Langage de programmation C :

La syntaxe du langage

La syntaxe du langage

- **Instructions, Expressions et Opérateurs**
- **Les structures de contrôle**
- **La récursivité**
- **Les conversions de types**
- **Principales fonctions d'entrées-sorties standard**

Instructions, Expressions et Opérateurs

- Les instructions :
  - **Une instruction** représente une tâche à accomplir par l'ordinateur. En langage C, on écrit une instruction par ligne et elle se termine par un point virgule(à l'exception de #define et #include). Par exemple:

    x=2+3;

    est une instruction d'affectation. Elle demande à l'ordinateur d'ajouter 2 et 3 et d'attribuer le résultat à la variable x.
  - **Les blocs**:
    - Un bloc (ou instructions composées) est un groupe d'instructions entre accolades:

```
{
  printf("Hello");
  printf("world!");
}
```

```
{printf("Hello");
  printf("world!");}
```

Instructions, Expressions et Opérateurs

- Les expressions:

  Une expression est une combinaison d'opérateurs et d'opérandes(variables, constantes). Autrement tout ce qui représente une valeur numérique. Une expression génère toujours un résultat d'un type bien défini qu'on appellera le *type de l'expression*.

  - **Les expressions Simples**
    - L'expression simple est constituée d'une seule variable, d'une constante. Par exemple:
      - PI //constante dont la valeur est définie par #define
      - Taux //variable
  - **Les expressions complexes**
    - Les expressions complexes sont constituées d'expressions plus simples avec des opérateurs. Par exemple:
      - 2+8
      - 8+(5*taux)+(taux*taux)/cout

Instructions, Expressions et Opérateurs

- Les opérateurs: opérateur d'affectation
  - L'opérateur d'affectation est le signe(=). Dans le programme C : x=y;

    ne signifie pas x égal y. Elle indique à l'ordinateur d'affecter la valeur de la variable y à x. Cette instruction doit être composée d'une expression à droite du signe égale, et d'un nom de variable à gauche de ce signe:

    variable=expression;
  - Exemple:
    - x=6+(y=4+5);
    - a=b=2;

Instructions, Expressions et Opérateurs

- Les opérateurs: opérateur d'affectation
  - **Opération et affectation combinées op=**

    var += exp ; équivalent à var = var + (exp) ;

    var -= exp ; var = var - (exp) ;

    var *= exp ; var = var * (exp) ;

    var /= exp ; var = var / (exp) ;

    var %= exp ; var = var % (exp) ;
  - **Attention** :

    pas d'espace entre l'opérateur op et le égal =,

    x *= y+1 est équivalent à x = x*(y+1) et pas x = x*y+1.

Instructions, Expressions et Opérateurs

- Opérateurs arithmétiques

Opérateur	Traduction	Exemple	Résultat
+	Addition	`x + y`	l'addition de x et y
-	Soustraction	`x - y`	la soustraction de x et y
*	Produit	`x * y`	la multiplication de x et y
/	Division	`x / y`	le quotient de x et y
%	Reste	`x % y`	Reste de la division euclidienne de x par y
+(unaire)	Signe positif	`+x`	la valeur de x
-(unaire)	Signe négatif	`-x`	la négation arithmétique de x
++(unaire)	Incrément	`x++` ou `++x`	x est incrémenté ($x = x + 1$). L'opérateur préfixe `++x` (resp. suffixe `x++`) incrémente x avant (resp. après) de l'évaluer
--(unaire)	Decrément	`x--` ou `--x`	x est décrementé ($x = x - 1$). L'opérateur préfixe `--x` (resp. suffixe `x--`) décrémente x avant (resp. après) de l'évaluer

Instructions, Expressions et Opérateurs

- Opérateurs arithmétiques
  - Remarques :
    - Les opérandes de ces opérateurs arithmétiques peuvent appartenir à tout type arithmétique seul l'opérateur % requiert des types entiers.
    - Le résultat d'une division d'entiers est aussi un entier, Exemple :
      - 6 / 4 // Resultat: 1
      - 6 % 4 // Resultat: 2
      - 6.0 / 4.0 // Resultat: 1.5

Instructions, Expressions et Opérateurs

- Opérateurs arithmétiques
  - Remarques
    - Les opérateurs unaires opèrent sur une seule variable ou opérande.
    - Concernant l'incrémentation pré/postfixe, voici un petit exemple: Supposons que la valeur de N soit égale à 3 :
      - Incrémentation postfixe : X = N++ ;

        Résultat : N = 4 et X = 3
      - Incrémentation préfixe : X = ++N ;

        Résultat : N = 4 et X = 4
      - Décrémentation postfixe : X = N-- ;

        Résultat : N = 2 et X = 3
      - Décrémentation préfixe : X = --N ;

        Résultat : N = 2 et X = 2

Instructions, Expressions et Opérateurs

- Opérateurs de comparaison

  - Toute comparaison est une expression de type **int** qui renvoie la valeur **0** (faux) ou **1** (vraie). Il faut que les opérandes soient du même type arithmétique (ou des pointeurs sur des objets de même type).
  - Attention : il ne faut pas confondre l'opérateur d'égalité (==) avec celui d'affectation (=).
  - Les différents opérateurs de comparaison sont détaillés dans le tableau ci-dessous.

Instructions, Expressions et Opérateurs

- Opérateurs de comparaison

Opérateur	Traduction	Exemple	Résultat
<	inférieur	x < y	1 si x est inférieur à y
<=	inférieur ou égal	x <= y	1 si x est inférieur ou égal à y
>	supérieur	x > y	1 si x est supérieur à y
>=	supérieur ou égal	x >= y	1 si x est supérieur ou égal à y
==	égalité	x == y	1 si x est égal à y
!=	inégalité	x != y	1 si x est différent de y

Instructions, Expressions et Opérateurs

- Opérateurs de comparaison

```
#include <stdio.h>
Int main (){
   int x=14,y=1;   // x est différent de y
   if (x = y)         //erreur!!! il faudrait écrire 'if (x == y)'
      printf("x est égal à y (%d=%d)\n",x,y);
  else
      printf("x est différent de y (%d!=%d)\n",x,y);

system(''pause'');
return 0;
}
```

Instructions, Expressions et Opérateurs

- Opérateurs logiques

Les opérateurs logiques, permettent de combiner le résultat de plusieurs expressions de comparaison en une seule expression logique. Les opérandes des opérateurs logiques peuvent être n'importe quel scalaire. Toute valeur différente de 0 est interprétée comme vraie (et 0 correspond à 'faux'). Comme pour les expressions de comparaisons les expressions logiques renvoient une valeur entière (0 =faux ; 1=vraie).

Remarque :

les opérateurs && et || évaluent les opérandes de gauche à droite et le résultat est connu dès l'opérande de gauche. Ainsi, l'opérande de droite n'est évaluée que si celle de gauche est vraie dans le cas de l'opérateur && (respectivement fausse dans le cas de l'opérateur ||).

Exemple:

(i < max) && (f(14) == 1), la fonction f n'est appelée que si i < max.

Instructions, Expressions et Opérateurs

- Opérateurs logiques

Opérateur	Traduction	Exemple	Résultat
&&	ET logique	x && y	1 si x et y sont différents de 0
||	OU logique	x || y	1 si x et/ou y sont différents de 0
!	NON logique	!x	1 si x est égal à 0. Dans tous les autres cas, 0 est renvoyé.

- Exemples
  - L'expression : 32 && 40 vaut 1
  - L'expression : !65.34 vaut 0
  - L'expression : !!0 vaut 0

Instructions, Expressions et Opérateurs

- **Autres opérateurs**
  - **Opérateur séquentiel (,)**
    - **<expr1> , <expr2>,., <exprN>**
    - Exprime des calculs successifs dans une même expression. Le type et la valeur de l'expression sont ceux du dernier opérande.
    - **Exemple : y=(x = 5 , x + 6);** L'expression **y a pour valeur 11**
  - **Opérateur conditionnel (? :)**
    - **<expression> ? <expr1>: <expr2>**
    - <expression> est évaluée. Si sa valeur est non nulle, alors la valeur de <expr1> est retournée. Sinon, c'est la valeur de <expr2> qui est renvoyée.
    - **Exemple : max = a > b ? a : b**
    - si a est le plus grand, alors affectation à max du contenu de a sinon affectation du contenu de b

Instructions, Expressions et Opérateurs

- **Autres opérateurs**

Op.	Traduction	Exemple	Résultat
()	Appel de fonction	`f(x,y)`	Exécute la fonction f avec les arguments x et y
(type)	cast	`(long)x`	la valeur de x avec le type spécifié
`sizeof`	taille en bits	`sizeof(x)`	nombre de bits occupé par x
? :	Evaluation conditionnelle	`x?:y:z`	si x est différent de 0, alors y sinon z
,	séquencement	`x,y`	Evalue x puis y

Exercices

1) Soit les déclarations suivantes :

```
int n=10, p=4;
long q=2;
float x=1.75;
```

Donner le type et la valeur de chacune des expressions suivantes

a) n+q
b) n+x
c) n%p+q
d) n<p
e) q+3 *(n>p)
f) x*(q==2)
g) x*(q=2)
h) (q-2)&&(n-10)

Exercices

1) Soit les déclarations suivantes :

```
int n=10, p=4;
long q=2;
float x=1.75;
```

Donner le type et la valeur de chacune des expressions suivantes

a) n+q
b) n+x
c) n%p+q
d) n<p
e) q+3 *(n>p)
f) x*(q==2)
g) x*(q=2)
h) (q-2)&&(n-10)

a) long 12
b) float 11,75
c) long 4
d) int 0
e) long 5
f) float 1,75
g) float 3.5
h) int 0

Exercices

2. n étant de type int, écrire une expression qui prend la valeur :

   -1 si n est négatif

   0 si n est nul

   1 si n est positif

Exercices

2. n étant de type int, écrire une expression qui prend la valeur :
   - -1 si n est négatif
   - 0 si n est nul
   - 1 si n est positif

```
n ? (n>0 ? 1: -1) : 0
```

Les structures de contrôle

- On appelle structure de contrôle toute instruction servant à contrôler le déroulement de l'enchaînement des instructions à l'intérieur d'un programme, ces instructions peuvent être des instructions conditionnelles ou itératives.
- Parmi les structures de contrôle, on distingue :
  - structures de choix
    - if….else (choix conditionnel)
    - switch (choix multiple)
  - structures répétitives ou itérative ou boucle
    - for
    - while
    - do…while
  - Branchement inconditionnel
    - Break, goto, continue

Les structures de contrôle : if-else

- La construction **if-else** (*si-sinon*) est la construction logique de base du langage C qui permet d'exécuter un bloc d'instructions selon qu'une condition est vraie ou fausse.
- **Syntaxe**
  - **Forme 1**
    - **if** (expression)
      instruction1;

    la forme if est ici dans sa forme la plus simple.

    Si expression est vraie, instruction1 est exécutée.

    Si expression est fausse, instruction1 est ignorée.

Les structures de contrôle : if-else

- **Syntaxe: Forme 2**
  - ```
    if (expression)
      instruction1;
    else
      instruction2;
    ```

Si expression est vraie, instruction1 est exécutée, sinon c'est instruction2 qui est exécutée.

- **Forme 3**
  - ```
    if (expression1)
          instruction1;
    else if (expression2)
          instruction2;
    else
          instruction3;
    ```

Les instructions if sont imbriquées. Si expression1 est vraie, instruction1 est exécutée. Dans le cas contraire expression2 est évaluée si cette dernière est vraie instruction2 est exécutée. Si les deux expressions sont fausses, c'est instruction3 qui est exécuté

Les structures de contrôle : if-else

- Exemples:
  - Exemple 1
    - if (salaire >45.000)
      tax=0.30;
      else
      tax=0.25;
  - Exemple 2
    - if (age <18)
      printf(''mineur'');
      else if (age <65)
      printf(''adulte'');
      else
      printf(''personne agée'');

Les structures de contrôle : if-else

```
#include <stdio.h>
main (){
  int i;
     printf("Tapez un nombre entier positif ou negatif: ");
     scanf("%d", &i);
  if (i<0) {
         i=-i;
         printf("J'ai remis i à une valeur positive.\n");
    }
 else
         printf("Vous avez tapé un nombre positif.\n");

system(« pause»);
}
```

Les structures de contrôle : Switch

- L'instruction switch est l'instruction de contrôle la plus souple du langage C. Elle permet à votre programme d'exécuter différentes instructions en fonction d'une expression qui pourra avoir plus de deux valeurs.

- On l'appelle aussi l'instruction d'aiguillage. Elle teste si une expression prend une valeur parmi une suite de constantes, et effectue le branchement correspondant si c'est le cas.

Les structures de contrôle : Switch

- Syntaxe

```
switch (<variable>) {
    case <valeur 1> : <action 1>; break;
    case <valeur 2> : <action 2>; break;
       …
    default : <action n>;
 }
```

Les structures de contrôle : Switch

- Remarques **:**
  - Le fonctionnement de cette instruction est le suivant :
    - expression est évaluée ;
    - s'il existe un énoncé case avec une constante qui est égale à la valeur de l'expression, le contrôle est transféré à l'instruction qui suit cet énoncé;
    - si un tel case n'existe pas, et si énoncé default existe, alors le contrôle est transféré à l'instruction qui suit l'énoncé default ;
    - si la valeur de l'expression ne correspond à aucun énoncé case et s'il n'y a pas d'énoncé default, alors aucune instruction n'est exécutée.
  - Attention.
    - Lorsqu'il y a branchement réussi à un case, toutes les instructions qui le suivent sont exécutées, jusqu'à la fin du bloc ou jusqu'à une instruction de rupture (break).

Les structures de contrôle : Switch

```
#include<stdio.h>
   main() {
 int a,b,y;
  char operateur;
    printf(‘’ Entrez un opérateur (+, -,* ou /):’’);
    scanf(‘’%c’’,& operateur);
    printf(‘’ Entrez deux entiers:’’);
    pcanf(‘’%d’’,&a); Scanf(‘’ %d’’,&b);
  switch(operateur){
            case '-' : y=a-b; printf(‘’%d’’,y);break;
            case '+' : y=a+b; printf(‘’%d’’,y); break;
            case '*' : y=a*b; printf(‘’%d’’,y); break;
            case '/' : y=a/b; printf(‘’%d’’,y); break;
            default : printf(‘’opérateur inconnu\n’’);break;
}
}
```

Les structures de contrôle : while

- L'instruction while permet de répéter des instructions, tant qu'une condition est vérifiée.
- Syntaxe
  - **Forme 1**

```
while (condition)
        une-instruction;
```

  - **Forme 2**

```
while (condition) {
des instructions;
}
```

Les structures de contrôle :while

- La boucle while fonctionne de la façon suivante:
  1. La condition est évaluée,
  2. si cette condition est fausse, l'instruction while se termine,
  3. si la condition est vraie les instructions sont exécutées
  4. l'exécution reprend à l'étape 1.
- Exemple

```
#include <stdio.h>
 main (){
    int i=1, N;
     printf(" donnez N=\n");
     scanf("%d",&N);
    while (i <= N) {
              printf("%d ", i);
     i = i+1;}
  system("pause");}
```

Les structures de contrôle :while

- while imbriqué

```
#define MAX 4
int i=0,j;
while (i < MAX ) {
        j=0;
        while (j < MAX ) {
            printf(" position :  %d \t  %d\n",i,j);
            j++;
        }
    i++;
}
```

Les structures de contrôle : do-while

- L'instruction **do-while**, exécute le bloc d'instructions tant qu'une condition reste vraie. Dans cette boucle le test de la condition s'effectue à la fin de la boucle.
- Syntaxe
  - **Forme 1**

```
do
        une-instruction;
while (condition);
```

  - **Forme 2**

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

Les structures de contrôle : do-while

- Exemple :

```
#include <stdio.h>
#define MAX 4
 main() {
     int i=MAX;
    do {
         printf("Valeur de i : %i\n",i);
             i++;
         } while (i < MAX);
}
```

Les structures de contrôle : for

- A l'instar des instructions while et do-while l'instruction **for** permet d'exécuter un certain nombre de fois un bloc d'une ou plusieurs instructions.
- Syntaxe:
  - for([expression1] ; [expression2] ; [expression3]) {
    liste d'instructions
    }
  - Les crochets [et] signifient que leur contenu est facultatif
  - Dans la construction de for :
    - **expression1 :** effectue les initialisations nécessaires avant l'entrée dans la boucle;
    - **expression2 :** est le test de continuation de la boucle ; le test est évalué avant l'exécution du corps de la boucle;
    - **expression3 :** est évaluée à la fin du corps de la boucle.

Les structures de contrôle : for

- **Remarques**
  - En pratique, expression1 et expression3 contiennent souvent plusieurs initialisations séparées par des virgules.
    - for(i=0,j=1,k=5;....;...)
  - Les expressions expression1 et expression3 peuvent être absente(les points-virgules doivent apparaître.
    - ```
      i=1;
      for(;i<=5;) {
      printf(''%d fois\n'',i);
      i++;}
      ```
  - Lorsque expression2 est absente, l'expression correspondante est considérée comme vraie. Par conséquent, for(;;) est une boucle infinie
      ```

Les structures de contrôle : for

- Remarques :par définition l'instruction de for

```
for ( expression1 ; expression2 ; expression3 )
{
    liste d'instructions;
}
```

est équivalente à

```
expression1;
while(expression2)
{
    liste d'instructions;
    expression3;
}
```

Les structures de contrôle :for

- **Exemples**
  - Programme pour calculer la somme de 1 à 100

```
int n,total;
for(total=0,n=1;n<=100;n++)
        total+=n;

 printf("la sommes des nombres de 1 à 100 est %d
  \n",total);
```

  - Programme pour calculer la factorielle d'un entier n:

```
int n, i,fact;
for(fact=1,i=1;i<=n;i++)
        fact*=i;
printf("%d!= %d \n",n,fact);
```

Les structures de contrôle : break, goto et continue

- Instruction break
  - On a vu le rôle de l'instruction break; au sein d'une instruction de branchement multiple switch.
  - L'instruction break peut, plus généralement, être employée à l'intérieur de n'importe quelle boucle (for ; while ; do-while). Elle permet l'abandon de la structure et le passage à la première instruction qui suit la structure.
  - En cas de boucles imbriquées, break fait sortir de la boucle la plus interne.

Les structures de contrôle : break, goto et continue

- Instruction break : exemple

```
#include <stdio.h>
 main( ){
int i;
   for (i = 1 ; i<=10 ; i++){
        printf(''début tour %d\n'' ,i) ;
        printf('' bonjour\n'');
        if (i ==3)  break ;
        printf(" fin tour %d\n", i) ;
         }
        printf(" après la boucle \n") ;
    }
```

Les structures de contrôle : break, goto et continue

- **Instruction break : exemple**

```
#include <stdio.h>
 main( ){
int i;
   for (i = 1 ; i<=10 ; i++){
        printf(''début tour %d\n'' ,i) ;
        printf('' bonjour\n'');
        if (i ==3)  break ;
        printf(" fin tour %d\n", i) ;
         }
        printf(" après la boucle \n") ;
    }
```

Evaluation:

Début tour 1
bonjour
fin tour 1
Début tour 2
bonjour
fin tour 2
Début tour 3
bonjour
Après la boucle

Les structures de contrôle : break, goto et continue

- Instruction **continue**

  - L'instruction continue peut être employée à l'intérieur d'une structure de type boucle (for ; while ; do-while).

  - Elle produit l'abandon de l'itération courante et fait passer directement à l'itération suivante d'une boucle.

Les structures de contrôle : break, goto et continue

- Instruction **continue**
- Exemple :

```
main(){
 int i;
   for(i=1;i<=10;i++){
     printf(''début tour %d\n'' ,i) ;
        if (i <4)  continue ;
        printf('' bonjour\n'');
     }
}
```

Evaluation:
Début tour 1
Début tour 2
Début tour 3
Début tour 4
Bonjour
Début tour 5
Bonjour

Les structures de contrôle : break, continue et goto

- Instruction **goto**
  - Elle permet le branchement en un emplacement quelconque du programme.

```
main(){
  int i;
    for(i=1;i<=10;i++) {
     printf(''début tour %d\n'' ,i) ;
          printf('' bonjour\n'');
          if (i ==3)  goto sortie ;
          printf('' fin tour%d\n", i);
     }
sortie:printf (" après la boucle\n") ;
}
```

Evaluation:
Début tour 1
Bonjour
fin tour 1
Début tour 2
bonjour
fin tour 2
Début tour 3
Bonjour
Après la boucle

La récursivité

- Le langage C autorise des appels de fonctions. Celle-ci peut prendre deux aspects:
  - *Récursivité directe*: une fonction comporte, dans sa définition, au moins un appel à elle-même.
  - *Récursivité croisée*: l'appel d'une fonction entraîne celui d'une autre fonction qui, à son tour, appelle la fonction initiale(le cycle pouvant d'ailleurs faire intervenir plus de deux fonctions)

La récursivité

- Exemple

```
#include <stdio.h>
 int fact (int x) {
   printf ("Computing fact %d\n", x) ;
    if (x == 1)
         return 1 ;
   else
      return(  x * fact (x-1)) ;
}
 main (){
  printf ("5! = %d\n", fact (5)) ;
}
```

Les conversions de types

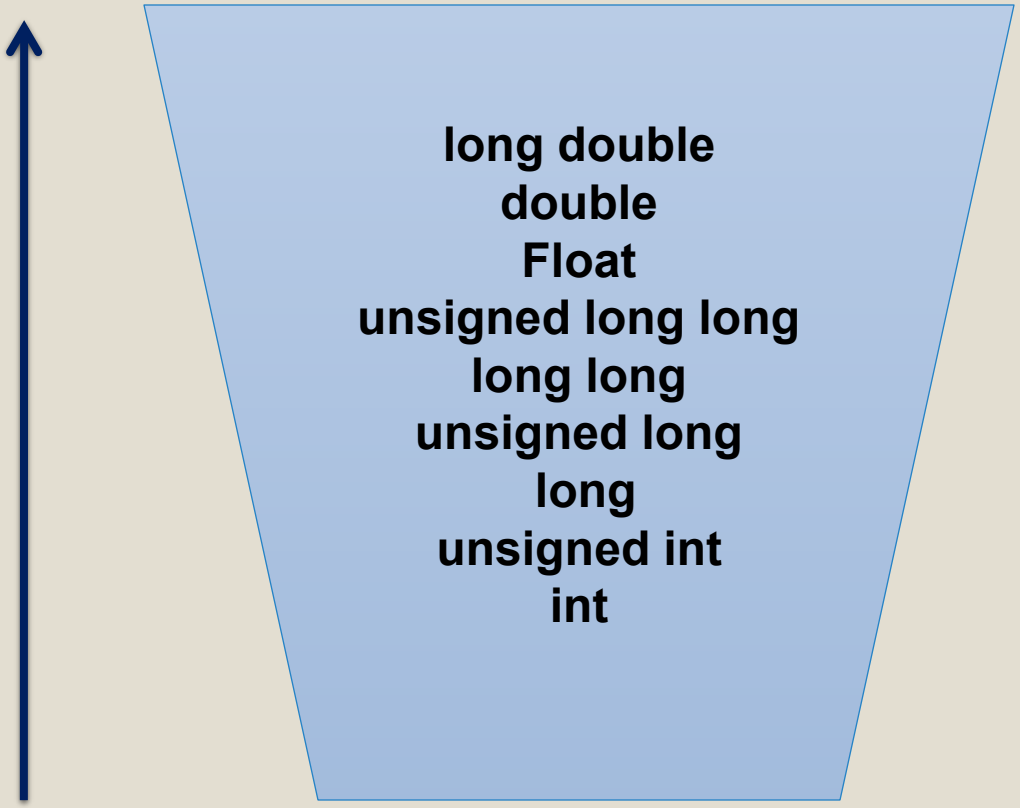
- La conversion de type est un outil très puissant, elle doit donc être utilisée avec prudence. Le compilateur fait de lui-même des conversions lors de l'évaluation des expressions. Pour cela il applique des règles de conversion implicite. Ces règles ont pour but la perte du minimum d'information dans l'évaluation de l'expression. Ces règles sont les suivantes:
  - Règle de Conversion Implicite
    - Convertir les éléments de la partie droite d'une expression d'affectation dans le type de la variable ou de la constante le plus riche.
    - Faire les opérations de calcul dans ce type.
    - Puis convertir le résultat dans le type de la variable affectée.

Les conversions de types

- La notion de richesse d'un type est précisée dans la norme [ISO89]. Le type dans lequel le calcul d'une expression à deux opérandes doit se faire est donné par les règles suivantes :
  1. si l'un des deux opérandes est du type long double alors le calcul doit être fait dans le type long double ;
  2. sinon, si l'un des deux opérandes est du type double alors le calcul doit être fait dans le type double ;
  3. sinon, si l'un des deux opérandes est du type float alors le calcul doit être fait dans le type float ;
  4. si l'un des deux opérandes est du type unsigned long int alors le calcul doit être fait dans ce type ;
  5. si l'un des deux opérandes est du type long int alors le calcul doit être fait dans le type long int ;

Les conversions de types

- Hiérarchie des conversions arithmétiques habituelles est la suivante:



Les conversions de types

- Conversion implicite
  - Il est possible de forcer la conversion d'une variable (ou d'une expression) dans un autre type avant de l'utiliser par une conversion implicite. Cette opération est appelée "cast". Elle se réalise de la manière suivante :

(type) expression

  - Prenons pour exemple l'expression : i = (int) f + (int) d ;
  - f et d sont convertis en int, puis additionnés. Le résultat entier est rangé dans i.

Les conversions de types

- Exemple

float f ; double d ; int i ; long li ;	
li = f + i ;	i est transformé en float puis additionné à f, le résultat est transformé en long et rangé dans li.
d = li + i ;	i est transformé en long puis additionné à li, le résultat est transformé en double et rangé dans d.
i = f + d ;	f est transformé en double, additionné à d, le résultat est transformé en int et rangé dans i.

Principales fonctions d'entrées-sorties standard

- Il s'agit des fonctions de la librairie standard stdio.h utilisées avec les unités classiques d'entrées-sorties, qui sont respectivement le clavier et l'écran.
- Ces fonctions sont:
  - getchar
  - putchar
  - puts
  - printf
  - scanf

Principales fonctions d'entrées-sorties standard

- La fonction getchar
  - La fonction getchar permet la récupération d'un seul caractère à partir du clavier. La syntaxe d'utilisation de getchar est la suivante : **var=getchar();**
  - Notez que var doit être de type char. Exemple :

```
#include <stdio.h>
main() {
    char c;
    printf("Entrer un caractère:");
    c = getchar();
    printf("Le caractère entré est %c\n",c);
}
```

Principales fonctions d'entrées-sorties standard

- La fonction putchar
  - permet l'affichage d'un seul caractère sur l'écran de l'ordinateur. putchar constitue alors la fonction complémentaire de getchar. La syntaxe d'utilisation est la suivante : putchar(var);
  - où var est de type char. Exemple :

```
#include <stdio.h>
main() {
  char c;
  printf("Entrer un caractère:");
  c = getchar();
  putchar(c);
}
```

Principales fonctions d'entrées-sorties standard

- La fonction puts
  - Syntaxe : puts(ch);
  - Cette fonction affiche, sur stdout, la chaîne de caractères ch puis positionne le curseur en début de ligne suivante. puts retourne EOF en cas d'erreur.
  - Exemple :

```
#include <stdio.h>
main() {
   char * toto = "on est super content!";
   puts(toto);
}
```

Principales fonctions d'entrées-sorties standard

- La fonction d'écriture à l'écran formattée printf
  - La fonction printf est une fonction d'impression formatée, ce qui signifie que les données sont converties selon le format particulier choisi. Sa syntaxe est la suivante :

printf("chaîne de contrôle", $expression_1$, . . . , $expression_n$);

  - La chaîne de contrôle contient le texte à afficher et les spécifications de format correspondant à chaque expression de la liste. Les spécifications de format ont pour but d'annoncer le format des données à visualiser. Elles sont introduites par le caractère %. Le i-ème format de la chaîne de contrôle sera remplacé par la valeur effective de $expression_i$.

Principales fonctions d'entrées-sorties standard

- La fonction d'écriture à l'écran formattée printf
  - Les différents formats de la fonction printf

format	Conversion en	écriture
%d	int	décimale signée
%ld	long int	décimale signée
%u	unsigned int	décimale non signée
%lu	unsigned long	décimale non signée
%o	unsigned int	Octale non signée
%lo	unsigned long	Octale non signée
%x	unsigned int	Héxadécimal non signée
%lx	unsigned long	Héxadécimal non signée

Principales fonctions d'entrées-sorties standard

- La fonction d'écriture à l'écran formattée printf
  - Les différents formats de la fonction printf

format	Conversion en	écriture
%f	float	décimale virgule fixe
%lf	Long float	décimale virgule fixe
%e	double	décimale notation exponentielle
%le	long double	décimale notation exponentielle
%g	double	décimale, représentation la plus courte parmi %f et %e
%lg	long double	décimale, représentation la plus courte parmi %lf et %le
%c	unsigned char	caractère
%s	char*	Chaîne de caractères

Principales fonctions d'entrées-sorties standard

- La fonction d'écriture à l'écran formattée printf
  - Les différents formats de la fonction printf
  - Exemple

```
printf("|% d|\n",14);      | 14|
printf("|% d|\n",-14);     |-14|
printf("|%x|\n",0x56ab);    |56ab|
printf("|%X|\n",0x56ab);    |56AB|

printf("|%10d|\n",14);              | 14|
printf("|%f|\n",1.234567890123456789e5);       |123456.789012|
printf("|%.4f|\n",1.234567890123456789e5);      |123456.7890|
```

Principales fonctions d'entrées-sorties standard

- La fonction d'écriture à l'écran formattée printf
  - Les différents formats de la fonction printf
  - Exemple

```
printf("|%e|\n",1.23456789012345789e5); |1.234568e+05|
printf("|%.4e|\n",1.23456789012345789e5); |1.2346e+05||
printf("|%.4g|\n",1.23456789012345789e-5); |1.235e-05|
printf("|%.4g|\n",1.23456789012345789e-3); |0.001235|
printf("|%.8g|\n",1.23456789012345789e5); |123456.79|
```

Principales fonctions d'entrées-sorties standard

- La fonction de saisie scanf
  - La fonction scanf permet de récupérer les données saisies au clavier, dans le format spécifié. Ces données sont stockées aux adresses spécifiées par les arguments de la fonction scanf (on utilise donc l'opérateur d'adressage & pour les variables scalaires). scanf retourne le nombre de valeurs effectivement lues et mémorisées .La syntaxe est la suivante :

    scanf("chaîne de contrôle", & arg1, . . . , & argn);

  - La chaîne de contrôle indique le format dans lequel les données lues sont converties. Comme pour printf, les conversions de format sont spécifiées par un caractère précédé du signe %.

Principales fonctions d'entrées-sorties standard

- La fonction de saisie scanf
  - Exemple

```
#include <stdio.h>
main() {
    int i ;
   printf("entrez un entier sous forme hexadecimale i = ");
   scanf("%x",&i);
   printf("i = %d\n",i);

}
```