ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS.
Mathématiques
Alg-C

Programmation langage C

Introduction à l'algorithmique

A.BELCAID
anasss.belcaid@gmail.com

Salle 1, TD 2 / 23 février 2016

Liste programmes

Hello World !

```c
/*
        Programme qui affiche un simple message
*/

#include <stdio.h>


int main()
{
        printf("Hello World\n");
        return 0;
}
```

Programme 1.1 : Hello World

Somme

```c
/* programme pour lire deux enitiers
 et afficher leur somme*/
#include <stdio.h>
int main()
{
        int a, b;              //entrées du programme
        int somme;
        //lecture
        printf("donner a :");scanf("%d",&a);
        printf("donner b :");scanf("%d",&b);
        //affectation
        somme=a+b;
        //afficher le resultat
        printf("somme= %d\n",somme );
        return 0;
}
```

Programme 1.2 : somme deux entiers

Le langage *C* comporte un ensemble de types basiques qui se répartissent en 3 catégories :

Les Entiers : identifiés par le mot clé **int**.

Types de base

Le langage *C* comporte un ensemble de types basiques qui se répartissent en 3 catégories :

Les Entiers : identifiés par le mot clé **int**.

Les Réels : : mots clé :

Le langage *C* comporte un ensemble de types basiques qui se répartissent en 3 catégories :

Les Entiers : identifiés par le mot clé **int**.

Les Réels : : mots clé :

- **float** :codé en simple précision.

Types de base

Le langage *C* comporte un ensemble de types basiques qui se répartissent en 3 catégories :

Les Entiers : identifiés par le mot clé **int**.

Les Réels : : mots clé :

- **float** :codé en simple précision.
- **double** : codé en double précision.

Le langage *C* comporte un ensemble de types basiques qui se répartissent en 3 catégories :

Les Entiers : identifiés par le mot clé **int**.

Les Réels : : mots clé :

- **float** :codé en simple précision.
- **double** : codé en double précision.

Caractères : mot clé **char**

Le langage *C* comporte un ensemble de types basiques qui se répartissent en 3 catégories :

Les Entiers : identifiés par le mot clé **int**.

Les Réels : : mots clé :

- **float** :codé en simple précision.
- **double** : codé en double précision.

Caractères : mot clé **char**

Types dérivés

ceci nous permet de définir d'autre types (*dérivées*) qui sont soit :

Le langage *C* comporte un ensemble de types basiques qui se répartissent en 3 catégories :

Les Entiers : identifiés par le mot clé **int**.

Les Réels : : mots clé :

- **float** :codé en simple précision.
- **double** : codé en double précision.

Caractères : mot clé **char**

Types dérivés

ceci nous permet de définir d'autre types (*dérivées*) qui sont soit :

- *structurés* : comme les **tableaux**.

Le langage *C* comporte un ensemble de types basiques qui se répartissent en 3 catégories :

Les Entiers : identifiés par le mot clé **int**.

Les Réels : : mots clé :

- **float** :codé en simple précision.
- **double** : codé en double précision.

Caractères : mot clé **char**

Types dérivés

ceci nous permet de définir d'autre types (*dérivées*) qui sont soit :

- *structurés* : comme les **tableaux**.
- *simples* : comme les **pointeurs** et les **énumérations**.

C offre 3 tailles différentes d'entiers, désignées par les mot clés suivants :

Définition	*Domaine*	*Nombres d'octet*
short int	$[-2^{15}, 2^{15}-1]$	2
int	$[-2^{31}, 2^{31}-1]$	4
long	$[-2^{63}, 2^{63}-1]$	8

C offre 3 tailles différentes d'entiers, désignées par les mot clés suivants :

Définition	*Domaine*	*Nombres d'octet*
short int	$[-2^{15}, 2^{15}-1]$	2
int	$[-2^{31}, 2^{31}-1]$	4
long	$[-2^{63}, 2^{63}-1]$	8

Remarque

Le mot *unsigned* (non signé) permet de définir des nombres positifs afin d'améliorer l'intervalle atteint.
exemple :
unsigned int **a** alors **a** $\in [0, 2^{16}-1]$

codes

```c
/*
programme pour illustrer l'intervalle atteint par chaque type */
#include <stdio.h>
#include <math.h>
int main()
{
        int a;                  //entier signe
        short c;                // entier code sur 2 octets
        long d;                 //entier long;
        c=32768;                // c = 2^15
        printf("%d\n",c );      //valeur négative
        a=2147483647;           // a = 2^31 - 1
        printf("a=%d\n",a);
        a=2147483648;           // a = 2^31
        printf("a=%d\n", a);    //valeur négative ???
        d=2147483648;           //pas d'overflow
        printf("d=%ld\n",d);

        return 0;
}
```

Programme 2.1 : Intervalle entiers

Ils permettent de représenter, de manière approchée,une **partie** des nombres réels.

Définition	*min*	*max*	*Nombres d'octet*
float	3.4^{-38},	3.4^{38}	4
double	1.7^{-308}	1.7^{308}	8
long double	3.4^{-4932}	1.1^{4932}	10

Ils permettent de représenter, de manière approchée,une **partie** des nombres réels.

Définition	*min*	*max*	*Nombres d'octet*
float	3.4^{-38},	3.4^{38}	4
double	1.7^{-308}	1.7^{308}	8
long double	3.4^{-4932}	1.1^{4932}	10

Remarque

Quelle que soit la machine utilisée, on assure que l'**erreur de troncature** :

- ne dépassera pas 10^{-6} pour le type **float**.
- ne dépassera pas 10^{-10} pour le type **long double**

Il y a deux notations pour représenter les constantes flottantes :

Notation décimale : elle doit comporter **obligatoirement** un point(correspondant à la virgule).

$23.24 \quad -87.3 \quad -34 \quad -.23 \quad 4.$

Il y a deux notations pour représenter les constantes flottantes :

Notation décimale : elle doit comporter **obligatoirement** un point(correspondant à la virgule).

$23.24 \quad -87.3 \quad -34 \quad -.23 \quad 4.$

Notation exponentielle :

$1.2E2 \quad -32E-3 \quad 2.e14 \quad 33.0e14$

Notation des flottants

Il y a deux notations pour représenter les constantes flottantes :

Notation décimale : elle doit comporter **obligatoirement** un point(correspondant à la virgule).

$23.24 \quad -87.3 \quad -34 \quad -.23 \quad 4.$

Notation exponentielle :

$$1.2E2 \quad -32E-3 \quad 2.e14 \quad 33.0e14$$

On peut imposer à une constante flottante d'être :

- **float** : en faisant suivre son écriture par la lettre F

$$3.5F \quad .53f$$

Il y a deux notations pour représenter les constantes flottantes :

Notation décimale : elle doit comporter **obligatoirement** un point(correspondant à la virgule).

$23.24 \quad -87.3 \quad -34 \quad -.23 \quad 4.$

Notation exponentielle :

$$1.2E2 \quad -32E-3 \quad 2.e14 \quad 33.0e14$$

On peut imposer à une constante flottante d'être :

- **float** : en faisant suivre son écriture par la lettre F

$$3.5F \quad .53f$$

- **long double**, en ajoutant la lettre L.

$$3.5L \quad 4E-10l$$

La déclaration d'une variable de type **caractère** se fait par le mot clé char. on distingue trois catégories de caractères :

Caractères imprimables : Une constante de type caractère est notée en écrivant le caractère entre **apostrophes**.
Exemple : : 'a' 'z' '$' '4'

La déclaration d'une variable de type **caractère** se fait par le mot clé char. on distingue trois catégories de caractères :

Caractères imprimables : Une constante de type caractère est notée en écrivant le caractère entre **apostrophes**.
Exemple : : 'a' 'z' '$' '4'

Caractères non imprimables : possédant une représentation conventionnelle utilisant le caractère \ (*antislash*)
Exemple :

La déclaration d'une variable de type **caractère** se fait par le mot clé char. on distingue trois catégories de caractères :

Caractères imprimables : Une constante de type caractère est notée en écrivant le caractère entre **apostrophes**.
Exemple : : 'a' 'z' '$' '4'

Caractères non imprimables : possédant une représentation conventionnelle utilisant le caractère \ (*antislash*)
Exemple :

- Changement de ligne (\n).

La déclaration d'une variable de type **caractère** se fait par le mot clé char. on distingue trois catégories de caractères :

Caractères imprimables : Une constante de type caractère est notée en écrivant le caractère entre **apostrophes**.
Exemple : : 'a' 'z' '$' '4'

Caractères non imprimables : possédant une représentation conventionnelle utilisant le caractère \ (*antislash*)
Exemple :

- Changement de ligne (\n).
- tabulation (\t)

La déclaration d'une variable de type **caractère** se fait par le mot clé char. on distingue trois catégories de caractères :

Caractères imprimables : Une constante de type caractère est notée en écrivant le caractère entre **apostrophes**.
Exemple : : 'a' 'z' '$' '4'

Caractères non imprimables : possédant une représentation conventionnelle utilisant le caractère \ (*antislash*)
Exemple :

- Changement de ligne (\n).
- tabulation (\t)

Caractères spécifiques : ils ne peuvent pas être représentés classiquement comme (? " \).

liste des Caractères non imprimables et spécifiques

Notation C	*Signification*
\a	Cloche ou bip
\b	Retour arrière(**Backspace**)
\f	Saut de page (**Form Feed**)
\n	Saut de ligne (**Line Feed**)
\r	Retour Chariot(**Carriage Return)**
\t	Tabulation horizontale
\v	Tabulation verticale
\\	\
\'	'
\"	"
\?	**?**

.

- La déclaration des constantes se fait par la **directive** #define

Exemple

```
#define n 10
#define eps 1E-6
```

Dans ce cas *n* et *eps* ne sont pas des variables.

.

- La déclaration des constantes se fait par la **directive** #define

Exemple

```
#define n 10
#define eps 1E-6
```

Dans ce cas *n* et *eps* ne sont pas des variables.

- On peut aussi déclarer des variables constantes par le mot clé **const** :

Exemple

```
const float pi=3.1415;
```

Toute modification de *pi* sera rejetée par le **compilateur**.

voici la liste des opérateurs arithmétiques en **C**.

+	Addition
-	Soustraction
*	Multiplication
/	Division
%	modulo

voici la liste des opérateurs arithmétiques en **C**.

+	Addition
-	Soustraction
*	Multiplication
/	Division
%	modulo

Exemple

$23\%6 = 5$
$5/2 = 2$ (*Division entière*)

voici la liste des opérateurs arithmétiques en **C**.

+	Addition
-	Soustraction
*	Multiplication
/	Division
%	modulo

Exemple

$23\%6 = 5$
$5/2 = 2$ (*Division entière*)
$5./2 = 2.5$

voici la liste des opérateurs arithmétiques en **C**.

+	Addition
-	Soustraction
*	Multiplication
/	Division
%	modulo

Exemple

$23\%6 = 5$
$5/2 = 2$ (*Division entière*)
$5./2 = 2.5$

C, il n'y a pas d'opérateur de puissance. Il est remplacé par la **fonction** pow.

Ce sont des opérateurs de **comparaison**

Opérateur	*Signification*	**Priorité**
$<$	Inférieur à	2
$<=$	Inférieur ou égal à	2
$>$	Supérieur à	2
$>=$	Supérieur ou égal à	2
$==$	Comparaison(Egal à)	1
$!=$	Différent de	2

Ce sont des opérateurs de **comparaison**

Opérateur	*Signification*	**Priorité**
$<$	Inférieur à	2
$<=$	Inférieur ou égal à	2
$>$	Supérieur à	2
$>=$	Supérieur ou égal à	2
$==$	Comparaison(Egal à)	1
$!=$	Différent de	2

Exemple

$\mathbf{a} < \mathbf{b} == \mathbf{c} >= \mathbf{d} \Longleftrightarrow (\mathbf{a} < \mathbf{b}) == (\mathbf{c} < \mathbf{d})$

Ce sont des opérateurs de **comparaison**

Opérateur	*Signification*	**Priorité**
$<$	Inférieur à	2
$<=$	Inférieur ou égal à	2
$>$	Supérieur à	2
$>=$	Supérieur ou égal à	2
$==$	Comparaison(Egal à)	1
$!=$	Différent de	2

Exemple

$\mathbf{a < b == c >= d} \Longleftrightarrow \mathbf{(a < b) == (c < d)}$

Le résultat de l'évaluation d'une expression logique est soit **0** soit **1**(C'est un *entier* non pas un **booléen**).

liste des opérateurs logiques :

Opérateur	*Signification*
&&	**ET**(AND)
||	**OU**(OR)
!	**NON**(NOT)

liste des opérateurs logiques :

Opérateur	*Signification*
&&	**ET**(AND)
||	**OU**(OR)
!	**NON**(NOT)

Exemple

$(5 < 3)\&\&(30 > 20)$	prend la valeur 0.
$(5 < 3)\|\|(30 > 20)$	prend la valeur 1.
$!(5 < 3)$	prend la valeur 1.

Remarque

Les opérateurs logiques sont de **priorité** inférieure à celle des opérateurs arithmétique et relationnels.

Opérateur d'affectation

-

affectation simple

i=2 affecte la valeur 2 à **i**.
j=**i**

-

affectation simple

i=2 affecte la valeur 2 à **i**.
j=**i**

- La **priorité** de l'affectation est **inférieure** à celle de tous les opérateurs arithmétiques et relationnels.

affectation simple

i=2 affecte la valeur 2 à **i**.
j=**i**

- La **priorité** de l'affectation est **inférieure** à celle de tous les opérateurs arithmétiques et relationnels.

Exemple

C=a+b/d ; évalue $a + \frac{b}{d}$ **puis** affecte le résultat à **C**.
C+5=X ; n'a pas de sens.

affectation simple

i=2 affecte la valeur 2 à **i**.
j=**i**

- La **priorité** de l'affectation est **inférieure** à celle de tous les opérateurs arithmétiques et relationnels.

Exemple

C=a+b/d ; évalue $a + \frac{b}{d}$ **puis** affecte le résultat à **C**.
C+5=X ; n'a pas de sens.

- L'affectation possède une **association** de droite à gauche. $a = b = 2$ évaluation d'abord de $b = 2$

- Dans une expression on peut avoir des opérandes avec des types **différents**.Dans ce cas, le **Compilateur C** utilise une *Conversion* selon une **hiérarchie** qui permet de ne pas dénaturer la valeur initiale :

- Dans une expression on peut avoir des opérandes avec des types **différents**.Dans ce cas, le **Compilateur C** utilise une *Conversion* selon une **hiérarchie** qui permet de ne pas dénaturer la valeur initiale :

- Dans une expression on peut avoir des opérandes avec des types **différents**.Dans ce cas, le **Compilateur C** utilise une *Conversion* selon une **hiérarchie** qui permet de ne pas dénaturer la valeur initiale :

hiérarchie

- **short**→**int** → **long** → **float** → **double** → **long double**

- Dans une expression on peut avoir des opérandes avec des types **différents**.Dans ce cas, le **Compilateur C** utilise une *Conversion* selon une **hiérarchie** qui permet de ne pas dénaturer la valeur initiale :

hiérarchie

- **short**→**int** → **long** → **float** → **double** → **long double**

- Dans une expression on peut avoir des opérandes avec des types **différents**.Dans ce cas, le **Compilateur C** utilise une *Conversion* selon une **hiérarchie** qui permet de ne pas dénaturer la valeur initiale :

hiérarchie

- **short**→**int** → **long** → **float** → **double** → **long double**

Exemple

```
int b,c ;   float x ;   long a ;
a = a * b
```

- Dans une expression on peut avoir des opérandes avec des types **différents**.Dans ce cas, le **Compilateur C** utilise une *Conversion* selon une **hiérarchie** qui permet de ne pas dénaturer la valeur initiale :

hiérarchie

- **short**→**int** → **long** → **float** → **double** → **long double**

Exemple

```
int b,c ;   float x ;   long a ;
a = a * b
```

- Dans une expression on peut avoir des opérandes avec des types **différents**.Dans ce cas, le **Compilateur C** utilise une *Conversion* selon une **hiérarchie** qui permet de ne pas dénaturer la valeur initiale :

hiérarchie

- **short**→**int** → **long** → **float** → **double** → **long double**

Exemple

int b,c ; **float** x ; **long** a ;
$\mathbf{a} = \mathbf{a} * \mathbf{b}$ b est implicitement converti en **long**
$\mathbf{x} = \mathbf{a} + \mathbf{c} * \mathbf{x}$

- Dans une expression on peut avoir des opérandes avec des types **différents**.Dans ce cas, le **Compilateur C** utilise une *Conversion* selon une **hiérarchie** qui permet de ne pas dénaturer la valeur initiale :

hiérarchie

- **short**→**int** → **long** → **float** → **double** → **long double**

Exemple

int b,c ; **float** x ; **long** a ;

$\mathbf{a} = \mathbf{a} * \mathbf{b}$ b est implicitement converti en **long**

$\mathbf{x} = \mathbf{a} + \mathbf{c} * \mathbf{x}$ toutes la variables sont **converties** en **float**

-

Opérateurs d'incrémentation et décrémentation

++i	: opérateur pré-incrémentation
i++	: Opérateur de post-incrémenation
− − i	: Opérateur pré-décrémentation
i − −	: Opérateur post-décrémentation

++i	: opérateur pré-incrémentation
i++	: Opérateur de post-incrémenation
−−i	: Opérateur pré-décrémentation
i−−	: Opérateur post-décrémentation

Exemple

```
y=4 ;
x=++y-2 ;
```

++i	: opérateur pré-incrémentation
i++	: Opérateur de post-incrémenation
− − i	: Opérateur pré-décrémentation
i − −	: Opérateur post-décrémentation

Exemple

```
y=4;
x=++y-2;
```

$\Rightarrow \quad x = 3 \quad y = 5$

++i	: opérateur pré-incrémentation
i++	: Opérateur de post-incrémenation
− − i	: Opérateur pré-décrémentation
i − −	: Opérateur post-décrémentation

Exemple

```
y=4;
x=++y-2;
```

$\Rightarrow \quad x = 3 \quad y = 5$

```
z=y++ -2;
```

++i	: opérateur pré-incrémentation
i++	: Opérateur de post-incrémenation
− − i	: Opérateur pré-décrémentation
i − −	: Opérateur post-décrémentation

Exemple

```
y=4;
x=++y-2;
```
$\Rightarrow \quad x = 3 \quad y = 5$
```
z=y++ -2;
```
$\Rightarrow \quad z = 3 \quad y = 6$

Opérateur d'affectation élargie

Forme Générale

Var *opérateur*=Expression $\Longleftrightarrow$ **Var**=**Var** opérateur Expression ;

Exemple

i+=3 ; $\Longleftrightarrow$ i=i+3 ;

- a*=b ; $\Longleftrightarrow$ a=a*b ;

c/=(3*i-1) $\Longleftrightarrow$ c=c/(3*i-1) ;

Opérateur CAST

- Il permet de forcer la conversion des type.

Exemple

int n,p ;
double(n/p) :Évaluation de l'expression (n/p) puis conversion du résultat en **double**.
float c ;
int(c/p) :Evaluation de l'expression (c/p) puis élimination de la partie réelle.

- C'est un opérateur qui permet de **simplifier** les instructions de *sélection*.

- C'est un opérateur qui permet de **simplifier** les instructions de *sélection*.

syntaxe

Var=(expression) ?valeur1 : valeur2

- **Var** aurra **valeur1** si (expression) est *vraie*, sinon elle prendra valeur2

- C'est un opérateur qui permet de **simplifier** les instructions de *sélection*.

syntaxe

Var=(expression) ?valeur1 : valeur2

- **Var** aurra **valeur1** si (expression) est *vraie*, sinon elle prendra valeur2

Exemple

$max(a, b) \iff max = (a < b)?x : y$

- Elle permet **d'afficher** des informations à l'écran, déclarée dans le fichier *stdio.h*.

Syntaxe

- *printf* (**format**, **list-expressions**)

Format : une chaine de caractère qui peut contenir soit
- ▷ des caractères simples.
- ▷ des code de format repérés par % suivis par le **le code de conversion**

List-expressions : suite d'expressions séparées par (,)

- **Exemple :** printf("factorial(%d)=%d",n,fact) ;

Codes de conversion

c	char(aussi pour short et int)
d	int(aussi char ou short)
u	unsigned int
ld	long
lu	unsigned long.
f	float en virgule flottante.
e	double ou float en notation scientifique.
g	meilleure representation entre f et e
s	chaine de caractères.

```c
/* quelque exemples des codes des types pour printf*/
#include <stdio.h>

int main()
{
        int a=3; long fact=6;
        float x=3.1415;  double y=23.138548664; // y = e^x
        char c='E';                             // char
        double eps=1E-15;                        //ϵ = 10^−15

        //afficher un caractère
        printf("c=%c\n",c);

        //afficher un entier et un long
        printf("factorial(%d)=%ld\n",a,fact);

        //afficher x et exponentielle x
        printf("exp(%f)=%lf\n",x,y);

        //afficher une tolérance epsilon
        printf("eps=%lf\n",eps ); //virgule floattante
        printf("eps=%e\n",eps);   //format scientifique
        printf("eps=%g\n",eps );  //meilleure format
        return 0;
}
```

- elle permet de **lire** des informations du clavier. cette fonction est aussi déclarée en *stdio.h*

- elle permet de **lire** des informations du clavier. cette fonction est aussi déclarée en *stdio.h*

syntaxe

scanf(**format**,**list-addresses**);

- elle permet de **lire** des informations du clavier. cette fonction est aussi déclarée en *stdio.h*

syntaxe

scanf(**format**,**list-addresses**);

Exemple

scanf("**%d**",&n); lecture d'une valeur entière.
scanf("**%f%f**",&x,&y); lecture de deux valeurs réels.

- La séquence(*bloc d'instruction*).

- La séquence(*bloc d'instruction*).
- Les structures de contrôle.

- La séquence(*bloc d'instruction*).
- Les structures de contrôle.

  if . . . else : Branchement conditionnel.

- La séquence(*bloc d'instruction*).
- Les structures de contrôle.

  if . . . else : Branchement conditionnel.
  switch : Branchement multiple.

- La séquence(*bloc d'instruction*).
- Les structures de contrôle.

  if . . . else : Branchement conditionnel.
  switch : Branchement multiple.

- les **boucles** :

- La séquence(*bloc d'instruction*).
- Les structures de contrôle.

  if . . . else : Branchement conditionnel.
  switch : Branchement multiple.

- les **boucles** :

  finies : **for**

- La séquence(*bloc d'instruction*).
- Les structures de contrôle.

  if . . . else : Branchement conditionnel.
  switch : Branchement multiple.

- les **boucles** :

  finies : **for**

- La séquence(*bloc d'instruction*).
- Les structures de contrôle.

  if . . . else : Branchement conditionnel.
  switch : Branchement multiple.

- les **boucles** :

  finies : **for**
  infinies : ▷ **while**. . ..

- La séquence(*bloc d'instruction*).
- Les structures de contrôle.

  if . . . else : Branchement conditionnel.
  switch : Branchement multiple.

- les **boucles** :

  finies : **for**
  infinies :
  - **while**. . ..
  - **do** . . . **while**.

Syntaxe 1

if (*condition*)
    bloc d'instruction.

Syntaxe 1

if (*condition*)
    bloc d'instruction.

Syntaxe 2

if (*condition*)
    bloc d'instruction 1 ;
else
    bloc d'instruction 2 ;

Remarque

Les **blocs d'instruction** sont quelconques, et donc peuvent contenir des **structures de contrôle** aussi.

Coordonnées polaires

```c
/* calculer les coordonnées polairs d'un point A(x,y) */
#include <stdio.h>
#include <math.h>

int main()
{
        float x,y;              // coordonnées cartesiennes
        float r,theta;          // coordonnées polaires

        //lecture
        printf("donner x=");scanf("%f",&x);
        printf("donner y=");scanf("%f",&y);

        r=sqrt(x*x+y*y);                // r = sqrt(x^2 + y^2)

        if(x>0)
                theta=atan(y/x);            // θ = arctang(y/x)
        else if(x<0)
                theta=atan(y/x)+M_PI;       //M_PI=\pi
        else
        {
                if(y>0)
                        theta=M_PI/2;
                else if(y<0)
                        theta=-M_PI/2;
                else
                        theta=0;
        }

        printf("(%f,%f)=%f exp(%f)\n",x,y,r,theta );
        return 0;
}
```

Definition

L'instruction *switch* est une instruction de séléction multiple qui généralise la selecion simple.

Definition

L'instruction *switch* est une instruction de séléction multiple qui généralise la selecion simple.

syntaxe

```
switch( expression)
{
  case Constante1 : [Instruction1]
                    break ;
  case Constante2 : [Instruction2]
                    break ;
  ⋮
  default : [Instruction]
}
```

Switch

```c
/* illustration de l'utilisation de l'instruction switch*/
#include <stdio.h>

int main()
{
        int choix;

        printf("Choisissez une methode d'interpolation :\n");
        printf("(1=lagrance), (2=newton), (3=regularisation)\n");
        scanf("%d",&choix)

        //traitementn selon la methode choisie
        switch(choix)
        {
                case 1 :
                        //Interploation par  Lagrange;
                        break;
                case 2  :
                        //Interpolation par Newton
                        break;
                case 3 :
                        //Regularisation
                        break;

                default :
                        //aucun choix sortir
        }
        return 0;
}
```

Programme 5.2 : Utilisatin switch

syntaxe

for(**Initalisations** ; **test d'arrêt** ; **Incrémentation**)
    bloc d'instructions.

[1] Exercices 6,7

syntaxe

for(**Initalisations** ; **test d'arrêt** ; **Incrémentation**)
    bloc d'instructions.

[1] Exercices 6,7

syntaxe

for(**Initalisations** ; **test d'arrêt** ; **Incrémentation**)
    bloc d'instructions.

[1] Exercices 6,7

syntaxe

for(**Initalisations** ; **test d'arrêt** ; **Incrémentation**)
    bloc d'instructions.

Exemple

calcul de

$$S = \sum_{i=1}^{n} i$$

[1] Exercices 6,7

syntaxe

for(**Initalisations** ; **test d'arrêt** ; **Incrémentation**)
    bloc d'instructions.

Exemple

calcul de

$$S = \sum_{i=1}^{n} i$$

[1] Exercices 6,7

syntaxe

for(**Initalisations** ; **test d'arrêt** ; **Incrémentation**)
bloc d'instructions.

Exemple

calcul de

$$S = \sum_{i=1}^{n} i$$

[1]

```
int i,S;
for(S=0,i=1;i<n;i++)
        S+=i;
printf("S=%d\n",S );
```

[1] Exercices 6,7

syntaxe

initalisation ;
while(**condition**)
{
    Traitement ;
    Incrémentation
}

Exemple

Calculer $\min\{n | \sum_{i=1} n\frac{1}{i} > A\}$

```
//initialisation
S=0;i=1;
while(S<A)
{
        S+=1./i;
        i++;
}
printf("n cherche est %d\n",i );
```

syntaxe

initalisation ;
do
{
    Traitement ;
    Incrémentation
}**while**(*condition*) ;

Lecture controllée

Forcer la lecture d'un entier positif.

```c
int a;

do
{
        printf("donner a :");
        scanf("%d",&a);
} while (a<0);
// on met la condition contraire!!!!
// aussi faire attention au;
```

C possède des instructions de branchement comme :

break

sert à **interrompre** le déroulement d'une boucle.

C possède des instructions de branchement comme :

break

sert à **interrompre** le déroulement d'une boucle.

continue

elle permet de passer au tour suivant d'une boucle

Exemples

```c
int i;
for(i=1;i<=10;i++)
{
        if(i%3==0)
                break;
        printf("i=%d\n",i);
}
```

Exemples

```c
int i;
for(i=1;i<=10;i++)
{
        if(i%3==0)
                break;
        printf("i=%d\n",i);
}
```

```c
int i;
for(i=0;i<=10;i++)
{
        if(i%3==0)
                continue;
        printf("i=%d\n",i)
}
```

Exemples

```c
int i;
for(i=1;i<=10;i++)
{
        if(i%3==0)
                break;
        printf("i=%d\n",i);
}
```

Affichage

1 2

```c
int i;
for(i=0;i<=10;i++)
{
        if(i%3==0)
                continue;
        printf("i=%d\n",i)
}
```

Exemples

```c
int i;
for(i=1;i<=10;i++)
{
        if(i%3==0)
                break;
        printf("i=%d\n",i);
}
```

Affichage

1 2

```c
int i;
for(i=0;i<=10;i++)
{
        if(i%3==0)
                continue;
        printf("i=%d\n",i)
}
```

Affichage

1 2 4 5 7 8 10