran.

Travaux Pratiques 13 : E/S Fichiers

Présentation

Introduction à l'utilisation des fonctions d'E/S fichiers de niveau user de la librairie standard du C.

Exercices

1. **Niveau 1**. *Vérifier* les résultats du programme de révision en le compilant et en l'exécutant.

2. **Niveau 2**. Modifier `aged.c` (TP 10) comme suit :

   En utilisant la structure existante, permettre à l'utilisateur de saisir les champs name et data d'autant de structures que voulues, calculer les autres champs et sauver dans un fichier (ouvert en écriture) nommer `data.rec`, une structure à la fois. Gérer un compteur d'enregistrements. Ne pas utiliser de tableau :

   Quand l'utilisateur a fini, fermer le fichier.

   Demander à l'utilisateur un numéro d'enregistrement, ouvrir le fichier en lecture, lire l'enregistrement voulu et l'afficher.

   Puis fermer le fichier et quitter.

   Nommer le fichier source `newaged.c`.

3. **Niveau 3**. Ecrire un programme qui gère un fichier contenant les chaînes saisies par l'utilisateur et triées. Le nom du fichier sera passé par la ligne de commande :

   Si aucun nom de fichier n'est passé en argument, sortir avec un message d'usage.

   Demander à l'utilisateur les chaînes d'un tableau (d'au moins 8 éléments) en les rangeant dans l'ordre alphabétique.

- voir page suivante -

Quand l'utilisateur a entré toutes ou quelques chaînes (*au moins 2*), les écrires dans le fichier dont le nom est passé dans `argv[1]`.

Afficher les lignes / enregistrements et demander un numéro à détruire.
Détruire les numéros demandés.

Afficher le nouveau contenu et sortir.

Conseil 1 : Détruire l'enregistrement dans le tableau et réécrire tout le fichier. N'essayez pas de trier le fichier en place sauf si vous le désirez. Assurez-vous que vous ouvrez le fichier avec le mode w pour le créer ou l'effacer à l'ouverture.

Conseil 2 : *`strcmp()`* vous aidera à trier les chaînes en ordre alphabétique. Mais pour ne pas rendre ce TP excessivement long, vous trouverez ci-dessous une procédure de tri (remarquez les arguments) :

```
void ssort(int cnt,char sa[AMAX][SMAX],char str[]);
/* nb de chaînes dans le tableau */
/* tableau des chaînes */
/* chaîne à ranger dans le tableau */
{
  int ith;
  static char temp[SMAX], nul[1] = {'\0'};

  for (ith = 0; ith <= cnt; ith++) {
     if ((strcmp(sa[ith], str) > 0) ||
         (strcmp(sa[ith], nul)== 0)) {
        strcpy(temp, sa[ith]);
        strcpy(sa[ith], str);
        strcpy(str, temp);
     }
  }
} /* fin de ssort */
```

Conseil 3 : La fonction pour réarranger le tableau après suppression est similaire mais plus simple.

Nommer le fichier source `alphile.c`

Plus sur **cc** *et le Préprocesseur*

Objectifs

- Ecrire et utiliser les directives simples du préprocesseur et écrire des macros.
- Faire des compilations conditionnelles à l'aide des directives `#ifdef`.
- Définir des macros comme des fonctions en utilisant les marqueurs # et ## du préprocesseur.

Macros du Préprocesseur

Macros prédéfinies :

`__TIME__`	chaîne de la forme : "hh:mm:ss" indiquant l'heure de la compilation.
`__DATE__`	chaîne de la forme : "Mmm dd yyyy", Indiquant la date de compilation. Exemple : Dec 25 1991
`__FILE__`	Chaîne de caractère correspondant au nom du fichier source utilisé durant la compilation.
`__LINE__`	No de lignes dans le fichier courant. (constante décimale)
`__STDC__`	1 pour le mode de conformance -Xc ; 0 sinon.

```
#include <stdio.h>

int main(void)
{
    printf("Programme : %s.\nDate de compil.: %s.\n",__FILE__,
         __DATE__);
    return 0;
} /* fin de main */
```

Macros du Préprocesseur (suite)

- Définition de macros objets :

  `#define` <*identitifiant*> <*liste de remplacement*> *newline*

- Exemples

```
#define SIZE    256
#define MESSAGE "Enchanté de vous voir!"
```

- Définition des macros fonctions :

  `#define` *ident(liste opt.) liste de remplacement newline*

- Exemples :

```
#define SQ(X)       ((X) * (X))
#define PRINT(S1) printf("Date %s\n",(S1))
```

- L'identifiant immédiatement après le `#define` est appelé le *nom de la macro.*

Macros Fonctions

Les macros fonctions produisent du code *in-line*, et ainsi repésentent un gain de vitesse en évitant le temps dû à l'appel de fonction.

```
#include <stdio.h>
#define ABS(a) ((a)<0?-(a):(a))/* renvoie la valeur abs. de a */
#define CUBE(b) ((b)*(b)*(b))/* renvoie le cube de b */
#define MIN(a, b) ((a)<(b)?(a):(b))/* renvoie le min. de a et b */

int main(void)
{
     int num1, num2;

     printf("Entrer un entier à élever au cube :  ");
     scanf("%d", &num1);
     printf("Le cube de %d = %d!\n", num1, CUBE(num1));
     printf("Entrer un autre entier :  ");
     scanf("%d", &num2);
     printf("La valeur absolue de %d = %d.\n", num2, ABS(num2));
     printf("Le plus petit de %d et %d = %d.\n", num1, num2,
          MIN(num1,num2));
     return 0;
} /* fin de main */
```

Précautions à L'Utilisation des Macros

- Si la macro *SQR* est utilisée de la manière suivante, l'argument sera incrémenté deux fois ce qui ne correspond pas au résultat attendu !
- Les parenthèses doivent être utilisées autour de la macro et de *chaque* argument pour s'assurer une évaluation *correct*e. (voir *ABS*) :

```
#include <stdio.h>
#define SQR(a)  ((a)*(a))       /* renvoie le carré de a */
#define ABS(b)  b < 0 ? -b:b  /* renvoie la valeur abs de b ?*/
int main(void)
{
        int num1, num2;

        printf("Entrer un entier à élever au carré :  ");
        scanf("%d", &num1);
        printf("Le carré de %d = %d!\n", num1, SQR(++num1));
        printf("La valeur abs. de 3-5 = %d.\n",ABS(3-5));

        return 0;
} /* fin de main */
```

```
/*      Avec les déclarations ci-dessus... */

SQR(++num1);            /* est étendu comme... */
((++num1) * (++num1)); /* PAS bon ! */


ABS(3-5);               /* est étendu comme... */
3-5 < 0 ? -3-5 : 3-5;   /* PAS bon! */
```

Le Marqueur # du Préprocessseur

- Si le marqueur # précède immédiatement un argument dans la liste de remplacement d'une macro-fonction l'ensemble est remplacé par une chaine de caractère correspondant à l'argument (transformation en littéral).

Soit :

```
#define MAKESTR(X) # X "!"
int abc, xyz;
```

et les appels :

```
MAKESTR(abc);
MAKESTR(abc xyz);
```

Donnent :

"abc""!"
"abc xyz""!"

- Il ne peut y avoir que des espaces entre le # et l'argument dans la liste de remplacement ; ne pas mettre de parenthèses autour de l'argument.

Le Marqueur ## du Préprocesseur

- Si le marqueur *##* précède ou suit immédiatement un argument dans la liste de remplacement, l'ensemble sera remplacé par l'argument lui-même (collage des 2 arguments).
- Avant de reparcourir la macro pour d'autres noms d'arguments, le marqueur *##* est effacé et *concaténé* avec l'élément suivant.

Soit :

```
#define PASTE(X,Y) X ## Y
int a, xyz;
```

Les appels:

```
PASTE(a, xyz);
PASTE(a, 1);
PASTE(1., 34);
```

Donnent :

```
axyz            /* ceci n'est pas une chaîne */
a1
1.34
```

- Il ne peut y avoir que des espaces entre ## et le paramètre dans la liste de remplacement. *Ne pas* mettre de parenthèses autour de l'argument.

Compilation Conditionnelle

- Le préprocesseur fournit des possibilités de compilation conditionnelle, qui permettent à certaines portions de code d'être soit compilées soit ignorées en fonction de certaines conditions.

- Il y a trois types de directives conditionnelles. Chacune des directives contrôle la compilation des lignes de code en-dessous, jusqu'à la rencontre de la directive `#endif` :

```
#if      /* vrai si la constante qui suit est différente de 0 */

#ifdef   /* vrai si l'argument est défini par #define avant */

#ifndef  /* vrai si l'argument n'est pas défini par #define avant */

#elif    /* avec une expression constante, else conditionnel */

#else    /* complément de #if, #ifdef ou #ifndef *

#endif   /* fin de la portion de code concernée par la compilation
            conditionnelle */
```

Compilation Conditionnelle

Exemples de compilation conditionnelle :

```
% more my_prog.c
...
#ifdef DEBUG
        /* printf de debug... */
        printf("my_func : compteur de boucle %d\n", index);
#endif
...
% cc -DDEBUG -Xc my_prog.c -o my_prog
% more my_prog.c
...
#if (DEBUG > 50)
        /* printf de debug lourd ... */
        printf("Gros debug!\n");
#else
        printf("Petit debug.\n");
#endif
...
% cc -DDEBUG=42 -Xc my_prog.c -o my_prog
% more my_prog.c
...
#define DEBUG
#ifdef DEBUG
        /* printf de debug... */
        printf("Gros debug!\n");
#endif
...
```

Annuler la Définition de Constantes

- Un nom `#define` (`#define NOM`) peut être oublié de force par la directive `#undef` (`#undef NOM`).
- Tous les `#ifdef NOM` à suivre dans le source seront évalués FAUX.
- Inversement, tous les `#ifndef NOM` seront évalués VRAI :

```
% more my_prog.c
...
#undef NOM
#ifndef NOM
        printf("NOM est non-défini !\n");
#endif
#ifdef sun
        printf("sun est défini.\n");
#endif
#ifdef sparc
        printf("sparc est défini.\n");
#endif
...
```

(annulation sur la ligne de commande)

```
% cc -Usun -Xa my_prog.c -o my_prog
```

Allocation Dynamique de Mémoire 15

Objectifs

- Allouer dynamiquement la mémoire dans un programme C.
- Libérer la mémoire allouée dynamiquement au préalable.
- Programmer une liste *simplement chaînée.*

Evaluation

Travaux Pratiques 15 et révision de module.

Image Mémoire d'un Process

- Il y a essentiellement 2 manières pour un programme d'acquérir de la mémoire pour ses données. L'une est la déclaration de variables. L'autre est l'allocation dynamique (au moment de l'exécution).
- L'allocation dynamique de mémoire met à disposition du programme, de la mémoire additionnelle par extension du tas (heap), une des parties du segment data du process.

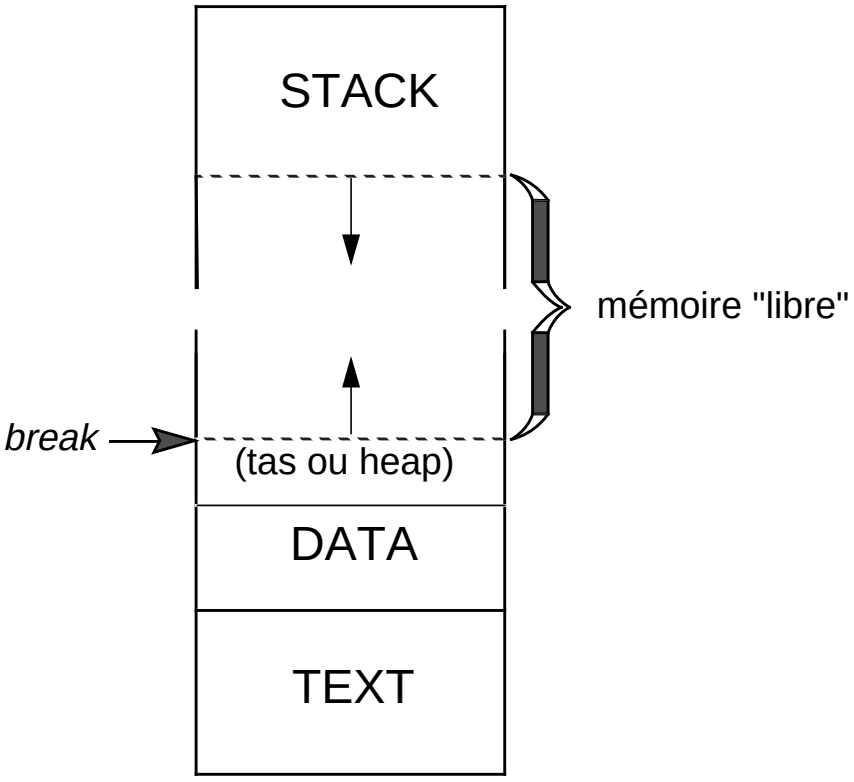


Image du process

Allocation Dynamique de Mémoire – *malloc()*

Format

```
#include <stdlib.h>
void *malloc(size_t taille)
```

- *malloc()* est la manière la plus banale pour demander de la mémoire dynamique à SunOS. La quantité de mémoire allouée est au moins de *taille* octets.
- *malloc()* renvoie un pointeur sur la mémoire allouée. Si rien n'a été alloué ou qu'une erreur s'est produite, *malloc()* renvoie le pointeur *NULL*.
- On peut forcer le type de l'adresse retournée par *malloc()* en préfixant l'appel avec *(type)*, afin de l'adapter au type d'objet pour lequel l'allocation est faite.

```
#include <stddef.h>
#include <stdlib.h> /* déclaration de malloc() */
#include <stdio.h>  /* déclaration de gets() */
#include <string.h> /* déclaration de strcpy() */

#define BUFFER 100

int main(void){
        char *dynarr;

        dynarr = (char *) malloc(BUFFER);

        strcpy(dynarr, "hello");
        puts(dynarr);

        printf("Entrer une chaîne : ");
        fgets(dynarr, BUFFER, stdin);
        printf("Chaîne saisie = %s \n", dynarr);
        return 0;
}
```

Allocation Dynamique de Mémoire – *calloc()*

Format

```
#include <stdlib.h>
void *calloc(size_t nelem, size_t taille)
```

- *calloc()* est utilisé pour alloué l'espace d'un *tableau*. Ce tableau a *nelem* éléments, et l'espace contigu alloué est au moins de (*nelem* * *taill*e) octets.
- En cas de succès, *calloc()* renvoie un pointeur sur la mémoire allouée. En cas d'erreur ou d'impossibilité d'allocation, la fonction renvoie le pointeur *NULL*.
- *calloc()* renvoie de la mémoire initialisée à 0.
- Un tableau créé dynamiquement (avec *calloc()* ou *malloc()*) peut être traité *exactement* comme un tableau statique classique :

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>/* déclaration de calloc() */
int main(void) {
     int *int_ptr, *ip, num = 0, index, asize;
     char tmp[16];
     printf("Quelle taille pour le tableau ?:  ");
     asize = atoi(fgets(tmp, 16, stdin));/* saisie/conversion */
     if ((int_ptr = (int *)calloc(asize, sizeof(int))) != NULL) {
        for (ip = int_ptr; ip < (int_ptr + asize); ip++, num++) {
           /* initialisation notation pointeur... */
           *ip = num;
        } /* fin de for */
        /* c'est possible autrement... */
        for (index = 0; index < asize; index++) {
           /* initialisation mode tableau, notation indicée... */
           int_ptr[index] = index % 3;
        } /* fin de for */
     }
     return 0;
} /* fin de main */
```

Allocation Dynamique de Mémoire – *free()*

Format

```
#include <stdlib.h>
void free(void *ptr)
```

- *free()* est utilisée pour rendre au système la mémoire allouée avec *malloc()* ou *calloc()*.
- L'argument *free()* est le pointeur retourné préalablement par *malloc()* ou *calloc()*:

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h> /*déclarations de malloc(), free(), et atof()*/
#define BUFSIZE 40
struct record {
        char name[BUFSIZE];
        float salary;
        int age;
};
int main(void)
{
        struct record *sptr;
        char tmp[BUFSIZE];
        int size = sizeof(struct record);
        void some_func(struct record *);
        if ((sptr = (struct record *)malloc(size)) != NULL) {
            printf("Entrez votre nom :  ");
            fgets(sptr->name, BUFSIZE, stdin);
            printf("Entrez votre âge :  ");
            sptr->age = atoi(fgets(tmp, BUFSIZE, stdin));
            printf("Votre salaire annuel :  ");
            sptr->salary = atof(fgets(tmp, BUFSIZE, stdin));
        } /* fin de if */
        /* traitement quelconque de la structure... */
        some_func(sptr);
        free(sptr);/* restitution de la mémoire au système */
        return 0;
} /* fin de main */
```

Révision Partielle

malloc( )

Q. En quoi l'allocation de mémoire peut-elle être *dynamique* ?

R.

Q. Qu'est-ce que représente la valeur retournée par `malloc()`?

R.

Q. Que fait l'instruction `ptr=(struct record *)malloc(size);`?

R.

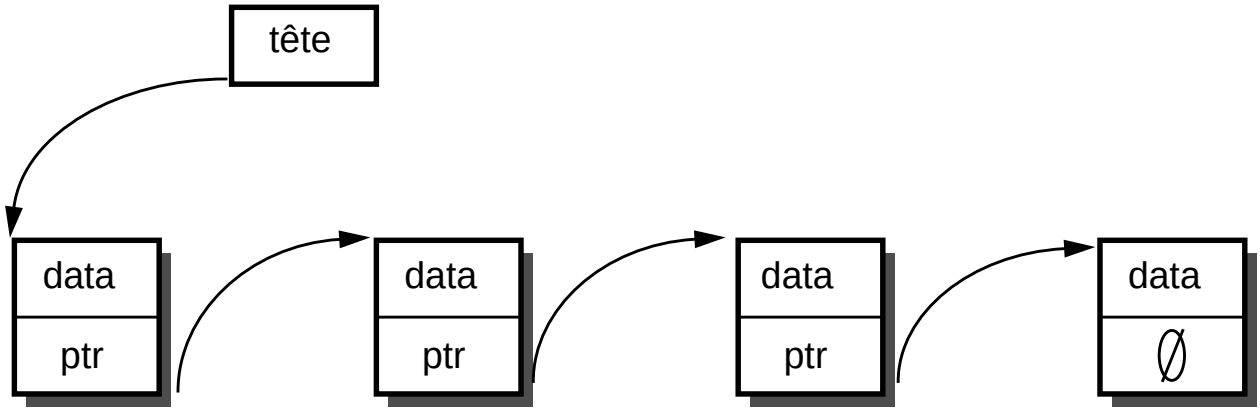
Q. Quelle est l'instruction qui rend la mémoire au système, et pourquoi doit-on l'utiliser ?

R.

Ecrire un bout de programme qui déclare une structure avec un tableau de 24 caractères, un entier, un flottant, et un pointeur. Utiliser `malloc()` pour allouer dynamiquement la mémoire pour une structure (attention, juste un bout de pgm) :

Après les Tableaux - La Liste Chaînée

- Les listes chaînées sont une forme fréquente de *structure de données dynamiques.*
- Les listes chaînées sont utilisées notamment dans 2 cas. Pour les tableaux de taille inconnue, et pour le traitement et le stockage de base de données.
- Les listes sont généralement soit *simplement* chaînées, ou *doublement* chaînées. Ceci fait référence au nombre de pointeurs vers d'autres éléments de la liste que l'on trouve dans la structure d'un *nœud de liste* :

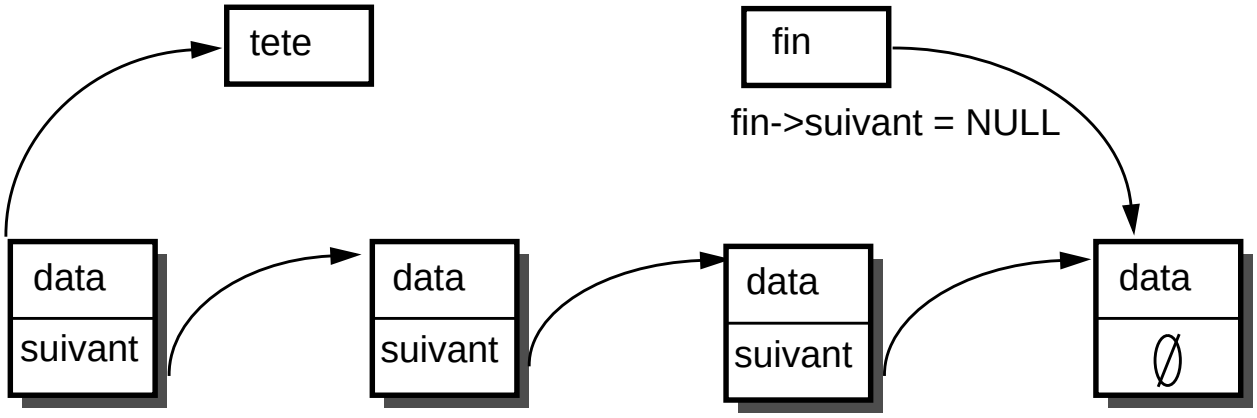


Liste simplement chaînée

Construction d'une Liste Simplement Chaînée

- Les listes peuvent être construites d'au moins deux manières : en ajoutant simplement des éléments à la fin, ou en insérant les éléments à des emplacements spécifiques pour un ordre donné (listes triées).
- Toute l'attention doit être portée à la conservation de la *tête* de la liste, et à la terminaison de la liste par un pointeur *NULL* :

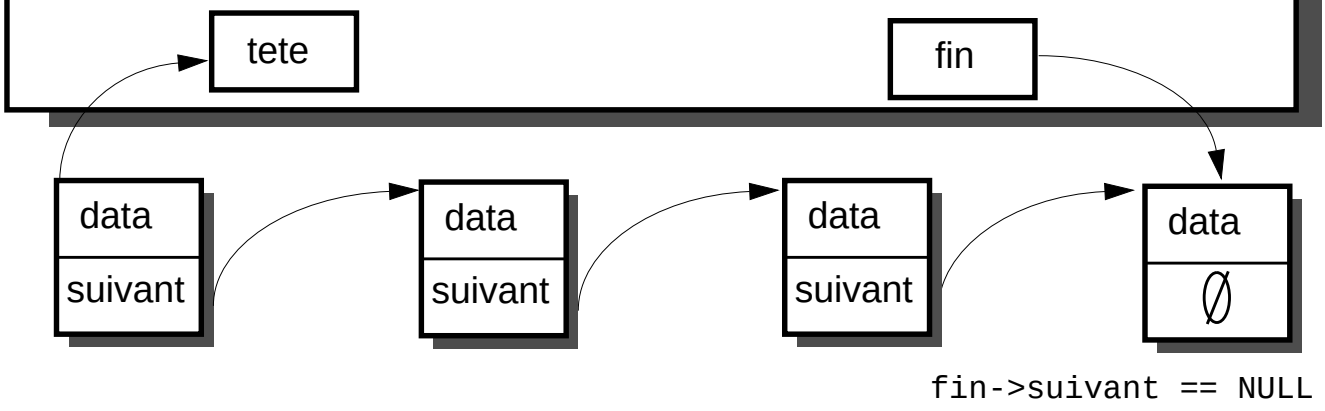
```
struct node {
        int data;               /* les données sont là */
        struct node *suivant;/* pointeur vers le nœud suivant */
} *tete, *fin;              /* pointeurs de type (struct *node) */
int cntr = 1;
tete = (struct node *)malloc(sizeof(struct node));      /* 1er élément */
fin = tete;                 /* conservation de la tête de liste */
fin->data = cntr;           /* renseignement du 1er élément */
fin->suivant = NULL;        /* fin de liste sur NULL */
while (encore) {
    fin->suivant=(struct node *)malloc(sizeof(struct node));/* suivant */
    fin = fin->suivant;     /* avance de un dans la liste */
    fin->data = ++cntr;     /* renseignement de l'élément courant */
    fin->suivant = NULL;    /* fin sur NULL */
} /* fin de while */
...
```



Parcourir une Liste Simplement Chaînée

- Dans un parcours de liste, tous les éléments de la liste doivent être consultés pour trouver le suivant.
- Le pointeur *NULL* est l'indicateur de fin. Une liste est parcourue en passant de suivant en suivant, tant que le pointeur *NULL* n'est pas rencontré :

```
struct node {
        int data;   /* données de la liste */
        struct node *suivant;/* pointe vers le suivant */
} *tete, *fin;      /* pointeur de type (struct *node) */
int cntr = 1;

fin = tete ;    /* préserve la tête de liste */
while (fin != NULL) {
        printf("élém %d:  donnée= %d\n", cntr, fin->data);
        cntr++;
        fin = fin->suivant; /* pointe sur le suivant */
} /* fin de while */

...
```



La Liste Simplement Chaînée en Action

- Exemple de programme montrant l'utilisation d'une liste simplement chaînée pour mémoriser les enregistrements d'un fichier :

```
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>     /* déclarations de malloc(), et free() */
#define TAILLE 256
int main(void) {
        struct db {
                char data[TAILLE];
                struct db *suivant;
        } *tete, *fin, *temp;
        char str[TAILLE];
        int db_size = sizeof(struct db);
        printf("Entrer une chaîne de caractères : ");
        if (fgets(str, TAILLE, stdin) == NULL) {
                fprintf(stderr, "Erreur sur fgets\n");
                exit(1);
        }
        tete = (struct db *)malloc(db_size);
        strcpy(tete->data, str);
        fin = tete;
        printf("Entrer une chaîne de caractères : ");
        while (fgets(str, TAILLE , stdin) != NULL) {
            fin->suivant = (struct db *)malloc(db_size);
            fin = fin->suivant;
            strcpy(fin->data, str);
            printf("Entrer une chaîne de caractères : ");
        }
        fin->suivant = NULL;
        fin = tete ;
        printf("\n\n");
        while (fin) {
                printf("Enregistrement: %s\n", fin->data);
                temp = fin;
                fin = fin->suivant;
                free(temp);
        }
        return 0;
} /* fin de main */
```

PAGE INTENTIONNELLEMENT BLANCHE

Un Autre Exemple de Liste Chaînée

```c
#include <stddef.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#define DATA_LENGTH 100
typedef struct link {
        char  data[DATA_LENGTH];
        struct link *next;
} Link;
void    add(void), delete(void), print(void);
Link    *new_link(char *), *head, *curr;
int main(void) {
     int ch;
    head = curr = new_link("");
    for (;;) {
        printf("Ajoute, Détruit, Suivant, Tête, Impression,"
        " Quitte :  ");
        ch = getchar();
        while (getchar() != '\n');
        switch (isupper(ch) ? tolower(ch) : ch) {
            case 'a':
                add();
                break;
            case 'd':
                delete();
                break;
            case 's':
                if(curr->next == NULL)
                    printf(".fin.\n");
                else {
                    curr = curr->next;
                    printf("%s", curr->data);
                }
                break;
            case 't':
                curr = head;
                printf(".top.\n");
                break;
            case 'i':
                print();
                break;
            case 'q':
                exit(0);
                break;
        } /* fin de switch */
    }   /* fin de for */
    return 0;
} /* fin de main */
```

Un Autre Exemple de Liste Chaînée (suite)

```
Link *new_link(char *data)
{
        Link *temp;
        if ((temp = (Link *)malloc(sizeof(Link))) == NULL) {
            fprintf(stderr, "Erreur de malloc\n");
            exit(1);
        }
        strcpy(temp->data, data);
        temp->next = NULL;
        return temp;
}
void add(void)
{
        Link *temp;
        char data[DATA_LENGTH];
        printf("donnée ?: ");
        fgets(data, DATA_LENGTH, stdin);
        temp = new_link(data);
        temp->next = curr->next;
        curr = curr->next = temp;
}
void delete(void)
{
        Link *prev;
        if (head->next == NULL)
            return;  /* liste vide */
        if (curr == head)
            head = head->next;
        else { /* recherche l'élément précédent */
            for (prev = head; prev->next != curr; prev = prev->next);
                prev->next = curr->next;
        }
        free(curr);
        curr = head;
}
void print(void)
{
        Link *temp;
        printf(".tête.\n");
        for (temp=head->next; temp != NULL; temp=temp->next)
            printf("%s", temp->data);
        printf(".fin.\n");
}
```

Travaux Pratiques 15 : Mémoire Dynamique et Listes

Présentation

Introduction au concept et à l'utilisation de l'allocation dynamique de mémoire et aux listes simplement chaînées.

Exercices

1. **Niveau 1**. *Vérifier* les résultats du bout de code de révision, en le complétant (en faire un programme), en le compilant et en l'exécutant.

2. **Niveau 2**. Modifier le programme du TP 10 `strux.c` :

   Au lieu d'un tableau de 4 éléments, gérer une liste chaînée de longueur indéterminée. Continuer d'ajouter des éléments tant que l'utilisateur le demande (ajouter au moins un élément).

   Sauver les informations dans un fichier à la fin de la saisie, et rendre la mémoire dynamique au système. Le nom du fichier sera passé par la ligne de commande. Sinon, donner un message d'erreur et quitter le programme.

   Après sauvegarde dans le fichier, le relire et reconstruire la liste.

   Après reconstruction, demander à l'utilisateur soit un nom, soit s'il veut visualiser toute la liste.

3. Ecrire une fonction qui affiche toute la liste et/ou affiche juste l'élément demandé par l'utilisateur, et un message en cas de recherche vaine.

   Quand c'est fini rendre la mémoire au système et quitter.

   Utiliser des fonctions chaque fois que possible, afin de découper le programme.

4. **Niveau 3** *(facultatif)*. Gérer la liste triée par âge.
   Nommer le fichier source `listrux.c`.

Introduction aux Fonctions Récursives

Objectifs

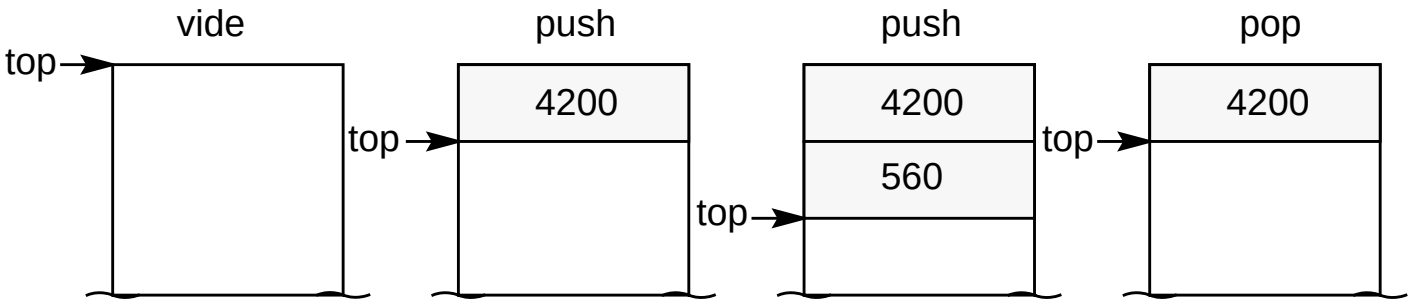
- Expliquer la définition d'une fonction récursive.
- Ecrire des fonctions récursives simples en C.

Evaluation

Travaux Pratiques 16.

Définition de la Pile

- Une *pile* est un ensemble de registres câblés ou une portion de la mémoire *principale*, qui sont utilisés pour des calculs arithmétiques ou pour des opérations *internes*. Les piles fonctionnent sur le principe dernier entré-premier sorti *(last-in-first-out)* (LIFO).
- En informatique, le terme *empiler (push)* revient à ajouter un élément en haut d'une pile (top).
- En informatique toujours, le terme *dépiler (pop)* revient à enlever l'élément en haut de pile :



Remarque : Le pointeur de pile "descend" lorsqu'on empile quelque chose, car, en mémoire, la pile est toujours allouée depuis les adresses hautes vers les adresses basses.

Qu'est-ce que la récursivité?

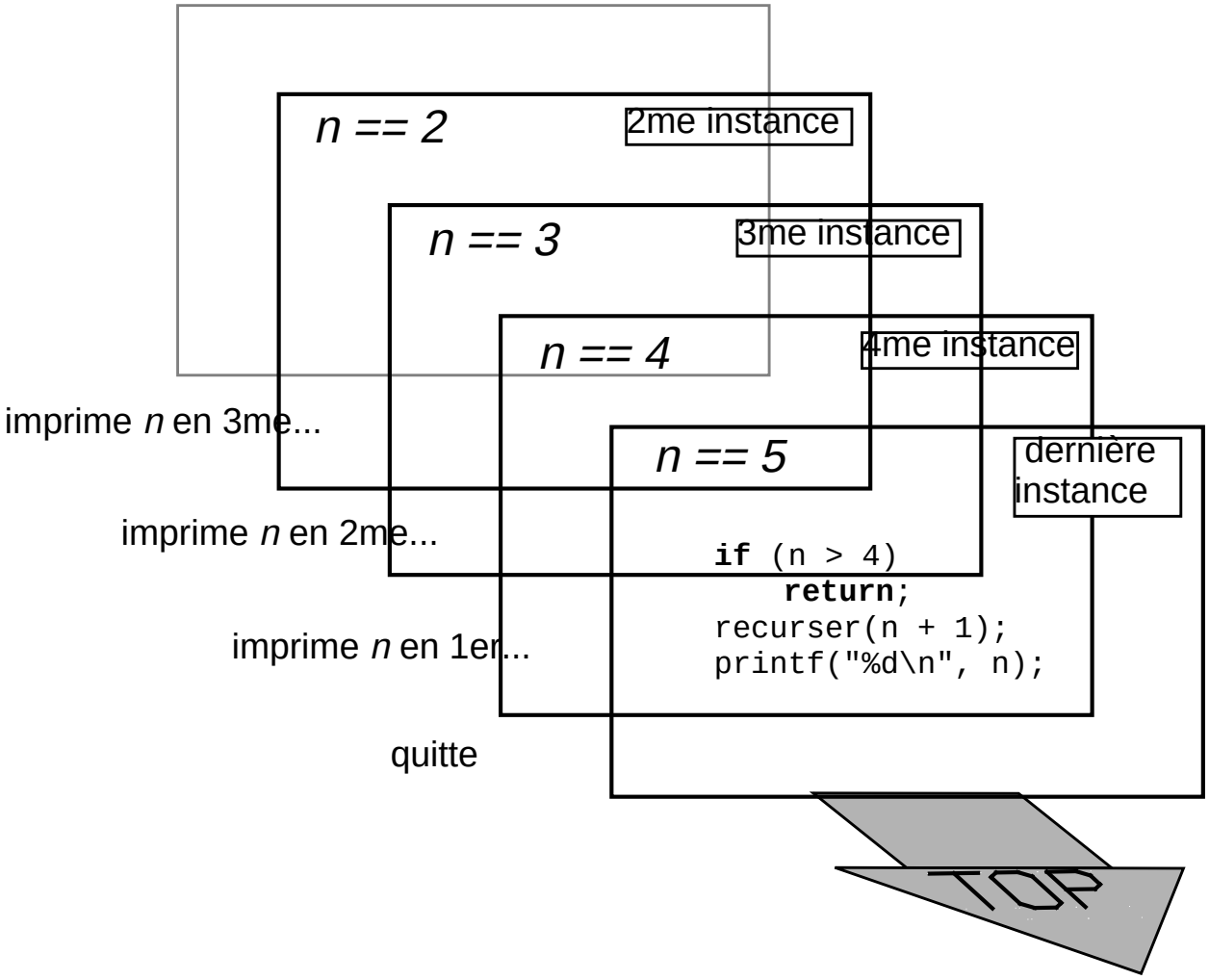
- Une fonction récursive est une fonction qui, pour remplir sa mission, doit s'appeler *elle-même* :

```
#include <stdio.h>
int main(void) {

    void recurser(int);

    printf("L'infinité est le possible rendu inévitable.\n");
    recurser(1);
    return 0;
} /* fin de main */

void recurser(int n) {
        if (n > 4)
            return;
        recurser(n + 1);
        printf("%d\n", n);

} /* fin de recurser */
```

- Généralement, une fonction récursive n'est jamais appelée indéfiniment. Il doit exister une ou plusieurs conditions de fin de boucle récursive.

Mécanismes d'une Fonction Récursive

- Pour chaque instance de fonction, un ensemble unique / différent de variables locales (toutes ayant les mêmes noms) est créé dans la *pile*.
- Les fonctions récursives semblent souvent s'exécuter en sens inverse des appels. Dans l'exemple, tous les appels sont faits *avant* la 1ère impression de *n* ; l'impression commence à 4 puis redescend :



Métaphore de la Pile

Applications de la Récursivité

- La fonction la plus présentée comme exemple de récursivité est celle qui calcule la *factorielle* d'un nombre. On la note : *n!* (factorielle n).
- La factorielle d'un nombre positif *n* est le produit des nombres de 1 à *n*. Si *n* == 4 alors *n !* == 4 * 3 * 2 * 1 == 24. Si *n* <= 1, alors *n!* == 1.
- Exemple de fonction récursive calculant *des factorielles* :

```
#include <stdio.h>
int main(void)
{
    unsigned int num, factorial(unsigned int);
    char tmp[8];

    printf("Entrer un nombre (n <= 13):  ");
    num = atoi(fgets(tmp, 8, stdin));
    printf("La factorielle de %d = %d.\n",num, factorial(num));
    return 0;
} /* fin de main */

unsigned int factorial(unsigned int n)
{
    if (n <= 1)
        return(1);
    else
        return(n * factorial(n - 1));

} /* fin de factorial */
```

Travaux Pratiques 16 : Récursivité (Facultatif)

Présentation

Compilation et utilisation d'un programme contenant la fonction récursive *`factorial()`*.

Exercices

1. **Niveau 1 :** Vérifier que le programme de la page 16-5 fonctionne en le compilant et en l'exécutant.

   Ajouter une boucle pour que le programme calcule toutes les factorielles entre 1 et un nombre saisi *par l'utilisateur.* Si l'utilisateur saisit 10, le programme affiche les factorielles de 1, 2, 3, 4, 5, 6, 7, 8, 9, et 10.

   Si l'utilisateur entre un nombre supérieur ou égal à 14 donner un message d'erreur ou d'usage et quitter le programme.

2. **Niveau 2 :** *(facultatif)* Faire un programme qui vérifie la limite de calcul de factorielle. On peut utiliser une boucle for. Utiliser des unsigned int et afficher les résultats.

   Nommer le programme `factorial.c`

Conseils de Mise au Point

Cette annexe contient quelques informations utiles pour régler des erreurs de compilation ou d'exécution.

Erreurs Fréquentes

Généralités

variables non-initialisées.

Erreur de dépassement de 1.

Traiter les tableaux en commençant les indices à 1 au lieu de 0.

Oublier la fin des commentaires */ .

Pas de point-virgule en fin d'instruction.

Types, Opérateurs et Expressions

Utiliser *`char`* pour *`int`* pour la valeur retournée par *`getchar()`*.

Backslash (\) tapé pour (/), comme dans /n au lieu de \n.

Déclarer des arguments de fonction (non prototypée) après l'accolade.

Utiliser des opérateurs relationnels sur les chaînes, comme s == "end", au lieu de *`strcmp()`*.

Oublier le caractère '\0' à la fin d'une chaîne de caractères.

Utiliser = pour ==.

Erreur de 1 dans les boucles avec indices.

Erreur de priorité ou d'associativité des opérateurs.

Erreurs Fréquentes

Structures de Contrôle

`else` mal placé.

`break` manquant dans un `switch`.

Boucle accidentellement jamais exécutée.

Structures des Fonctions et Programmes

Mauvais ordre d'argument.

Mauvais type d'argument (fonctions non ou mal prototypées).

Penser qu'une variable `static` soit réinitialisée à chaque appel.

Manque de parenthèses dans la définition d'une macro.

Pointeurs et Tableaux

Passer une valeur au lieu d'un pointeur, et vice-versa.

Confondre `char` avec `char *`.

Déclarer des pointeurs sur chaînes de caractères sans allocation de mémoire avant utilisation.

Confondre quotes (`'\n'`) et guillemets (`"\n"`).

Mots-Clefs et Table ASCII

Cette annexe présente les informations suivantes :

- Mots-Clefs réservés
- La table ASCII (Décimale)

Mots-clefs Réservés

Les mots-clefs suivants sont réservés en C ANSI et ne peuvent être utilisés comme identifiants :

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Sun ANSI C définit un mot-clef de plus, *asm*. Mais *asm* n'est pas accepté dans le mode de conformance *-Xc*. L'usage de *asm* est pris comme une *common extension*.

La Table ASCII (Décimale)

Noter que les majuscules, les minuscules et les chiffres sont des ensembles de valeurs consécutives. Mais ces ensembles sont séparés.

0 nul	1 soh	2 stx	3 etx
4 eot	5 enq	6 ack	**7 bel**
8 bs	9 ht	**10 nl**	11 vt
12 np	13 cr	14 so	15 si
16 dle	17 dc1	18 dc2	19 dc3
20 dc4	21 nak	22 syn	23 etb
24 can	25 em	26 sub	27 esc
28 fs	29 gs	30 rs	31 us
32 sp	33 !	34 "	35 #
36 $	37 %	38 &	39 ’
40 (	41)	42 *	43 +
44 ,	45 -	46 .	47 /
48 0	49 1	50 2	51 3
52 4	53 5	54 6	55 7
56 8	57 9	58 :	59 ;
60 <	61 =	62 >	63 ?
64 @	65 A	66 B	67 C
68 D	69 E	70 F	71 G
72 H	73 I	74 J	75 K
76 L	77 M	78 N	79 O
80 P	81 Q	82 R	83 S
84 T	85 U	86 V	87 W
88 X	89 Y	90 Z	91 [
92 \	93]	94 ^	95 _
96 ‘	97 a	98 b	99 c
100 d	101 e	102 f	103 g
104 h	105 i	106 j	107 k
108 l	109 m	110 n	111 o
112 p	113 q	114 r	115 s
116 t	117 u	118 v	119 w
120 x	121 y	122 z	123 {
124 |	125 }	126 ~	127 del

Mémento du C

Mémento du C

/* commentaires en slash étoile et étoile slash */

Squelette de Prototype de Fonction

```
#include <file.h>
int main(void)
{
        déclarations;
        instructions;
}
type
function(paramètres)
{
        /*corps de fonction */
}
```

Déclarations

```
char a, string [], *cptr;
int i, iarr[], *iptr;
float f, flarr[], *fptr;
double dd, dblarr[], *dptr;
struct tagname {
   /*déclaration de membres*/
}variable-list;
typedef old-type new-name;
```

Fonctions d'entrée

```
scanf("%d%f%c%s",&i,&f,&c,string);
while ( (c=getchar()) != '\n')
```

Fonctions de sortie

```
printf("%d%.2 %c%s\n",i,f,c,string);
```

Opérateurs Relationnels

```
< > <= >= == != && ||
```

Opérateurs d'affectation

```
= += -= *= /= %=
```

Mémento du C (suite)

Structures de Contrôle

```
condition ? exp_si_vrai:exp_si_faux

if (expression) {
    instructions;
}
else if (expression) {
    instructions;
}
else {
    instructions;
}

for (initialisation; condition; pas) {
    instructions;
}

while (expression) {
    instructions;
}
```

do {
instructions;
} **while** (expression);

switch (expression) {
case valeur1:
instructions;
break;
case valeur2:
instructions;
break;

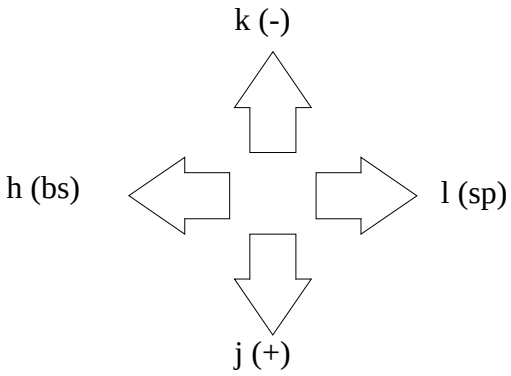
<etc>

default:
instructions;
break;
}

Pointeurs

```
int i, iptr*;
char string[], *stptr;
iptr = &i;
strptr = string;
*iptr = 5;
strptr = "Une chaîne";
*strptr = 'A';
```

Mémento vi



Déplacements du curseur :

h, j, k et l ,très bien pour les pros du clavier
^H (backspace), + (ou <CR>), -, barre d'espace
Touches flêchées comme diagramme ci-dessus

Déplacements page à page :

^f	une page en avant
^b	une page en arrière
^d	une demie page en avant
^u	une demie page en arrière

Déplacement sur l'écran

H	Home, haut d'écran
M	MIDDLE milieu d'écran
L	LAST dernière ligne de l'écran
G	GOTO dernière ligne du fichier
nG	GOTO *n-ième* ligne du fichier (ou :n)
^G	GIVES status du fichier

Déplacements sur la ligne

w	mot suivant
b	mot prècédent
e	fin du mot
0	début de ligne (ou ^)
$	fin de ligne

Recherche

/chaîne	recherche en avant de ’chaîne’ (exemple: /while)
n	next : occurence suivante de ’chaîne’ (N - précédente)

Sortie de vi

:q!	"laissez moi sortir" QUITTER sans sauver
:w	sauver sans quitter *vi*
:wq	sauver puis quitter *vi*
ZZ	sauver puis quitter *vi* (depuis le mode commande)

Passer en mode Insertion

Remarque : utiliser la touche ESC pour sortir de ce mode.

i	insérer avant le curseur
I	insérer en début de ligne
a	insérer après le curseur
A	ajouter en fin de ligne
o	ajouter une ligne sous le curseur
O	ajouter une ligne au-dessus du curseur
r	remplacer un caractère sans passer en insertion
R	mode refrappe, écrire sur le texte existant (ESC pour fin)
cw	change mot (c*n*w change *n* mots)
C	change jusqu’à la fin de ligne
u	"Ce n’est pas ce que je voulais" annule la dernière modification
U	Récupère la ligne complète

Note: Les caractères i, a, o, r, c et s (majuscules et minuscules) passent en mode insertion. Si le mode insertion est déclenché involontairement, taper ESC, puis u.

Copie de texte

Y	copier une ligne vers un buffer
*n*Y	copier *n* lignes
*n*yy	copier *n* lignes

Effacer

x	effacer un caractère (comme 'd espace')
dw	effacer un mot
D	effacer jusqu'à la fin de ligne
dd	effacer la ligne (en la mettant dans un buffer = couper)
*n*dd	effacer *n* lignes (*10dd* efface *10* lignes)

Placer

p	placer le contenu du buffer sur la ligne suivante
P	placer le contenu du buffer sur la ligne précédente
xp	permuter deux caractères

Recherche/Remplace (exemple)

/chaîne	rechercher la chaîne à remplacer
cw	remplacer de la manière adéquate (dw, r, s, etc.)
n	passer à l'occurence suivante de 'chaîne'
.	répéter la commande

Remplacement Global

:1,$s/old/new/g	de la ligne 1 à la fin de fichier ($) remplacer "old" par "new". Exemple. :1,$s/sun/Sun/g

Effacement global

:g/chaîne/d	effacer les lignes contenant une chaîne (exp.reg.) Exemple :g/###/d pour effacer les lignes contenant "###".

Insérer des fichiers

:r fichier	insère le contenu du fichier à l'emplacement du curseur.

Travailler sur 2 fichiers

:w	sauver fichier1 avant tout
:e file2	éditer un fichier2
:w	sauver le fichier2 avant de revenir au 1
:e #	retour au fichier1

Commandes Diverses

:! cmd	lancer une commande shell depuis l'éditeur
~	(tilde) MAJ->min et min->MAJ
%	mise en correspondance de parenthèses, accolades...
mx	pose la marque 'x' (:d'x efface jusqu'à la marque 'x'
^V	insertion de caractères spéciaux (exemple ^L)
?chaîne	recherche (comme /) mais en arrière
:n,n w fichier	sauve les lignes n à m dans fichier (exemple:2,20 w ff)
J	concatène la ligne suivante avec la ligne courante
:set ai	mode auto-indentation (crénelage de la marge gauche)
:set list	montre les fins de ligne et les caractères de contrôle
:set nows	arrête le rebouclage en début de fichier sur recherches
:set ts=*n*	change la valeur d'une tabulation (par défaut 8)
:set wm=*n*	inserer un newline à la colonne n

Savoir Lire le C

Cette annexe vous donne des informations concernant :

- La lecture de déclarations
- La lecture d’instructions

Déclarations

Règle : trouver l'identifiant et s'en écarter en spirale dans le sens des aiguilles d'une montre. Traiter l'intérieur des parenthèses avant l'extérieur et lire le type (le premier mot de la déclaration) en dernier.

Certains éléments de syntaxe doivent être lus comme suit :

*	est un pointeur sur un *ou* pointe sur un
[*n*]	est un tableau de *n*
( )	est une fonction qui retourne

Exemples :

`int x;`	`x` est un int
`int *y;`	`y` est un pointeur sur un int
`int *e[3];`	`e` est un tableau de 3 pointeurs sur int
`int (*c)[5];`	`c` est un pointeur sur un tableau de 5 int
`int f( );`	`f` est une fonction qui retourne un int
`int *b( );`	`b` est une fonction qui retourne un pointeur sur int
`int (*r)( );`	`r` pointe sur une fonction qui renvoie un int
`int **u;`	`u` pointe sur un pointeur sur int
`int *(*t)( );`	`t` pointe une fonction renvoyant un pointeur sur int

Instructions

Règle : interpréter comme ci-dessous. Toutes les expressions, sauf la dernière, sont lues de la gauche vers la droite.

`s = a`	`s` prend la valeur de `a`
`x == y`	`x` est égal à `y`
`k[3]`	k indice 3 (ou "k trois")
`m(r)`	m de r
`*t`	l'objet pointé par `t`
`&v`	l'adresse de *v*
`q->r`	`r` est un membre de la structure pointée par `q`

Exercices de Lecture de Déclarations

Exprimer en français les déclarations suivantes :

1. `float velocity;`

2. `char CommandString [25];`

3. `int *CurrentSector;`

4. `char **argv;`

5. `int *ViewscreenStatus[5];`

6. `float (*ShieldPower) [6];`

7. `int *NotIntuitive [900][36];`

8. `int (*ScreenDisplay) [900][36];`

9. `float *StarDate ( );`

10. `int (*Acceleration) ( );`

Exemples de Programmes Divers

Cette annexe contient des exemples tels que :

- Fonction à nombre variable d'arguments et *__STDC__*
- *getstring()*
- *getfloat()*
- Plus sur les unions
- Plus sur les listes chaînées
- Plus sur les fonctions chaînes de caractères
- Pointeurs sur fonctions
- Exemples divers

```
/* Définition d'une fonction à nombre variable d'arguments.
Utilisation de la macro __STDC__ pour déterminer si la compilation se
fait en ANSI C ou en C traditionnel.Compilation en C ANSI par "acc -
Xc var.c -o var"et en C traditionnel par "cc var.c -o var". */
#include <stdio.h>
#ifdef __STDC__
#include <stdarg.h> /* fichier .h ANSI pour les arg. variables */
#else
#include <varargs.h>/* fichier .h en C traditionnel */
#endif
#ifdef __STDC__
int fun(int, ...);/* prototypage ANSI C */
#endif
main(){
        fun(0);
        fun(2,3,"wokka");
        fun(4,12, "hello", 47, "goodbye");
}
#ifdef __STDC__
int fun(int count, ...)
#else
fun(va_alist)
va_dcl
#endif
{
#ifndef __STDC__
        int count;
#endif
        va_list argptr;
        int i, intparam;
        char *string;
#ifdef __STDC__
        va_start(argptr, count);
#else
        va_start(argptr);
        count = va_arg(argptr, int);
#endif
        printf("%d parameters\n", count);
        for (i = 0; i < count; i += 2) {
            intparam = va_arg(argptr, int);
            string = va_arg(argptr, char *);
            printf("entier = %d\nchaîne = %s\n", intparam, string);
        }
        va_end(argptr);
}       /* fin */
```

```
/*********************************************************
**
**    getstring - affiche le message d'invite passé en paramètre suivi
**            d'une virgule et d'un espace,lit une chaîne de N-1
**           caractères maximum, enlève le newline de fin (si besoin)
**            et traite les caractères restants éventuellement
**            dans le buffer d'entrée.
**
**            retourne un pointeur vers la chaîne ou le pointeur
**            NULL si la saisie rate
**
**      usage - char name[40];
**          if (getstring("Entrer votre nom", name, 40) == NULL)
**          {
**              fprintf(stderr, "Impossible de lire le nom\n");
**              exit(1);
**          }
**      note -compiler sans linker, puis linker avec votre programme.
**      note -assurez vous d'inclure stddef.h pour la définition de
**           NULL
*/
char *getstring(char *prompt, char *inbuf, int bufsize)
{
        int len;

        printf("%s: ", prompt);
        if (fgets(inbuf, bufsize, stdin) == NULL)
            return(NULL);
        else {
            len = strlen(inbuf) - 1;
            if (inbuf[len] == '\n')
                inbuf[len] = '\0';
            else            /* pas de '\n' trouvé dans */
                while (getchar( ) != '\n');/* inbuf, donc vide le */
                            /* buffer clavier */
            return(inbuf);
        }

} /* fin de getstring */
```

```
/*********************************************************
**
**    getfloat -affiche l'invite en paramètre suivi d'une virgule
**          et d'un espace, lit un réel sur stdin, puis retire
**          tous les caractères en trop du buffer clavier.
**
**          Si l'utilisateur entre autre chose qu'un nombre
**          getfloat affiche à nouveau l'invite et attend une
**          nouvelle saisie
**
**          La valeur obtenue auprès de l'utilisateur est la valeur
**          retournée par la fonction.
**
**      usage -float x;
**          x = getfloat("Entrer votre balance");
**      remarque -compiler sans linker puis linker avec votre prog.
*/
float  getfloat(char *prompt)
{
        float val;
        int status = 0;

        while (1) {
            printf("%s: ", prompt);
            status = scanf("%f", &val);
            while (getchar( ) != '\n');/* nettoyage des caractères*/
                                       /* laissés dans le buffer clavier */
            if (status)
                break;
            else
                printf("Veuillez reprendre la saisie.\n");
        }
        return(val);

} /* fin getfloat */
```

```
/* Utilisation d'une union */
#include <stdio.h>
#define A 0x41
int main(void)
{
        int index;
        union device_register {
            char byte[4];
            int word;
        } reg;

        printf("\n");
        for (index = 0; index < 4; index++) {
            reg.byte[index] = A + index;
            printf("Octet[%d]: '%c',", index, reg.byte[index]);
            printf(" en hexa : 0x%x.\n", reg.byte[index]);
        } /* fin de for */
        printf("\n");
        printf("Le mot entier : 0x%x.\n", reg.word);
        printf("\n");
        return 0;
} /* fin de main */
```

```
% a.out

Octet[0]: 'A', en hexa : 0x41.
Octet[1]: 'B', en hexa : 0x42.
Octet[2]: 'C', en hexa : 0x43.
Octet[3]: 'D', en hexa : 0x44.

Le mot entier : 0x41424344.

%
```

```
/* Manipulation de membres d'UNION. Ce programme s'appuie sur l'idée
d'une table de symboles d'un compilateur, et les constantes peuvent
être int, float ou  char. Cela pourrait être pratique si tous les
types prenaient le même encombrement. */

#include <stdio.h>
int main(void)
{
        union int_float_ou_char {/* forme de l'union  */
            int   i_val;
            float f_val;
            char c_val;
        };
        union int_float_ou_char value ;/* déclare une variable */
        char c, *ptr;
        int i;
        char value_type;
        for( ; (value_type = getchar( ) ) != '*'; ) {
/* Il est de la responsabilité du programmeur de connaître le type de
la valeur conservée dans l'union int_float_ou_char, en donnant à
value_type une valeur caractérisant ce type. */
            if (value_type == 'i')
                scanf("%d", &value.i_val);
            else if (value_type == 'f')
                scanf("%f", &value.f_val);
            else if(value_type == 'c') {
                c = getchar();
                value.c_val = getchar();
            }
            if (value_type == 'i')
                printf("La valeur est %d \n", value.i_val);
            else if(value_type == 'f')
                printf("La valeur est %f \n", value.f_val);
            else if(value_type == 'c')
                printf("La valeur est %c \n", value.c_val);
            else
                printf("Type %c de value_type erroné\n",
                     value_type);
            c = getchar( );/* retire le newline */
        } /* fin de for */
        return 0;
}
```

```c
/* Utilisation d'une liste simplement chaînée */

#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#define MAXBUF 120
#define PROMPT "Entrer du texte (^D pour quitter):  "
typedef struct node {
        char line [MAXBUF];
        struct node *next;
} node_t;

int main(void)
{
        node_t *head = NULL, *temp;
        char buf [MAXBUF];

        printf (PROMPT);/* construction de la 1ere entrée */
        if (gets(buf)) {
            head = (node_t *)malloc(sizeof(node_t));
            strcpy(head->line, buf);
            head->next = NULL;
            temp = head;
            printf(PROMPT);
        } /* fin de if */

        while (gets(buf)) { /* en faire tant qu'il faut */
            temp->next = (node_t *)malloc(sizeof(node_t));
            temp = temp->next;
            strcpy (temp->line, buf);
            temp->next = NULL;
            printf (PROMPT);
        } /* fin de while */

        printf("\n");
        temp = head;/* affiche toute la liste */
        while (temp != NULL) {
            printf ("%s\n", temp->line);
            temp = temp->next;
        } /* fin de while */
        return 0;
} /* fin de main */
```

```
/* Autre exemple de liste simplement chaînée */

#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#define MAXBUF 120
#define PROMPT "Entrer du texte (^D pour quitter): "
typedef struct node {
        char line [MAXBUF];
        struct node *next;
        } node_t;
int main(void)
{
        node_t head, *temp;/* remarque:head n'est pas un pointeur */
        char buf [MAXBUF];

        head.next = NULL;/* Dans ce cas il n'y a pas de code  */
        temp = &head;   /* spécial au premier élément */
        printf(PROMPT);
        while (gets(buf)) { /* ajouter tant qu'il faut */
            temp->next = (node_t *)malloc(sizeof(node_t));
            temp = temp->next;
            strcpy (temp->line, buf);
            temp->next = NULL;
            printf(PROMPT);
        } /* fin de while */

        printf("\n");
        temp = head.next;/* ATTENTION : différence avec l'exemple
                            précédent */
        while (temp != NULL) {
            printf("%s\n", temp->line);
            temp = temp->next;
        } /* fin de while */
        return 0;
} /* fin de main */
```

```
/* Déclaration d'un structure de liste chaînée. Au fur et à mesure de
la saisie de nouveaux noms, la mémoire est allouée puis ajoutée à la
structure existante.  */

#define EOS 0
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
        typedef struct person {
            struct person *link;
            char name[32];
        } Node, *Nodeptr;
        Nodeptr current, start, last;
        char temp_name[32];
        /* Saisie du 1er nom... */
        printf("Entrer un nom suivi de entrée : ");
        gets(temp_name);
        start = NULL;   /* La liste n'est pas encore commencée */
        while (temp_name[0] != EOS) {
            /* allocation de l'espace pour un nouveau nom */
            current = (Nodeptr)malloc(sizeof(Node));
            if (start == NULL) {                /*Si 1er de liste*/
                start = current;            /* début de liste */
                strcpy(current->name, temp_name);  /* met en liste */
            } else {                        /* liste déjà faite */
                strcpy(current->name, temp_name);  /* met en liste */
                last->link = current;       /* pointe plus loin */
            }
            last = current;             /* mémorise le dernier */
            printf("Entrer un nom suivi de entrée : ");
            gets(temp_name);
        } /* fin de while */
        current->link = NULL;   /* fin de liste */
        return 0;
}
```

```
/* Utilisation de strspn(), strpbrk(), strtok() */

#include <stdio.h>
#include <string.h> /* déclaration de strspn(),strpbrk(), strtok()*/
#define NUMS "0123456789"
int main(void) {
        static char s3[ ]="ces/mots/sont/des/tokens";
        char s1[255], *s2;
        int len;
        printf("Entrer une ligne suivie de entrée :  ");
        gets(s1);
        if (strspn(s1, NUMS) == strlen(s1))
            printf("Une ligne que de chiffres !\n");
        if ((s2 = strpbrk(s1, "~@#*&")) != NULL)
            printf("\"%s\" débute à s1[%d].\n",s2,
                        (strlen(s1)-strlen(s2)));
        s2 = strtok(s3, "/");
        while ((s2 != (char *)NULL)) {
            printf("%s\n", s2);
            s2 = strtok((char *)NULL, "/");
        } /* fin de while */
        return 0;
} /* fin de main */
```

```
% a.out
Entrer un ligne suivie de entrée : This is a str@ing I'm typing
"@ing I'm typing" débute à s1[13].
ces
mots
sont
des
tokens
%  a.out
Entrer un ligne suivie de entrée : 895720659237810392106
Une ligne que de chiffres !
ces
mots
sont
des
tokens
%
```

```c
/* Utilisation de pointeurs sur fonction */

#include <stdio.h>
#define MAXFA 5
int main(void)
{
    int c = 1;
    int f1(void), f2(void), f3(void), f4(void);
    int (*fa[MAXFA])();/* tableau de pointeur sur fonctions */

/*Renseigne 'fa' avec l'adresse des 4 fonctions. Aucune parenthèse
n'apparaît : le nom d'une fonction est l'adresse de la fonction */

    fa[0] = 0;
    fa[1] = f1;
    fa[2] = f2;
    fa[3] = f3;
    fa[4] = f4;

    srandom(time(0)%getpid());/*initialise le générateur random()*/
    while (c = fa[c]());/* tant que retour de f. diff. de 0 */
    return 0;
}
int f1(void) {
    int x = random() % MAXFA;/* nombre aléatoire entre 0 et 4 */
    printf("f1: %d\n", x);/*afficher le nombre aléatoire */
    return(x);      /* retourne le nombre aléatoire entre 0 et 4 */
}

int f2(void) {
    int x = random() % MAXFA;
    printf("f2: %d\n", x);
    return(x);
}

int f3(void) {
    int x = random() % MAXFA;
    printf("f3: %d\n", x);
    return(x);
}

int f4(void) {
    int x = random() % MAXFA;
    printf("f4: %d\n", x);
    return(x);
}
```

```
/*  Démonstration des classes d'allocation...     */


#include <stdio.h>
#define NL putchar('\n')

int i = 1;      /* i est global au programme et initialisée à 1 */
int main(void) {
    auto int j; /* j est local à main( ) */
    static int next(void);/* visible seulement dans le fichier */
    int new(int );
    for (j = 1; j <= 3; j++) {
        printf("ici next( ) %d", next( ) ); NL;
        printf("ici new( ) %d", new(i + j)); NL;
    }
    return 0;
}

static int next(void)
{
     return (i += 1);
}

int new(int k)
{
     return(k += i);
}
```

```
/* Exemple d'appel par référence.  Le programme donne des valeurs à
i et j, puis appelle une fonction qui assure que i est inférieur à j */

#include <stdio.h>
void main(void)
{
        int i = 9, j = 7;
        int order(int *, int *);
        order(&i, &j);/* passe les adresses */
        printf("i est maintenant %d et j %d \n", i, j);
}



int order(int *p, int *q)
{
        int temp;/* compare les valeurs pointées  */
        if (*p > *q)/* si celle pointée par p est supérieure à q,
                       échanger */
        {
            temp = *p;
            *p = *q;
            *q = temp;
        }
}
```

```
/* Exemple de tableau en argument de fonction. Ensuite, 2 fonctions
qui renvoient la somme de tableaux de réels. Le tableau est le 1 er
argument et le nombre d'éléments est le 2 ème. */
#include <stdio.h>
int main(void)
{
    float f_arr[100], f_sum, sum_1(float [], int), sum_2(float *, int
);
    int num, i = 0;

    puts("Combien de réels à ajouter ?");
    scanf("%d", &num);
    while (i < num) {
        scanf("%f", &f_arr[i]);
        i++;
    }
    if (num < 50)
        f_sum = sum_1(f_arr, num);
    else
        f_sum = sum_2(f_arr, num);
    printf("La somme des nombres est %9.6f \n", f_sum);
    return 0;
}


float sum_1(float array[], int n)/* notation indicée */
{
    int i;
    float sum = 0;

    for (i = 0; i < n; ++i)
        sum += array[i];
    return(sum);
}

float sum_2(float *ap, int n)/* notation pointeur */
{
    int i;
    float sum = 0;

    for (i = 0; i < n; ++i)
        sum += *(ap + i);
    return(sum);
}
```

```
/* Exemple d'appel par référence avec des tableaux en argument. Deux
fonctions STRCAT() et  STRCPY() concatènent 2 chaînes. */

#include <string.h> /* déclaration de strcpy() et strcat() */
#define MAXLEN 100
int main(void)
{
        char first[MAXLEN], second[MAXLEN], result[MAXLEN];

        printf("Entrer un mot : ");
        gets(first);
        printf("Entrer un autre mot : ");
        gets(second);
        puts("\n \n");
        strcpy(result, first);/* copie 'first' dans 'result' */
        strcat(result, second);/* ajouter 'second' à 'result' */
        printf("Première chaîne : %s \n", first);
        printf("Deuxième chaîne : %s \n", second);
        printf("Les deux jointes : %s \n", result);
        return 0;
}


void strcpy(char *to, char *from)/* adresses origines */
{
        while (*to++ = *from++);/* affectation puis incrément */
}


void strcat(char *to, char *from)/* concaténation */
{
        /* aller à la fin de la chaîne 'to'  */
        while (*to) to++;/* compte chaque caractère */
                /* jusqu'au caractère nul */
        /* copie le contenu de 'from' à la fin de 'to' */
        do {
            *to++ = *from;/* à la fin de 'to', affectation de   */
        } while (*from++);/* chaque car. de 'from', jusqu'à la */
                    /* rencontre du caractère nul */
}
```

```
/* Création et tri d'un tableau de pointeurs. Ce programme lit un
nombre de mots et les range à la suite dans un tableau. Puis le tableau
est trié, et la liste des mots ainsi triés est affichée. */
#include <stdio.h>
#define MAXWORDS 100
#define MAXSPACE 3000
int main(void) {
        char *p[MAXWORDS];/* tableau de pointeurs sur char  */
        char w[MAXSPACE];
        char *q = w;/* initialise q sur l'origine de w */
        int i, n;
        void bubble(char *[], int );
        printf("\n Combien de mots à trier ? ");
        scanf("%d", &n);
        if (n <= MAXWORDS) {
            printf("\n Entrer les %d mots : ", n);
            for (i = 0; i < n; ++i) {
/* puisque w[MAXSPACE] est fixe, q représente l'adresse courante dans
w[MAXSPACE]. Chaque mot lu est rangé à partir de q, puis après calcul
de sa longueur, ajout de 1 pour '\0' et incrément de q  */
                scanf("%s", p[i] = q);
                q += strlen(q) + 1;
            } /* fin de for */
            bubble(p, n);
            printf("\n %14s", "liste triée : ");
            for (i = 0; i < n; ++i)
                printf("%s \n %14s", p[i], "");
            printf("\n");
        } else
            printf("\n \nTrop de mots : %d maximums admis",
                    MAXWORDS);
        return 0;
} /* fin de main */
void bubble(char *p[], int n)/* tableau de pointeurs sur char */
{
        char *temp;
        int i, j;

        for (i = 0; i < n - 1; ++i)/* boucle for sur n mots */
            for (j = n - 1; i < j; --j)/* boucle pour comparer 2 */
                if (strcmp(p[j-1], p[j]) > 0) {/* mots adjacents */
                    temp = p[j-1];  /* échange */
                    p[j-1] = p[j];
                    p[j] = temp;
    }
}
```

```c
/* Entrée/Sortie caractère/caractère sur fichier. Copie un fichier
dans un autre. */

#include <stdio.h>
int main(void)
{
        int c;
        FILE *fp1, *fp2;/* Pointeurs de fichier */
        char name1[32], name2[32];

        puts("Entrer les noms des 2 fichiers sur 2 lignes. \n");
        gets(name1);/* saisie des noms de fichiers */
        gets(name2);
        fp1 = fopen(name1, "r");/* ouverture pour lecture */
        fp2 = fopen(name2, "w");/* ouverture pour écriture */
        while((c = getc(fp1)) != EOF )/* lit un char */
            putc(c, fp2);/* écrit un char   */
        fclose(fp1);    /* fermeture des fichiers */
        fclose(fp2);
        return 0;
}
```

```
/* Arguments passés sur la ligne de commande. Ce programme ouvre un
fichier puis lit chaque ligne comme une chaîne de caractères */

#include <stdio.h>
int main(int argc, char *argv)
{
    char record[128];
    FILE *filename;/* pointeur vers une structure FILE */

    puts("\n \n \n \t \t \t Lecture d'un fichier ASCII \n");
    if (argc != 2) {/* vérifie le nombre des arguments */
        puts("Entrer le nom du fichier sur la ligne de commande \n");
        exit(1);/* sortie du programme  */
    }
    /* ouvre un fichier ASCII en lecture, sortie du prog. si erreur */
    if (!(filename = fopen(argv[1], "r"))) {
        fprintf(stderr, "Erreur d'ouverture de %s \n", argv[1]);
        exit(1);
    }
    /* Lit une ligne à concurrence de 128 caractères */
    while (fgets(record, 128, filename) != 0)
        printf("%s", record);
    fclose(filename);
    return 0;
}
```

```
/* Exemple d'utilisation de variables structures. */


#include <stdio.h>

struct pencil {/* nom de structure (comme type) */
        int hardness;
        char maker;
        int number;
};
int main(void)
{
    struct pencil p[4];/* tableau de structures */
    struct pencil *pen_ptr;/* pointeur sur structure */

    p[0].hardness = 2;/* initialisation des membres de structures */
    p[0].maker = 'F';
    p[0].number = 482;
    p[1].hardness = 0;
    p[1].maker = 'G';
    p[1].number = 33;
    p[2].hardness = 3;
    p[2].maker = 'E';
    p[2].number = 107;
    p[3] = p[2];
    printf("   Numéro de fabricant de harnais \n\n");
/* initialise le pointeur pen_ptr au début du tableau, teste la fin
   et incrémente d'élément en élément */
    for (pen_ptr = p; pen_ptr <= p + 3; ++pen_ptr)
    /* imprime chaque membre... */
        printf("       %d         %c       %d\n",
            pen_ptr -> hardness, pen_ptr -> maker,
            pen_ptr -> number);
    return 0;
}
```

```
/* Exemple d'utilisation de champs de bits. */

#include <stdio.h>

struct  word_bytes  {
        unsigned int byte0 : 8,
        byte1 : 8,
        byte2 : 8,
        byte3 : 10;
  };
int main(void)
{
        struct word_bytes y;

        y.byte0 = 128;
        y.byte1 = 129;
        y.byte2 = 130;
        y.byte3 = 131;
        printf("%u \n", y.byte0);
        return 0;
}
```

```c
/* Exemple de déclaration de nouveau type par typedef. Le nouveau type
est utilisé pour déclarer a et b. Les valeurs des tableaux a et b sont
utilisées pour calculer celles de c. */

#include <stdio.h>
#define N 3
/* initialisation de matrices 3X3...*/
typedef int MATRIX;
MATRIX a[N][N] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
};
MATRIX b[N][N] = {
        {10, 20, 30},
        {10, 20, 30},
        {10, 20, 30}
};
MATRIX c[N][N];

int main(void)
{
    int i, j, k;

    for (i=0; i < N; ++i)
        for (j = 0; j < N; ++j) {
            c[i][j] = a[i][j] * b[i][j];
            printf("valeur de c[%d][%d] = %d \n", i, j, c[i][j]);
        }
    return 0;
}
```

```
/* Exemple de conversion explicite (CAST) pour un argument de la
fonction sqrt. */

#include <stdio.h>
#include <math.h>/* déclaration de la fonction sqrt */
#define BEGIN 1
#define END 20
int main(void)
{
        int count;
        double root;
        int square;

        printf("\t Table des carrés et des racines carrées \n \n");
        for (count = BEGIN; count <= END; count++) {
            square = count * count;/* élévation au carré */
            root = sqrt( (double) count);/*racine carrée */
                /* sqrt() demande un argument */
                /* de type double   */
            printf("Nombre : %d \t carré: %d \t racine: %10.3f \n"
                                    , count, square, root);
        }
        return 0;
}
```

```c
/* Manipulation de bits. */

#include <stdio.h>
int main(void)
{
        struct pcard {
            unsigned values : 4;
            unsigned suit : 2;
        } hand;
        void bit_print(int);

        printf("%d %d \n",(hand.values = 1), (hand.suit = 2));
        bit_print(hand.values);
        bit_print(hand.suit);
        return 0;
}

void bit_print(int v)
{
        int i, mask = 1;

        mask <<= 31;/* décalage à gauche de 31 bits  */
        for (i = 1; i <=32; ++i) {
            putchar(((v & mask) == 0) ? '0' : '1');
            v <<= 1;
        }
        putchar('\n');
}
```

```
/* Utilisation de calloc() pour l'espace alloué dynamiquement pour la
chaîne newstr. */

#include <stdio.h>
#include <stdlib.h>
int main(void)
{
        int i,  nelem;
        char *newstr;

        puts("Entrer le nombre de caractères : ");
        scanf("%d", &nelem);
        newstr = calloc(nelem, sizeof(char));
        puts("Entrer une chaîne de caractères :");
        scanf("%s", newstr);
        for (i = 0; i < nelem; ++i)
            printf("%c", newstr[i]);
        putchar('\n');
        free(newstr);/* rend la mémoire allouée au système */
        return 0;
}
```

Internationalisation, Grands Caractères et Caractères Multi-octets

Cette annexe donne des informations sur :

- Les jeux de caractères
- Idéogrammes et caractères multioctets
- Grands caractères et constantes chaînes de grands caractères
- Fonctions sur les chaînes et caractères multioctets
- Caractères nationaux et répertoires associés
- Exemple simple

Jeux de caractères

Le code de caractères utilisé généralement pour l'anglais est le code ASCII. Ce code est basé sur des caractères pouvant être codés sur 7 bits. Beaucoup de caractères utilisés dans les langages asiatiques et d'autres langages différents de l'anglais, nécessitent un codage sur plus d'un seul octet.

Comme les caractères sont représentés par des nombres, un type entier assez grand pour contenir les valeurs numériques des caractères est indispensable. L'ANSI C définit le type fondamental `wchar_t` (wide character type), dans `<stddef.h>`. C'est un `typedef` sur un type assez grand pour le plus grand des jeux de caractères supportés. (En Sun ANSI C, `wchar_t` est un `typedef` sur `long` (4 octets).)

Idéogrammes et Caractères Multioctets

Les langues asiatiques comportent beaucoup d'*idéogrammes*. Les idéogrammes sont codés sur des séquences d'octets, et le procédé de codage doit être capable d'identifier la séquence d'octets comme un idéogramme en particulier. L'ANSI C utilise *des caractères multioctets* pour les séquences d'octets des idéogrammes. Les caractères habituels sur un octets sont traités comme des cas particuliers de caractères multioctets.

Constantes grands caractères et chaînes de grands caractères

Une *constante grand caractère* est indiquée en faisant précéder une constante entre quotes de la lettre *L* :

```
L'x'
L'abc'
```

De même, une *constante chaîne de grands caractères* est signalée par la lettre *L* devant les guillemets, comme : `L"abcxyz"`. De telles constantes peuvent servir à initialiser un tableau de grands caractères (un tableau de `wchar_t`) :

```
wchar_t  x[]=L"abcxyz";
```

Equivalent à :

```
wchar_t x[]={L'a', L'b', L'c', L'x', L'y', L'z', 0};
```

Remarquer que le tableau contient un 0 à la fin (comme avec une chaîne de caractères habituelle qui rajoute '\0'.)

Un grand caractère a un codage externe multioctets et un codage interne en `wchar_t`.

Identité

Pour chaque grand caractère il existe un code multioctets correspondant et pour chaque caractère multioctet (comme défini dans le jeu de caractères), il existe un grand caractère correspondant. Egalement, chaque caractère multioctets a une valeur représentable en `wchar_t`.

La valeur d'un caractère multioctets correspondant à un caractère monooctet doit être la même que dans le code sur un seul octet. Le nombre d'octets utilisé pour représenter un grand caractère ne peut être supérieur à `MB_LEN_MAX` (5 pour le Sun ANSI C).

Fonctions sur les Caractères Multioctets

Les fonctions de la librairie de l'ANSI C traitant les grands caractères sont : `mblen()`, `mbtowc()`, et `wctomb()`. La signification et la syntaxe de ces fonctions sont les suivantes :

```
#include <stdlib.h>
#include <limits.h>
int mblen(const char *s, size_t n)
```

Retourne le nombre de caractères contenus dans le caractère multioctets pointé par *s*, si *s* n'est pas nul et si les *n* octets suivants ou moins forment un caractère multioctets valide (défini).

#include <stdlib.h>
#include <limits.h>
int mbtowc(wchar_t *pwc, const char *s, size_t n)

Retourne le nombre d'octets du caractère multioctets pointé par *s*, si *s* n'est pas nul ; en plus, la valeur numérique de type `wchar_t` qui correspond au caractère est déterminée (voir "Identité" ci-dessus). Si

pwc n'est pas nul, et que le caractère est valide, le code numérique est placé dans l'objet pointé par *pwc*. La fonction `mbtowc()` examine au plus *n* octets du tableau pointé par *s*.

Cette fonction ne retourne pas de valeur supérieure à *n* ni à `MB_CUR_MAX`.

```
#include <stdlib.h>
#include <limits.h>
int wctomb(char *s, wchar_t wchar)
```

`wctomb()` retourne le nombre d'octets nécessaires pour représenter le caractère correspondant à *wchar* (voir également "Identité"). La fonction stocke la représentation multioctets dans le tableau pointé par *s*, si *s* est non-nul.

La valeur retournée par `wctomb()` ne dépasse jamais `MB_CUR_MAX`.

Fonctions Chaînes Multioctets

Les deux fonctions supportant les constantes chaînes de grands caractères sont `mbstowcs()` et `wcstombs()`. Leurs syntaxes et significations sont :

#include <stdlib.h>
size_t mbstowcs(wchar_t *pwcs, char *s,size_t n)

Cette fonction prend la série de caractères multioctets pointée par *s* et la convertie dans la série de codes numériques correspondante. Les codes sont placés dans le tableau pointé par *pwcs*. Pas plus de *n* codes sont stockés dans le tableau. Les caractères après le caratère nul ne sont pas traités. `mbstowcs()` retourne le nombre d'éléments modifiés ou (size_t) -1 si un caractère invalide est trouvé.

```
#include <stdlib.h>
size_t wcstombs(char *s,wchar_t *pwcs,size_t n)
```

`wcstombs()` prend la série de codes numériques pointée par *pwcs* et la convertit dans la séquence de caractères multioctets correspondante. Les caractères sont placés dans le tableau pointé par *s*. Pas plus de *n* caractères sont mis dans le tableau.

Si un code sans correspondance est trouvé, la fonction retourne (size_t) -1. Sinon, c'est le nombre d'octets modifiés qui est retourné.

Local

Un *local* est un modèle ou une définition d'un environnement de langue native. Chaque programme d'application s'exécute dans le local du programme. Ce local définit le *jeu de caractères* (vu précédemment), les conventions de formatage de date et d'heure, le formatage monétaire et décimal, et l'ordre de classement. Si un programme utilise certaines fonctions (que nous verrons plus loin) qui dépendent d'une de ces conventions de format, le programmeur doit prendre en considération le local dans lequel le programme est utilisé.

Les catégories qui définissent le local d'un programme sont les suivantes :

LC_CTYPE - contrôle le comportement des fonctions caractères (toutes celles définies dans `ctype(3)`) et des fonctions sur les caractères multioctets.

LC_TIME - contrôle le format date et heure de la fonction `strftime()`.

LC_MONETARY - contrôle le format monétaire et les valeurs retournées par `localeconv()`.

LC_NUMERIC - affecte le signe décimal pour les fonctions de lecture et d'écriture, et pour les fonctions de conversion.

LC_COLLATE - affecte les fonctions `strcoll()` et `strxfrm()`.

LC_MESSAGES - affecte `gettxt()`, `catopen()`, `catclose()`, et `catgets()`.

LC_ALL - nomme le local de tout le programme.

La fonction `setlocale()` peut être utilisée pour demander ou positionner le local d'un programme dans une catégorie. Syntaxe :

```
#include <locale.h>
char *setlocale(int category, const char *locale)
```

Une valeur "C" pour *locale* spécifie l'environnement par défaut - l'environnement minimal pour les traductions C ; si la valeur de *locale* est la chaîne vide "", le local est pris dans la variable d'environnement dont le nom correspond à la catégorie. Si aucune variable n'existe pour la catégorie, la variable LANG est vérifiée.

Si un programmeur désire que son programme suive les *conventions* du pays où le programme sera utilisé, les instructions suivantes doivent apparaître au début du code :

```
#include <locale.h>
...
setlocale(LC_ALL,"");/* toutes valeurs par défaut */
```

Un pointeur nul par l'argument *locale* dans `setlocale()`, comme ,

```
setlocale(category, (char *) 0);
```

demande à la fonction de renvoyer la valeur de local pour la *category* spécifiée. Ceci est un moyen pour le programmeur de connaître le local courant d'une catégorie.

Au début d'un programme, l'équivalent d'un

```
setlocale(LC_ALL, "C");
```

est exécuté.

Répertoires Associés au Local

Chaque catégorie correspond à un ensemble de fichiers qui contiennent les informations définissant un local. L'emplacement de ces fichiers est :
`/usr/lib/locale/`*`locale_country/category/db_file_name`*.
Par exemple, le fichier pour la catégorie `LC_NUMERIC` du local "french" se situe sous `/usr/lib/locale/fr/LC_NUMERIC`.

Habituellement, l'implémentation fournit les fichiers de description des différents "local". Mais un programmeur peut définir lui-même un local. Voir `man chrtbl` pour plus de détails.

Exemple Simple

Cet exemple montre comment définir un fichier de description pour la catégorie `LC_CTYPE` et comment l'installer dans le bon répertoire. Puis un programme exemple montre comment utiliser ce nouveau fichier de description.

La commande `chrtbl(1M)` est utilisée pour créer un fichier décrivant les informations pour tous les jeux de caractères tenant sur un octet (7-bit et 8-bit) (un fichier `LC_CTYPE`). Il indique comment déterminer si un caractère est majuscule, minuscule, chiffre, ponctuation, espace, caractère de contrôle ou héxadécimal. Il contient également une table de conversion majuscules-minuscules. (La commande `chrtbl` crée aussi un fichier `LC_NUMERIC`, non-abordé dans cet exemple.) Les pages du `man` pour `chrtbl(1M)` décrivent le format du fichier de spécification. Ce fichier est fourni en argument de `chrtbl`.

Le programme exemple utilise les classifications suivantes :

Majuscules :	Å Ë Ø Ç Ñ
Minuscules :	å ë ø ç ñ
Chiffres :	**0123456789**
Hexa :	**0123456789abcdefABCDEF**
Espaces :	**Barre d'espace et Inséquable**
Blancs :	**Barre d'espace**
Ponctuation :	**! " # $ % & ' ( ) * + , - . /** **: ; < = > ? @ [\ ] ^ { | } ~**
Control Ch. Codes:	**000 - 037 0177**

Remarque : pour obtenir ces caractères, utiliser la touche compose (clavier type 4) comme indiqué ci-dessous :

TOUCHES	DESCRIPTION
compose * A	A Angström
compose " E	E tréma
compose / O	O barré
compose , C	C cédille
compose - D	eth islandais majuscule
compose ~ N	N tilde
compose H T	thorn islandais majuscule

Remarque : Les minuscules s'obtiennent de la même façon, en remplaçant les majuscules par des minuscules.

Pour créer le fichier de description `LC_CTYPE` avec la commande `chrtbl`, un fichier de spécification doit d'abord être créé. Ce fichier est listé ci-dessous. Le fichier de description est créé dans le répertoire courant, au moment de l'exécution de la commande `chrtbl` et son nom est `sample_ch`. Le fichier de spécification, `sample_ch_sp`, pour le jeu de caractères cité est :

```
LC_TYPE         sample_ch

isupper         0305  0313  0330  0307  0320  0321  0336
islower         0345  0353  0370  0347  0360  0361  0376
isdigit         060-071
isspace         040  0240
ispunct         041-057  072-0100  0133-0140  0173-0176
iscntrl         000-037  0177
isblank         040  0240
isxdigit        060-071  0141-0146  0101-0106

ul      <0305 0345>  <0313 0353>  <0330 0370>  <0307 0347> \
        <0320 0360>  <0321 0361>  <0336 0376>
```

Une fois le fichier créé, lancer la commande :

% **chrtbl sample_ch_sp**

Le système va générer le fichier de description `LC_CTYPE` d'après le fichier de spécification, avec le nom indiqué sur la ligne `LC_CTYPE`. Une fois créé, placer ce fichier dans la directory "local" concernée.

Dans l'exemple, l'administrateur va créer un répertoire dans `/usr/lib/locale`. Ce nouveau répertoire sera appelé sample_ch. Le répertoire `LC_CTYPE` sera sous `sample_ch`. Le fichier de description `sample_ch` sera placé dans ce sous-répertoire avec le nom complet `/usr/lib/sample_ch/LC_CTYPE/ctype`.

Maintenant, voici un programme `locale.c` qui change le local avec `setlocale()` pour utiliser le nouveau fichier de description :

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

int main(void)
{
    char *locale;
    char save_locale[100];
    int ch;
    locale = setlocale(LC_CTYPE,(char *)0);
    printf("\nLocal courant : %s\n\n", locale);
    strcpy(save_locale,locale);
    locale = setlocale(LC_CTYPE, "sample_ch");
    printf("Nouveau local : %s\n\n",locale);
    printf("Entrer des majuscules, (ctrl-d pour quitter) : \n");
    while ((ch=getchar()) != EOF){
        if (isupper(ch))
            printf("%c minuscule = %c\n",ch, tolower(ch));
        else if ((ch == '\n') || (ch == ' '))  /* newline ou espace */
            continue;
        else
            printf("%c n'est pas une majuscule\n", ch);
    }
    locale=setlocale(LC_CTYPE, save_locale);
    printf("\nLe local est remis à %s\n", locale);
}
```

Compiler avec la commande :

cc -Xc locale.c -o locale

Et lancer le programme pour obtenir :

```
% locale
Local courant : C
Nouveau local : sample_ch
Entrer des majuscules, (Ctrl-d pour quitter) :
A
A n’est pas une majuscule
B
B n’est pas une majuscule
29
2 n’est pas une majuscule
9 n’est pas une majuscule
a d
a n’est pas une majuscule
d n’est pas une majuscule
Å
Å minuscule = å

  minuscule =
  minuscule =
Ç
Ç minuscule = ç
Ñ
Ñ minuscule = ñ
å
å n’est pas une majuscule
Ø
Ø minuscule = ø

Le local est remis à C
%
```

Différences Entre Sun C et Sun ANSI C

La plupart de ces informations proviennent de l'Annexe A de la documentation "*SPARCompilers C 2.0 Transition Guide*".

Mots-Clef

`const`, `volatile`, et `signed` sont des mots-clef en ANSI C.
Ces mots sont traités comme des identifiants en Sun C.

`asm` est un mot-clef du Sun C. Il est traité comme un identifiant en ANSI C et en Sun ANSI C dans le mode *-Xc*.

Identifiants

ANSI C n'autorise pas le dollar ($) dans les identifiants.
Sun C le permet.

Long Float

Sun C accepte les déclarations de `long float` et le traite en `double`.
ANSI C refuse ces déclarations.

Constantes Caractère Multioctets

Sun C et ANSI C représentent les caractères multioctets différemment :

```
int i, mc = 'abcd';
char *cptr;
cptr = (char *)&mc;
   for (i = 0; i < 4; i++){
      printf("%c", *cptr);
      cptr++;
   }
```

donne :

```
abcd    /* en Sun C */
dcba    /* en ANSI C */
```

Constantes Entières

Sun C accepte `8` ou `9` dans une séquence d'escape octale ; ANSI C non :

```
char x, y;
x = '\78';  /* Sun C interprète en \70 */
y = '\79';  /* Sun C interprète en \71 */
```

Opérateurs d'Affectation

Sun C permet les espaces entre les paires suivantes, en les traitant comme deux marqueurs, alors que l'ANSI C les traite comme un seul marqueur. Aussi, ANSI C n'autorise pas les espaces (et génère un core si besoin).

*=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=

Calculs en Simple/Double Précision

Sun C promouvoit les opérandes d'une expression virgule flottante en `double`.
ANSI C permet les calculs sur les flottants en *simple* précision.

En Sun C les fonctions qui renvoient un `float` ont leur code de retour promu en `double`.
ANSI C requiert des valeurs en `float` pour ces fonctions.

Conservation du Signe ou de la Valeur

Sun C supporte *la conservation du signe*. Ainsi les `unsigned char` et `unsigned short` sont convertis en `unsigned int`.

ANSI C *conserve la valeur*. Les `unsigned char` et `unsigned short` sont convertis en `signed int` si ce type peut représenter toutes les valeurs du type d'origine ; sinon, ils sont convertis en `unsigned int`.

Convertion Explicite (Cast) d'une *lvalue*

Sun C supporte la convertion explicite des *lvalue(s)* :

```
(char *)ip = &char;
```

ANSI C ne l'accepte pas.

Déclarations `int` Implicites

Le compilateur Sun C accepte les déclarations sans type :

```
num;    /* implique :  int num;  */
```

ANSI C génère une erreur de syntaxe dans le cas ci-dessus.

Déclarations vides

Sun C accepte les déclarations vides :

```
int;
```

Sauf pour les *tags* de prototypage, ANSI C n'accepte pas les déclarations vides.

Spécifications de type sur des Typedefs

Le compilateur Sun C permet les spécifications de types comme `unsigned`, `short`, et `long` sur des déclarations de `typedef` :

```
typedef short small;
unsigned small x;
```

ANSI C ne permet pas la modification de déclarations `typedef` avec des spécifications de type. On obtient le message suivant à la compilation :

```
identifier redeclared: small
syntax error before or at: x
```

Types autorisés sur les champs de bits

Sun C permet tous les types entiers sur les champs de bit, et même des champs sans nom.
ANSI C ne supporte que les types `int`, `unsigned int`, et `signed int`. Les autres types sont indéfinis.

Type de la Condition d'un `switch`

Sun C permet `float` et `double` comme types de condition d'un `switch` en les convertissant en `int` :

```
main()
{
     float y = 4.3213;
     switch (y)
     {
          ...
     }
}
```

ANSI C ne permet que des entiers dans un `switch` : `int`, `char`, et `enum`. L'exemple génère une erreur de compilation avec ANSI C.

Directives du préprocesseur `#else` et `#endif`

Le préprocesseur du Sun C ignore tout marqueur après les directives `#else` ou `#endif` :

```
#ifdef TEST
#define YES_TEST 1
#else  else of test  /* 'else of test' est
            ignoré sans message de compilation*/
#define YES_TEST 0
#endif  test /* 'test' est ignoré */
```

ANSI C ne permet plus de marqueur après les directives du préprocesseur.

Collage de paramètres de macros et l'opérateur

En Sun C, le collage de paramètres se réalise avec un commentaire :

```
#define PASTE(A,B)    A/* commentaire */B
```

En ANSI C, le commentaire étant comme un blanc, il faut utiliser ## :

```
#define  PASTE(A,B)  A##B
```

Récursivité du Préprocesseur

Le préprocesseur du Sun C réalise des substitutions récursives.
En ANSI C, une macro n'est plus substituée si elle apparaît déjà dans la liste des remplacements effectués :

```
#define F(X)   X(arg)
F(F)
```

donne :

```
arg(arg)  /* Sun C */
F(arg)    /* ANSI C */
```

Substitution de Paramètre caractère dans les Macros

Le préprocesseur du Sun C substitue les caractères dans une constante si celle-ci correspond à une macro :

```
#define charize(c) 'c'
charize(Z)
```

donne :

```
'Z'
```

En ANSI C, le caractère n'est pas remplacé et il n'y a pas d'opération équivalente. La solution est d'utiliser la macro suivante :

```
#define charize(c) (c)
charize('Z')
```

Substitution de Paramètres chaîne dans les Macros

Le préprocesseur du Sun C remplace un paramètre apparaissant dans une chaîne de la définition de la macro :

```
#define str(a) "a!"
str(x y)
```

donne :

```
"x y!"   /* en Sun C */
"a!"     /* en ANSI C */
```

En ANSI C, l'opérateur **#** doit être utilisé. L'exemple s'écrit alors :

```
#define str(a) #a "!"
str(x y) /* donne "x y!" après substitution */
```

Noms `typedef` en paramètre de fontion

Sun C permet l'utilisation de noms `typedef` dans les paramètres d'une fonction, ce qui en fait, *cache* la déclaration `typedef`. ANSI C ne le permet pas.

Initialisation des agrégats propre à l'implémentation

Sun C utilise un algorithme de *bas-en-haut* pour l'analyse des initialisations partiellement elidées. ANSI C recommande une analyse *descendante*. Exemple :

```
struct {
int a[3];
int b;
}  w[] = { {1}, 2 };
```

donne en Sun C :

sizeof(w) = 16
w[0].a = 1, 0, 0
w[0].b = 2

et en ANSI C :

```
sizeof(w) = 32
w[0].a = 1, 0, 0
w[0].b = 0
w[1].a = 2, 0, 0
w[1].b = 0
```

Visibilité des fonctions `extern` et `static` d'un bloc

Le standard ANSI ne garantit pas la visibilité dans tout le fichier de fonctions déclarées dans un bloc, comme en Sun C.

L'Argument *`envp`* de *`main()`*

Sun C prend *`envp`* en troisième argument de `main()`.
Sun ANSI C permet aussi cet argument, mais ce n'est pas conforme au standard ANSI du C.

Concatènation de ligne avec Backslash

L'ANSI C concatène les lignes se terminant par un backslash (\) (suivi immédiatement par newline) avec la ligne suivante.
Sun C ne le fait pas.

Les Trigraphes dans les Chaînes de Caractères

Le Sun C ne supporte pas cette caractéristique de l'ANSI C. Il y a neuf séquences trigraphes qui sont :

trigraphe	car.	trigraphe	car.
??=	#	??>	}
??-	~	??'	^
??(	[	??!	|
??)	]	??/	\
??<	{		

Types

Le type ANSI `long double` n'existe pas en Sun C.

Constantes Virgule Flottante

Les suffixes ANSI : f, l, F et L ne sont pas reconnus en Sun C.

Les Constantes Entières peuvent avoir différents Types

Les suffixes U et u de constantes ANSI n'existent pas en Sun C.
En ANSI C, les constantes sans suffixe peuvent être non-signées et ainsi impliquer une arithmétique non-signée des expressions dans lesquelles elles apparaissent.

Constantes Caratères Larges

Le Sun C ne reconnaît pas la syntaxe ANSI des caractères larges (préfixe *L*) :

```
wchar_t   wc = L'x';
```

Constantes Caractères

L'ANSI C traite '\a' (bell) et '\x' (chiffre hexadécimal) comme des séquences spéciales. En Sun C ces séquences sont interprétées comme 'a' et 'x'. Exemple :

```
char bell = '\a';
printf("Ding Dong ! %c\n", bell);
printf("L'hexa f en base 10 vaut : %d\n",
 '\xf');
```

donne en Sun C :

```
Ding Dong ! /* pas de beep */
L'hexa f en base 10 vaut : 30822
```

et en ANSI C :

```
Ding Dong ! /* on entend un beep ! */
L'hexa f en base 10 vaut : 15
```

Chaînes Adjacentes

Le Sun C ne concatène pas les chaînes adjacentes comme l'ANSI C :

```
printf("Cette chaîne sera concaténée avec\n"
" celle là pour permettre de fractionner\n"
" de longues chaînes comme ça pour une\n"
" meilleure lisibilité\n");
```

Les Chaînes de Grands Caractères

La syntaxe ANSI n'est pas supportée en Sun C :

```
wchar_t    *ws = L"hello" ; /* ANSI seulement */
```

Pointeurs : `void *` et `char *`

Le pointeur ANSI `void *` existe aussi en Sun C.

L'Opérateur Plus Unaire : +

Cette fonctionnalité ANSI C n'est pas supportée en Sun C.

Prototypage de fonction - Ellipses

ANSI C permet l'usage d'ellipses "..." pour déclarer un nombre variable d'arguments.

Sun C ne reconnait pas cet usage.

Définition de Types

ANSI C permet la redéclaration par `typedef` à l'intérieur d'un bloc. Sun C refuse de telles redéclarations :

```
typedef int * iptr;
{
   typedef double iptr;  /* Bon pour l'ANSI C,
         mais erreur de compilation en Sun C */
}
```

Initialisation des Variables `extern`

ANSI C traite l'initialisation des variables explicitement déclarées `extern` comme des définitions.
Sun C ne supporte pas l'initialisation de variables déclarées explicitement `extern`. L'exemple suivant fonctionne en ANSI C. Il ne se compile pas en Sun C :

```
extern int x = 5;
main()
{
    printf("value of x is %d\n", x);
}
```

Initialisation d'Agrégats

Sun C ne supporte pas l'initialisation ANSI C des unions, tableaux et structures de la classe d'allocation `auto` :

```
main()
{
   union U {
       double d;
   } un = {1234.56789};
   struct st{
       int i;
       char ch;
   }  s = { 5, 'b'};
   int arr[20] = {1, 2};  /* arr[0] = 1,
                             arr[1] = 2, tous
       les autres membres initialisés à 0 */
   ...
}
```

Syntaxe des Directives du Préprocesseur,

ANSI C permet des blancs devant le # d'une directive.
En Sun C, le # d'une directive doit être en colonne 1.

Directive `#error`

Cette directive ANSI C n'est pas reconnue par le Sun C.

Noms de Macros Prédéfinis

Les macros suivantes sont définies en ANSI C mais pas en Sun C :

`__STDC__`	Dans une implémentation conforme ANSI cette macro est définie et non-nulle.
`__TIME__`	La valeur de cette macro est l'heure de compilation sous la forme "hh:mm:ss".
`__DATE__`	Macro donnant la date de compilation sous la forme "Mmm dd yyyy", (par exemple, Feb 11 1991)

Prototypage de fonction

Sun C ne supporte pas le prototypage ANSI des fonctions :

```
int main(void)  /* prototypée */
{
   double area_cir(double r);/* prototypée */
   printf("Surface d'un cercle de 2.5 de rayon ="
                  " %lf\n", area_cir(2.5));
   return 0;
}

double area_cir(double r)/* prototypée */
{
   return( 3.14159 * r * r);
}
```

Programmes des Travaux Pratiques

Travaux Pratiques 1

Programme : `sizes.c`

```
/* afficher la taille en octets des types de base */
int main(void)
{
        printf("taille d'un int \t= %d\n", sizeof(int));
        printf("taille d'un short \t= %d\n", sizeof(short));
        printf("taille d'un long \t= %d\n", sizeof(long));
        printf("taille d'un char \t= %d\n", sizeof(char));
        printf("taille d'un float \t= %d\n", sizeof(float));
        printf("taille d'un double \t= %d\n", sizeof(double));

        return 0;
} /* fin de main */
```

Programme : `printit.c`

```
int main(void)
{
        int val;
        char ch;

        val = 42;
        ch = 'z';
        printf("Le caractère comme un char : %c, comme un int :"
            " %d\n", ch, ch);
        printf("L'entier comme un char : %c, comme un int : %d\n",
             val, val);
        return 0;
} /* fin de main */
```

Travaux Pratiques 2

Programme : `diviz.c`

```
int main(void)
{
        int value, not_divisible;

        not_divisible = 1;
        printf("Entrer un nombre 10<=n<=100:  ");
        scanf("%d", &value);
        if ((value < 10)||(value > 100)) printf("Hors limites.\n");
        else
        {
            if (!(value % 2)) {
                printf("%d divisible par 2.\n", value);
                not_divisible = 0;
            }
            if (!(value % 3)) {
                printf("%d divisible par 3.\n", value);
                not_divisible = 0;
            }
            if (!(value % 5)) {
                printf("%d divisible par 5.\n", value);
                not_divisible = 0;
            }
            if (!(value % 7)) {
                printf("%d divisible par 7.\n", value);
                not_divisible = 0;
            }
            if (not_divisible)
                printf("%d n'est pas divisible par 2, 3, 5, ni 7.\n",
                     value);
        } /* fin de if */
        return 0;
} /* fin de main */
```

Programme : ages.c

```
#define NOW 1991
int main(void) {
        int date,temp,choice,ones,tens,hundreds,thousands,result ;
        printf("Entrez votre année de naissance :  ");
        scanf("%d", &date);
        if ((date < 1900) || (date > 1975))
            printf("Hum, vérifiez.\n");
        else {
            temp = date;
            ones = temp % 10;
            temp = temp / 10;
            tens = temp % 10;
            temp = temp / 10;
            hundreds = temp % 10;
            temp = temp / 10;
            thousands = temp % 10;
            printf("\nFaites un choix :\n\n");
            printf("\t1) Somme des chiffres.\n");
            printf("\t2) Produit des chiffres.\n");
            printf("\t3) Age courant.\n\n");
            printf("Entrez 1, 2, ou 3:  ");
            scanf("%d", &choice);
            switch (choice) {
            case 1:
                result = (thousands + hundreds + tens + ones);
                printf("La somme des chiffres est %d.\n", result);
                break;
            case 2:
                result = (thousands * hundreds * tens * ones);
                printf("Le produit des chiffres est %d.\n", result);
                break;
            case 3:
                result = NOW - date;
                printf("Votre âge est env. %d.\n", result);
                break;
            default:
                printf("Hors limites, fin.\n");
                break;
            } /* fin de switch */
        } /* fin de if else */
        return 0;
} /* fin de main */
```

Travaux Pratiques 3

Programme : review_io.c

```
#include <stdio.h>
#define BASE 16
int main(void)
{
        int num1 = BASE * 2;    /* décimal */
        int num2 = 0xFF;        /* hexa */
        int num3 = 0777;        /* octal */
        int result, x=0, ch;
        float fnum = 42.0 + (float)num1;
        float f = 0.0;
        char ch1 = 'a';
        char ch2 = ch1 + 1;
        char ch;

        printf("Num1 : %d, num2 : 0x%x, num3 : 0%o.\n",
                 num1, num2, num3);
        printf("Ch1: %c, ch2: %c.\n", ch1, ch2);
        printf("Fnum: %f.\n", fnum);

        printf("Entrez un caractère :  ");
        ch = getchar( );
        getchar();      /* enlève le newline */
        printf("Le caractère est %c.\n", ch);

        printf("Entrez un autre caractère :  ");
        scanf("%c", &chr);
        getchar();      /* enlève le newline */
        printf("Le 2ème caractère est %c.\n", chr);

        printf("Entrez un entier puis un réel :  ");
        result = scanf("%d%f", &x, &f);
        printf("Result = %d, x = %d, f = %f.\n", result, x, f);
        return 0;
} /* fin de main */
```

Programme : mixed_io.c

```
#include <stdio.h>
int main(void)
{
        int num;
        float val;
        char ch;

        printf("Entrez un caractère alphabétique :  ");
        scanf("%c", &ch);
        getchar();      /* enlève le newline */
        if ((ch > 64)&&(ch < 123))
            printf("Le caractère est %c.\n", ch);
        else
            printf("Mauvaise saisie.\n");
        printf("Entrez un entier :  ");
        scanf("%d", &num);
        printf("Décimal : %d, hexa : 0X%x, octal : 0%o.\n",
                 num, num, num);
        printf("Entrez un entier :  ");
        scanf("%f", &val);
        printf("Le produit de %d et %f est %.5f.\n",
             num, val, (num * val));
        printf("Notation Scientifique : %E.\n", (num * val));
        printf("Partout où vous irez, je serai.\n");
        return 0;
} /* fin de main */
```

Programme : tva.c

```
#include <stdio.h>
#define RATE .186
int main(void)
{
    double dollar, amt_pd, tax, s_tax(double, double);
    int tmp, round;

    printf("Entrez le montant de l'achat : ");
    scanf("%lf", &dollar);
    tax = s_tax(dollar, RATE);
    amt_pd = dollar + tax;

    /* Gestion des erreurs d'arrondi dans les conversions
    décimal-binaire */

    tmp = amt_pd * 1000.0;
    if ((tmp % 10) > 4)
        round = 1;
    else
        round = 0;
    tmp = (tmp / 10) + round;
    amt_pd = tmp / 100.0;
    printf("Montant HT :\t%8.2f \nTVA :\t\t%9.3f TTC : \t %8.2f \n"
         ,dollar, tax, amt_pd);
    return 0;
} /* fin de main */

double s_tax(double value, double rate)
{
    return(value * rate);
} /* fin de la fonction s_tax */
```

Travaux Pratiques 5

Programme : `triangle.c`

```
#include <stdio.h>
int main(void)
{
        int base, height, width, ch;

        printf("Entrez un caractère : ");
        ch = getchar( );
        printf("Entrez n tel que 1<n<=80 : ");
        scanf("%d", &base);
        if ((base > 80)||(base <= 1)) {
            printf("Base Hors limite, forcée à 40.\n");
            base = 40;
        } /* fin de if */
        for (height = 1; height <= base; height++) {
            for (width = 1; width <= height; width++)
                printf("%c",(char)ch);
            printf("\n");
        } /* fin de for */
        return 0;
} /* fin de main */
```

Programme : `loops.c`

```
#include <stdio.h>
#define BEGIN 97    /* code ASCII de 'a' */
#define END 122     /* code ASCII de 'z' */
#define MIDDLE 110  /* code ASCII de 'n' */
#define NEWLINE "\n"    /* newline */
int main(void)
{
        int index, step, limit;
        char ch, ret;

        printf(NEWLINE);
        do {
            printf("Entrez une minuscule : ");

            /* lecture du caractère et du newline... */
            scanf("%c%c", &ch, &ret);
        } while ((ch > END) || (ch < BEGIN));

        if (ch >= MIDDLE) { /* fin de l'alphabet */
            step = -1;
            limit = BEGIN - 1;
        }
        else {      /* début de l'alphabet */
            step = 1;
            limit = END + 1;
        } /* fin de if else */

        for (index = ch; index != limit; index += step)
            printf("%c  ", index);
        printf(NEWLINE);
        return 0;
} /* fin de main */
```

Travaux Pratiques 6

Programme : `reverse.c`

```
#include <stdio.h>
#define MAX 15
int main(void)
{
        int index, temp, mid, val;
        int arr[MAX];

        for (index = 0, val = 10; index < MAX; index++, val += 10)
            arr[index] = val;

        mid = MAX / 2;
        for (index = 0; index < MAX; index++)
            printf("%d  ",arr[index]);
        printf("\n");

        for (index = 0; index < mid; index++) {
            temp = arr[index];
            arr[index] = arr[(MAX - index) - 1];
            arr[(MAX - index) - 1] = temp;
        } /* fin de for */

        for (index = 0; index < MAX; index++)
            printf("%d  ",arr[index]);
        printf("\n");
        return 0;
} /* fin de main */
```

Programme : dimension2.c

```
#include <stdio.h>
#define ROWS 10
#define COLS 2
int main(void)
{
        int row, col, iarray[ROWS][COLS];

        for (row = 0; row < ROWS; row++) {
            for (col = 0; col < COLS; col++) {
                printf("Entrer tableau[%d][%d]: ", row, col);
                scanf("%d", &iarray[row][col]);
            }
        }
        for (row = 0; row < ROWS; row++) {
            for (col = 0; col < COLS; col++) {
                printf("%d ", iarray[row][col]);
            }
            printf("\n");
        }
        return 0;
} /* fin de main */
```

Travaux Pratiques 8

Programme : `average.c`

```
#include <stdio.h>
#define SIZE 10
#define COLUMNS 10
int main(void)
{
        int ar[SIZE] = {5, 15, 25, 35, 45, 55, 65, 75, 85, 95};
        int *end;
        double  avg, average(int *, int *);
        void print_array(int *, int *);

        end = &ar[SIZE - 1];
        print_array(ar, end);/* Affiche les initialisations */
        avg = average(ar, end);/* calcul de la moyenne */
        printf("La moyenne est %f.\n", avg);
        print_array(ar, end);/* Affiche le tableau fois la moyenne */
        return 0;
} /* fin de main */

double average(int *a, int *end)
{
        int *p, sum;
        double avg;
        for (p = a; p <= end; p++)
            sum += *p;
        avg = (double) sum / (end - a +1);
        for (p = a; p <= end; p++)
            *p = (int) (*p * avg);/* modification du tableau */
        return (avg);
} /* fin de average() */

void print_array(int *a, int *end)
{
        int *p;
        for (p = a; p <= end; p++)
            printf("%d \n ",*p);
        printf("\n");
} /* fin de print_array() */
```

Programme : preverse.c

```
#include <stdio.h>
#define MAX 15

int main(void)
{
        int temp, index;
        static int arr[MAX] ={10,20,30,40,50,60,70,80,
                        90,100,110,120,130,140,150};
        int *ptr, *end, *mid;
        void print_array(int *, int *);

        end = &arr[MAX - 1];
        mid = end -(MAX / 2);
        print_array(arr, end);
        for (ptr=arr,index=0; ptr<=mid; ptr++,index++) {
            temp = *ptr;
            *ptr = *(end - index);
            *(end - index) = temp;
        } /* fin de for */

        print_array(arr, end);
        return 0;
} /* fin de main */

void print_array(int *a, int *limit)
{
        int *p;

        for (p = a; p <= limit; p++)
            printf("%d ",*p);
        printf("\n");

} /* fin de print_array() */
```

Programme : `funcptr.c`

```
#include <stdio.h>      /* déclaration de printf() */

int main(void)
{
        int (*fptr)(const char *, ...);
        fptr = printf;
        (*fptr)("Ici la chaîne que vous voulez.\n");
        return 0;
}
```

Travaux Pratiques 9

Programme : `lupes.c`

```
#include <stdio.h>
#include <ctype.h>  /* header pour les fonctions caractères */
#define BEGIN 'a'   /* code ASCII de 'a' */
#define END 'z'     /* code ASCII de 'z' */
#define MIDDLE 'n'  /* code ASCII de 'n' */
#define NEWLINE "\n"/* newline */

int main(void)
{
        int index, step, limit;
        char ch, ret;

        printf("\n");
        do {
            printf("Entrer une lettre :  ");
            scanf("%c%c", &ch, &ret);
            /* conversion majuscule -> minuscule... */
            if (isalpha(ch))
                ch = tolower(ch);
        } while (!(isalpha(ch))) ;

        if (ch >= MIDDLE) {
            step = -1;
            limit = BEGIN - 1;
        }
        else {
            step = 1;
            limit = END + 1;
        } /* fin de if */

        for (index = ch; index != limit; index += step)
            printf("%c ", index);
        printf(NEWLINE);
        return 0;
} /* fin de main */
```

Programme : stringy.c

```
#include <string.h>
#define MAX 80
int main(void)
{
        char first[MAX], second[MAX], both[(MAX * 2)+ 6];
        int index, slen(char *);

        printf("Entrez une chaîne l<=%d:  ", MAX);
        gets(first);
        printf("Entrez une autre chaîne l<=%d:  ", MAX);
        gets(second);
        printf("Longueur de la 1ère : %d.\n", slen(first));
        printf("Longueur de la 2ème : %d.\n", slen(second));
        strcpy(both,  first);
        strcat(both, " *** ");
        strcat(both,  second);
        printf("Les deux réunies :\n");
        printf("%s\n",both);
        return 0;
} /* fin de main */

/* définition de fonction... */
int slen(char *str)
{
        char *ptr;

        /* incrément du pointeur jusqu'à la fin de la chaîne... */
        for (ptr = str; *ptr != '\0'; ptr++);
        /* renvoie la différence, càd la longueur de la chaîne... */
        return(ptr - str);

} /* fin de slen() */
```

Travaux Pratiques 10

Programme : aged.c

```
#include <stdio.h>
#define NOW 1991
int main(void) {
    int date, temp, choice, ones, tens, hundreds, thousands;
    struct record {
        char name[20];
        int birth_year;
        short age;
        short sum;
        short product;
    } rec;
    printf("Entrez votre prénom et votre année de naissance : ");
    scanf("%s %d", rec.name , &date);
    if ((date < 1900) || (date > 1975))
        printf("Hum, vérifiez bien l'année %d.\n",date);
    else {
        ones = (temp = rec.birth_year = date) % 10;
        tens = (temp /= 10) % 10;
        hundreds = (temp /= 10) % 10;
        rec.sum=(thousands=(temp /= 10) % 10)+ hundreds + tens +ones;
        rec.age = NOW - date;
        rec.product = thousands * hundreds * tens * ones;
        printf("1) Somme des chiffres 2) Produit 3) Age \n");
        printf("Entrez 1, 2, ou 3 : ");
        scanf("%d", &choice);
        switch (choice) {
        case 1:
            printf("La somme des chiffres est %d.\n", rec.sum);
            break;
        case 2:
            printf("Le produit est %d.\n",rec.product);
            break;
        case 3:
            printf("%s, vous avez %d ans.\n",rec.name, rec.age);
            break;
        default:
            printf("Choix hors-limites, sortie.\n");
        } /* fin de switch */
    } /* fin de if */
    return 0;
} /* fin de main */
```

Programme : strux.c

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#define SMAX 4
#define MAX 20
struct record {
        char first[MAX];
        char last[MAX];
        short age;
};
int main(void)
{
        int index, choice;
        char temp[SMAX];
        struct record rex[SMAX];
        void print_strux(struct record);

        for (index = 0; index < SMAX; index++) {
            printf("Entrez le prénom du N˚%d:  ", index+1);
            gets(rex[index].first);
            printf("Entrez le nom du N˚%d:  ", index+1);
            gets(rex[index].last);
            printf("Entrez l'âge de %s :  ", rex[index].first);
            gets(temp);
            rex[index].age = atoi(temp);
        } /* fin de for */

        do {
            printf("Entrez le N˚ à afficher (1-4):  ");
            scanf("%d", &choice);
            if ((choice > 4)||(choice < 1))
                break;
            print_strux(rex[(choice-1)]);
        } while (1);
        return 0;
} /* fin de main */

/* Définition de la fonction print_strux() ... */
void print_strux(struct record rec)
{
        printf("%s %s\n", rec.first, rec.last);
        printf("%d\n",rec.age);
} /* fin de print_strux() */
```

Travaux Pratiques 11

Programme : mod11_review.c

```
#include <stdio.h>
int main(void) {
    struct bit_fields {
        char pad[3];
        unsigned int f1:1;
        unsigned int f2:1;
        unsigned int f3:1;
        unsigned int f4:1;
        unsigned int f5:1;
        unsigned int f6:1;
        unsigned int f7:1;
        unsigned int f8:1;
    };
    union status_flags {
        unsigned int word;
        struct bit_fields bflags;
    } flags;

    printf("Taille de bit_fields: %d\n",sizeof(struct bit_fields));
    printf("Taille de status_flags: %d\n",sizeof(flags));
    flags.bflags.f8 = 1;
    printf("F8 = %d\n", flags.bflags.f8);
    printf("Word = 0x%x\n", flags.word);
    flags.bflags.f7 = 1;
    printf("F7 = %d\n", flags.bflags.f7);
    printf("Word = 0x%x\n", flags.word);
    flags.bflags.f6 = 1;
    printf("F6 = %d\n", flags.bflags.f6);
    printf("Word = 0x%x\n", flags.word);
    flags.bflags.f5 = 1;
    printf("F5 = %d\n", flags.bflags.f5);
    printf("Word = 0x%x\n", flags.word);
    flags.bflags.f4 = 1;
    printf("F4 = %d\n", flags.bflags.f4);
    printf("Word = 0x%x\n", flags.word);
    flags.bflags.f3 = 1;
    printf("F3 = %d\n", flags.bflags.f3);
    printf("Word = 0x%x\n", flags.word);
    flags.bflags.f2 = 1;
    printf("F2 = %d\n", flags.bflags.f2);
    printf("Word = 0x%x\n", flags.word);
```

```
    flags.bflags.f1 = 1;
    printf("F1 = %d\n", flags.bflags.f1);
    printf("Word = 0x%x\n", flags.word);
    return 0;
} /* fin de main */
```

Programme : masks.c

```
#include <stdio.h>
#include <ctype.h>
#define LB 0xFF
#define HB 0xFF00

int main(void) {
    unsigned short num16, get_name(void);
    void error(void);

    num16 = get_name( );
    if (num16 == 0) error( );
    else if (num16 & LB)
        printf("Lettre dans la 1ère moitié de l'alphabet\n");
    else if (num16 & HB)
        printf("Lettre dans la 2ème moitié de l'alphabet\n");
    return 0;
} /* fin de main */

void error(void) {
    fprintf(stderr, "Erreur, et fin de programme.\n");
    exit(1);
} /* fin de error */

unsigned short get_name(void) {
    char name[20], *ch = &name[0];

    printf("Entrez votre prénom :  ");
    gets(name);
    if (isalpha(*ch)) {
        if (islower(*ch))
            *ch = toupper(*ch);   /* Conv. en Majuscule */
        if (*ch <= 'M')
            return(1);  /* retourne le bit le moins fort */
        else
            return(1<<8);/* retourne le bit le plus fort */
    } else
        return(0);
} /* fin de get_name() */
```

Travaux Pratiques 12

Programme : argmanip.c

```
#include <string.h>
int main(int argc, char **argv)
{
        void usage(char *), print_arg(char *, char, int);

        if ((argc > 3) || (argc < 2))
            usage(*argv);
        if (argc == 2)
            print_arg(*(argv+1), 'f', strlen(*(argv+1)));
        else {
            if ((strcmp(*(argv+1),"-f"))&&(strcmp(*(argv+1),"-r")))
                usage(*argv);
            if (!(strcmp(*(argv+1), "-f")))
                print_arg(*(argv+2), 'f', strlen(*(argv+2)));
            else
                print_arg(*(argv+2), 'r', strlen(*(argv+2)));
        } /* fin de if */
        return 0;
} /* fin de main */

void print_arg(char *str, char order, int len)
{
        int m, bumper=1, start=0, end=len;

        if (order == 'r') {
            end = bumper = -1;
            start = len-1;
        }
        for (m=start; m != end; m += bumper)
            putchar(str[m]);
        putchar('\n');
} /* fin de print_arg() */

void usage(char *prog)
{
        printf("\nUsage:\n");
        printf("\t%s [-f | -r] <argument>\n\n", prog);
        exit(1);
} /* fin de usage() */
```

Travaux Pratiques 13

Programme : newaged.c

```
#include <stddef.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#define NOW 1991
#define PATH "data.rec"/* nom codé en dur */

/* typedef global : visibilité depuis toutes les fonctions... */
typedef struct {
    char name[20];
    int birth_year;
    short age;
    short sum;
    short product;
} Record;

int main(void) {
    void read_file(int);
    int write_file(void);

    read_file(write_file( ));
    return 0;
} /* fin de main */

/******************** Définition des fonctions ******************/
int write_file(void) {
    void f_error(int, FILE *);
    char str[5];
    FILE *fp;
    Record rec;
    int date, temp, ones, tens, hundreds, thousands, cnt = 0;

    if ((fp = fopen(PATH, "w")) == NULL)
        f_error(1, fp);
    do {
        cnt++;
        printf("\nEntrez le prénom du %d :  ", cnt);
        gets(rec.name);
        printf("Entrez l'année de naissance de %s :  ", rec.name);
        temp = rec.birth_year = date = atoi(fgets(str, 6, stdin));
        ones = temp % 10;
        tens = (temp /= 10) % 10;
```

```
        hundreds = (temp /= 10) % 10;
        thousands = (temp /= 10) % 10;
        rec.age = NOW - date;
        rec.sum = thousands + hundreds + tens + ones;
        rec.product = thousands * hundreds * tens * ones;
        if (!(fwrite(&rec, sizeof(rec), 1, fp)))
            f_error(2, fp);
        printf("\nEncore une saisie [O] : ");
        fgets(str,5, stdin);
        if ((str[0] != '\0') && (str[0] != 'o') && (str[0] != 'O')){
            fclose(fp);
            printf("\n");
            break;
        }
    } while (1);
    return(cnt);
} /* fin de write_file() */

void read_file(int cnt) {
    void f_error(int, FILE *);
    FILE *fp;
    Record rec;
    int choice, rnum = 0;
    char str[5];

    if ((fp = fopen(PATH, "r")) == NULL)  f_error(1, fp);
    do {
        printf("Entrez le N°(1-%d) à voir, <CR> pour fin :  ", cnt);
        rnum = atoi(fgets(str, 5, stdin));
        if ((rnum < 1) || (rnum > cnt)) {
            fclose(fp);
            break;
        }
        if (fseek(fp, (long)(sizeof(rec) * (rnum - 1)), SEEK_SET))
            f_error(4, fp);
        if (!(fread(&rec, sizeof(rec), 1, fp)))
            f_error(3, fp);
        printf("\nPour %s, Entrez un choix :\n\n", rec.name);
        printf("\t1) Somme des chiffres.\n");
        printf("\t2) Produit des chiffres.\n");
        printf("\t3) Age actuel.\n\n");
        printf("Entrez 1, 2, ou 3 : ");
        choice = atoi(fgets(str, 5, stdin));
        switch (choice) {
        case 1:
                printf("La somme est %d.\n\n", rec.sum);
```

```
            break;
        case 2:
            printf("Le produit est %d.\n\n", rec.product);
            break;
        case 3:
            printf("%s a env. %d ans.\n\n", rec.name, rec.age);
            break;
        default:
            printf("Choix hors-limites.\n\n");
            continue;
        } /* fin de switch */
    } while (1);
} /* fin de read_file() */

void f_error(int etype, FILE *fp) {
    switch(etype) {
    case 1:
        fprintf(stderr, "Erreur d'ouv. de \"%s\", fin.\n", PATH);
        break;
    case 2:
        fprintf(stderr, "Erreur d'écriture : \"%s\", fin.\n", PATH);
        fclose(fp);
        break;
    case 3:
        fprintf(stderr, "Erreur de lecture : \"%s\", fin.\n", PATH);
        fclose(fp);
        break;
    case 4:
        fprintf(stderr, "Erreur en seek : \"%s\", fin.\n", PATH);
        fclose(fp);
        break;
    default:
        fprintf(stderr, "Erreur inconnue, fin.\n");
        break;
    } /* fin de switch */
    exit(1);
} /* fin de f_error() */
```

Programme : alphile.c

```
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define AMAX 16     /* indice de tableau maxi + 1 */
#define SMAX 81     /* longueur maxi de chaîne */

void f_error(char *, int, FILE *);

int main(int argc, char *argv[]) {
    void usage(char *);
    void ssort(int, char [][], char *);
    void shuffle(int, char [][], int);
    void print_array(char [][], int);
    void write_array(char *, int, char [][]);
    FILE *fp;
    static int cnt = 0;
    int index, rnum;
    char ans[4];
    static char sarray[AMAX][SMAX], temp[SMAX];

    if (argc != 2)
        usage(argv[0]);/* il faut un nom de fichier ! */
    printf("\n");
    while (cnt < AMAX) {/* boucle sur le tableau avec saisie...*/
        printf("Enter string #%d (<=80 chars):  ", cnt+1);
        fgets(temp, SMAX, stdin);
        if ((temp[0] == '\0') && (cnt >= 2))
            break;
        else if ((temp[0] == '\0') && (cnt < 2)) {
            printf("Au moins 2 chaînes !\n");
            continue;
        }
        if (cnt == 0) {/* 1ère chaîne saisie */
            strcpy(sarray[cnt], temp);
            cnt++;
            continue;
        }
        ssort(cnt++, sarray, temp);/* tri du tableau */
    } /* fin de while */
    printf("\n");
    print_array(sarray, cnt);   /* affichage du tableau */
    write_array(argv[1], cnt, sarray);  /* écriture dans fichier */
    printf("Suppression d'enregistrements ? [O]:  ");
```

```
    fgets(ans, 4, stdin);
    while ((ans[0] == '\0')||(ans[0] == 'O')||(ans[0] == 'o')) {
        printf("Entrez le numéro à détruire (1-%d):  ", cnt);
        rnum = atoi(fgets(ans, 4, stdin));
        if ((rnum < 1)||(rnum > cnt))
            break;
        shuffle(cnt--, sarray, rnum - 1);/* nouvel ordre */
        printf("Autre enregistrement ? [O]:  ");
        fgets(ans, 4, stdin);
        print_array(sarray, cnt);
    } /* fin de while */
    write_array(argv[1],cnt,sarray);/* écriture nouveau tableau */
    return 0;
} /* fin de main */

void print_array(char arr[AMAX][SMAX], int cnt) {
    int index;

    printf("\n");
    for (index = 0; index < cnt; index++)
        printf("Chaîne N° %d : %s\n", index + 1, arr[index]);
    printf("\n");
} /* fin de print_array() */

void write_array(char *file, int cnt, char arr[AMAX][SMAX]) {
    FILE *fp;
    int index;

    if ((fp = fopen(file, "w")) == NULL)
        f_error(file, 1, fp);
    for (index = 0; index < cnt; index++) {
        if (fprintf(fp, "%s\n", arr[index]) == EOF)
            f_error(file, 2, fp);
    }
    fclose(fp);
} /* fin de write_array() */

void ssort(int cnt, char sa[AMAX][SMAX], char str[]) {
/* cnt    = compteur de chaînes (dernier indice valide) *
 * sa[][] = tableau de chaînes                          *
 * str[]  = chaîne à insérer dans le tableau            */
    int height;
    static char temp[SMAX] ;
    for (height = 0; height <= cnt; height++) {
        if (strcmp(sa[height], str) > 0 || sa[height][0]== 0)) {
            strcpy(temp, sa[height]);
```

```
            strcpy(sa[height], str);
            strcpy(str, temp);
        }
    }
} /* fin de ssort() */

void shuffle(int cnt, char sa[AMAX][SMAX], int pos) {
/* pos =  position à détruire dans le tableau */
    int height;
    char temp[AMAX];
    if (cnt == pos) /* la chaîne à enlever est la dernière... */
        sa[pos][0] = '\0';/* supprime la dernière */
    else {
        for (height = pos + 1; height < cnt; height++)
            strcpy(sa[height-1], sa[height]);
        sa[cnt][0] = '\0';
    }
} /* fin de shuffle() */

void usage(char *prog) {
    fprintf(stderr, "\nUsage:\n");
    fprintf(stderr, "\t%s <chemin>\n\n", prog);
    exit(1);
} /* fin de usage */

void f_error(char *path, int etype, FILE *fp) {
    switch(etype) {
        case 1:
            fprintf(stderr,"Erreur d'ouv. \"%s\", fin.\n", path);
            break;
        case 2:
            fprintf(stderr, "Err. d'écriture : \"%s\",fin.\n",path);
            fclose(fp);
            break;
        default:
            fprintf(stderr, "Err. inconnue, fin.\n");
            break;
    } /* fin de switch */
    exit(1);
} /* fin de f_error() */
```

Travaux Pratiques 15

Programme : `listrux.c`

```
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#define MAX 20
/* Définition des structures de la base... */
struct record {
     char first[MAX];
     char last[MAX];
     short age;
     struct record *next;
};
/* Raccourcis globaux pour les déclarations... */
typedef struct record Rec;
#define SNULL (Rec *)NULL
#define SIZE sizeof(Rec)

int main(int argc, char *argv[]) {
   FILE *fp;
   int fwrite_list(FILE *, Rec *);
   Rec *head, *build_list(void), *fread_list(FILE *);
   void display_list(Rec *), insert_node(Rec **, Rec *);
   void free_list(Rec *), usage(char *);
   void err(char *, char *, int, FILE *);
   if (argc != 2)
      usage(argv[0]);
   printf("\n\n");
   head = build_list( );
   if ((fp = fopen(argv[1], "w")) == NULL)
      err(argv[0], argv[1], 1, fp);
   if (fwrite_list(fp, head))
      err (argv[0], argv[1], 2, fp);
   if ((fp = fopen(argv[1], "r")) == NULL)
      err(argv[0], argv[1], 1, fp);
   head = fread_list(fp);
   display_list(head);
   printf("\n");
   free_list(head);
      return 0;
} /* end main */
```

```
/*********************Définitions des Fontions *********************/

/* Fonction de construction de liste                                 *
 * boucle tant que l'utilisateur entre des chaînes non-nulles. */
Rec *build_list(void) {
   Rec *head, *crnt, *temp;
   void insert_node(Rec **, Rec *);
   char ans[4];

   head = (Rec *)malloc(SIZE);
   crnt = head;/* sauvegarde la tête de liste */
   crnt->next = SNULL;/* fin de liste */
   do {
      printf("Entrez le prénom :  ");
      gets(crnt->first);
      if (crnt->first[0]=='\0'){  /*fin de saisie ?*/
         if (crnt == head) {
            printf("Au moins 1 enregistrement !\n");
            continue;/*boucle pour au moins 1 enregistrement */
         }
         free(crnt);/* libère le dernier malloc */
         break;
      } /* fin de if */
      printf("Entrez le prénom :  ");
      fgets(crnt->last, MAX, stdin);
      printf("Entrez l'âge de %s :  ", crnt->first);
      crnt->age = (short)atoi(fgets(ans, 4, stdin));
      printf("\n");
      insert_node(&head, crnt);   /*insertion dans la liste */
      crnt = (Rec *)malloc(SIZE);
      crnt->next = SNULL;
   } while (1);
   return (head);
} /* fin de build_list */

/* Fonction d'insertion d'un élément de liste, triée *
 * par âges croissants.                               */
void insert_node(Rec **head, Rec *crnt)
{
   Rec *temp, *prev;

   temp = *head;
   if (temp == crnt)  /* premier élément de liste */
      return;
   else if (temp->age >= crnt->age){/* le nouveau est au début*/
      crnt->next = temp;
```

```
      *head = crnt;/* nouvelle tête de liste */
   } else {
      while (crnt->age > temp->age) {/* boucle sur la liste */
         prev = temp;
         temp = temp->next;
         if (temp == SNULL)
            break;          /* fin de liste */
         }
      prev->next = crnt;
      crnt->next = temp;
   } /* fin de if */
   return;
} /* fin de insert_node() */

/* Fonction d'écriture de la liste dans le fichier 'argv[1]'. */
int fwrite_list(FILE *fp, Rec *head) {
    Rec *crnt;
    do {
        crnt = head;
        head = crnt->next;
        crnt->next = SNULL;/* RAZ, l'ancienne adr. est non-valide */
        if (!(fwrite(crnt, SIZE, 1, fp)))
            return(2);
        free(crnt);     /* destruction à chaque pas */
    } while (head);
    fclose(fp);
    return(0);
} /* fin de fwrite_list */

/* Fonction de reconstruction de liste à partir du fichier argv[1] */
Rec *fread_list(FILE *fp) {
   Rec *head, *crnt, *prev;

   head = (Rec *)malloc(SIZE);
   crnt  = head;
   while (fread(crnt, SIZE, 1, fp)) {
      prev = crnt;
      crnt->next = (Rec *)malloc(SIZE);
      crnt = crnt->next;
   }
   prev->next = SNULL;
   free(crnt);/* libère le dernier alloué inutilisé */
   return(head);
} /* end fread_list */
```

```
/* Affiche le contenu de la liste, soit un élément à la fois   *
 * soit toute la liste.                                        */
void display_list(Rec *head) {
   int cnt, choice;
   char ans[40];
   Rec *crnt;

   do {
      printf("\nFaites votre choix parmi :\n");
      printf("\t(1) voir un élément\n");
      printf("\t(2) voir toute la liste\n");
      printf("\t(3) fin du programme\n");
      printf("Entrez 1, 2 ou 3 :  ");
      choice = atoi(fgets(ans, 40, stdin));
      switch (choice) {
         case 1:
            while (choice) {
               crnt = head;/* commence par le début... */
               printf("\nEntrez le prénom à visualiser :  ");
               gets(ans);
               while (crnt && strcmp(crnt->first, ans) ) {
                    crnt = crnt->next;
               } /* fin du while intérieur */
               if (crnt == SNULL)
                  printf("Enreg. de \"%s\" NON-TROUVE !\n", ans);
               else {
                  printf("\nEnreg. de %s :-----------------\n", ans);
                  printf("\tName: %s %s\n", crnt->first, crnt->last);
                  printf("\tAge: %d\n", crnt->age);
                  printf("-------------------------------------\n");
            } /* fin de if else */
            printf("Un autre ? [O]:  ");
            fgets(ans,40,stdin);
            if ((ans[0] != '\0')&&(ans[0] != 'O')&&(ans[0] != 'o'))
               choice = 0;/* fin de boucle */
            } /* fin de while extérieur */
            break;
         case 2:
            cnt = 0;  /* Compteur d'enreg. à 0 */
            crnt = head; /* pointeur sur la tête */
            printf("\n");
            while (crnt) {
               printf("Enreg n° %d :\n", ++cnt);
```

```
            printf("---------------------------------\n");
            printf("\tNom : %s %s\n", crnt->first, crnt->last);
            printf("\tAge: %d\n", crnt->age);
            printf("---------------------------------\n\n");
            crnt = crnt->next;
         } /* fin de while */
         break;
      case 3:
      default:
         exit(0);
   } /* fin de switch */
} while (1); /* fin de do-while */
} /* fin de display_list() */
/* Libération de la mémoire et destruction de liste */
void free_list(Rec *head) {
   Rec *prev;
   while (head) {
      prev = head;
      head = head->next;
      free(prev);
   }
} /* fin de free_list() */
/* Fonction d'affichage d'usage, en cas de mauvais appel du prog. */
void usage(char *prog) {
   printf("\nUsage:\n");
   printf("\t%s <nomdefichier>\n\n", prog);
   exit(0);
} /* fin de usage() */
/* Fonction d'information sur les erreurs d'E/S. */
void err(char *prog, char *file, int type, FILE *fp) {
   switch (type) {
      case 1:
         fprintf(stderr,"%s : erreur d'ouv. de %s, fin\n", prog, file);
         break;
      case 2:
         fprintf(stderr,"%s err. d'écriture sur %s  fin\n",prog,file);
         fclose(fp);
         break;
      default:
         fprintf(stderr, "%s : erreur inconnue, fin.\n", prog);
}
exit(1);
} /* fin de err() */
```

Travaux Pratiques 16

Programme (facultatif) : `factorial.c`

```
#include <stdio.h>
int main(void)
{
        unsigned int num, factorial(int);
        int index;
        char tmp[8];
        void err(void);

        printf("\nEntrez un nombre à factorialiser (n <= 13) : ");
        num = (unsigned int)atoi(fgets(tmp, 8, stdin));
        if (num >= 14)
            err( );

        /* affiche toutes les factorielles de 1 à num... */
        for (index = 1; index <= num; index++)
            printf("%d! est %d.\n", index, factorial(index));
        /* affiche les factorielles de 1 à trop_grand !... */
        printf("\n\n**** Test de limite n = 1 à n = 42...\n");

        for (index = 1; index <= 42; index++)
            printf("%d! est %u.\n", index, factorial(index));
        printf("\n");}
        return 0;
} /* fin de main */

unsigned int factorial(int n)
{
        if (n <= 1)
            return(1);
        else
            return(n * factorial(n - 1));
} /* fin de factorial */

void err(void)
{
        fprintf(stderr,"Saisie hors-limites, fin.\n");
        exit(1);
} /* fin de err */
```

Index

Symboles

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X