Introduction à la programmation en langage C

Pr. Mohamed El Ansari
Faculté des Sciences
Agadir

Historique

- Langage de programmation developpé en 1970 par Dennie Ritchie aux Laboratoires Bell d'AT&T.
- Il est l'aboutissement de deux langages :
  - BPCL d´evelopp´e en 1967 par Martin Richards.
  - B d´evelopp´e en 1970 chez AT&T par Ken Thompson.
- Il fut limité à l'usage interne de Bell jusqu'en 1978 date à laquelle Brian Kernighan et Dennie Ritchie publièrent les spécifications définitives du langage : ***The C programming Language***.

Historique

- Au milieu des années 1980 la popularité du langage était établie.
- De nombreux compilateurs ont été écrits, mais comportant quelques incompatibilités portant atteinte à l'objectif de portabilité.
- Il s'est ensuivi un travail de normalisation effectué par le comité de normalisation **X3J11** de l'**ANSI** qui a abouti en 1988 avec la parution par la suite du manuel : ***The C programming Language – 2ème édition.***

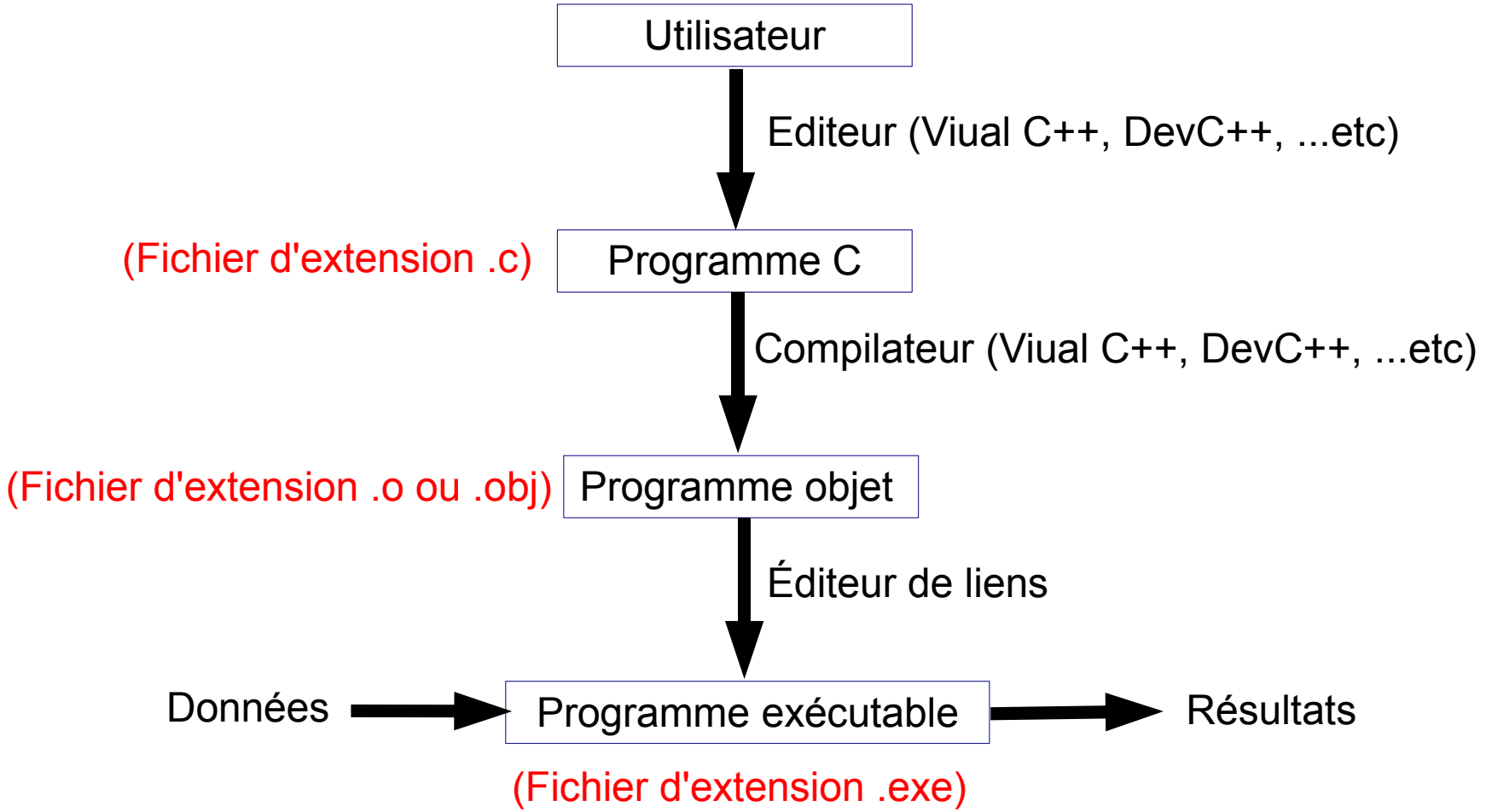
Intérêts du langage

- Langage polyvalent permettant le développement de systèmes d'exploitation, de programmes applicatifs scientifiques et de gestion.
- Langage structuré.
- Langage évolué qui permet néanmoins d'effectuer des opérations de bas niveau (***assembleur d'Unix***).
- Portabilité (en respectant la norme !) due à l'emploi de bibliothèques dans lesquelles sont reléguées les fonctionnalités liées à la machine.
- Grande efficacité et puissance.
- Langage permissif !!!

Qualités attendues d'un programme

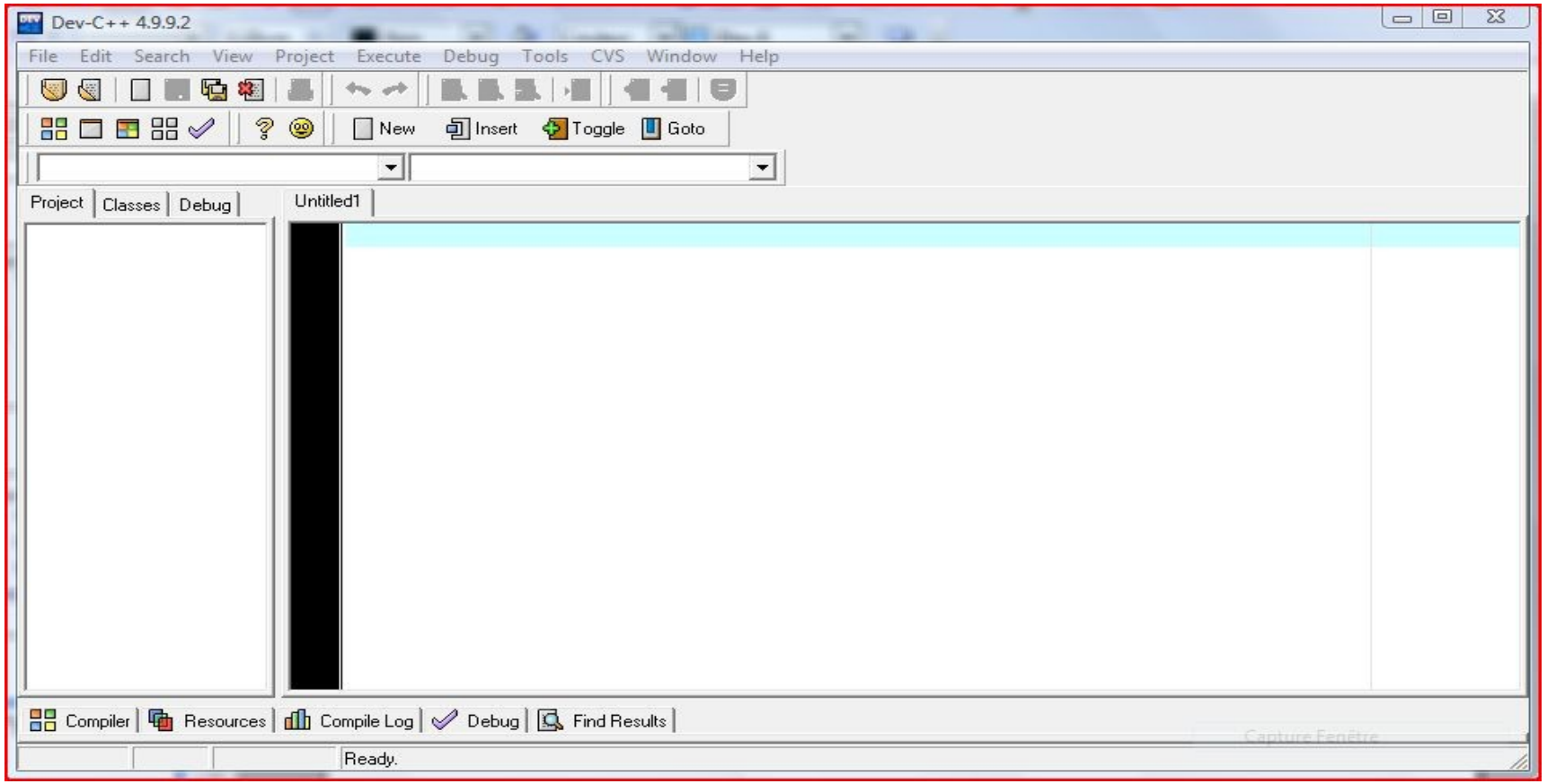
- Clarté
- Simplicité
- Modularité
- Extensibilité

Exécution d'un programme C



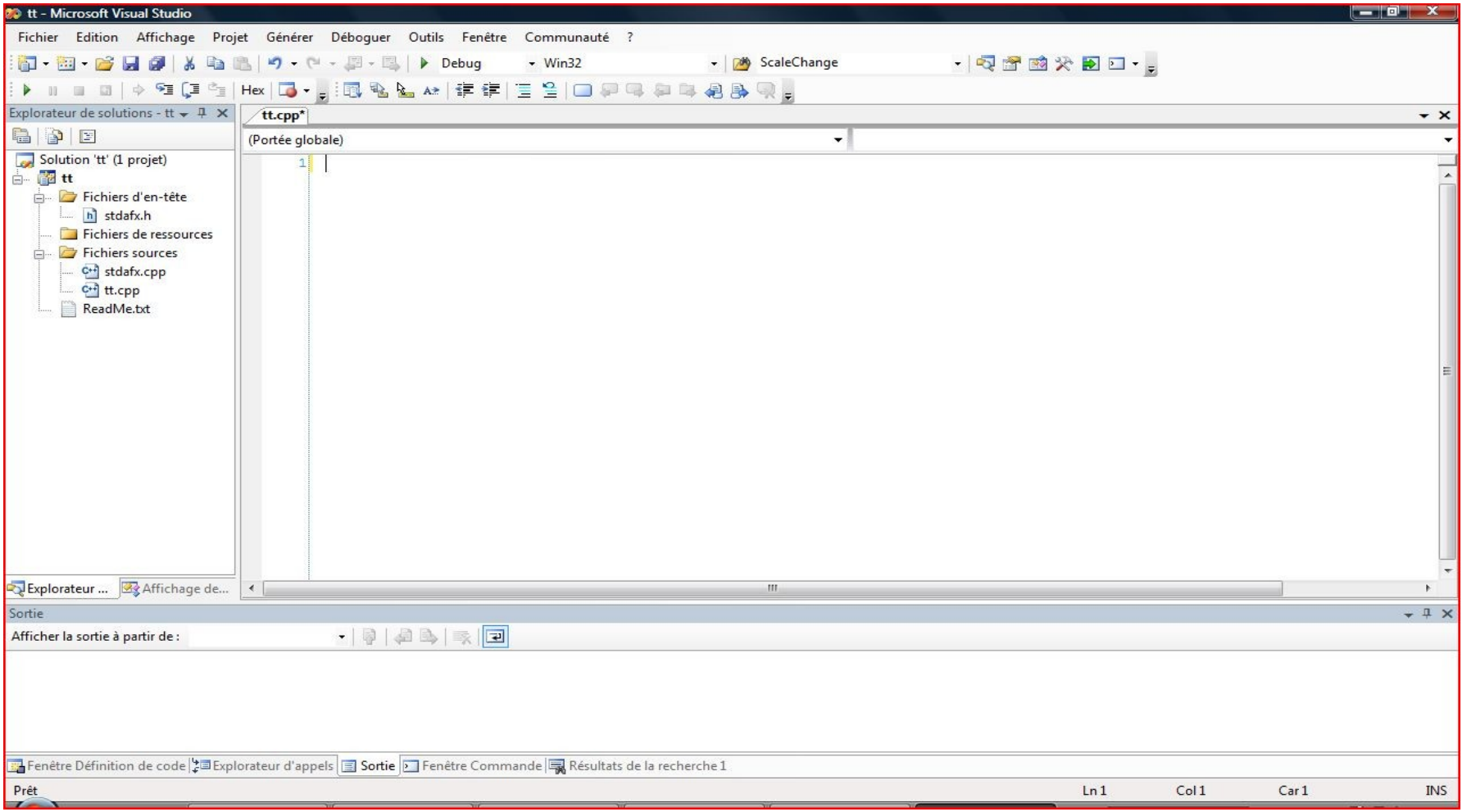
Éditeur de programme et compilateur (Sous Windows)

DevC++ : gratuit (pas de licence)



Éditeur de programme et compilateur (Sous Windows)

Microsoft Visual Studio



Premier programme C

- Programme permettant l'affichage du message : *Hello World!*

```
#include<stdio.h>
main()
{
   printf(''Hello World!\n'');
}
```

inclure la bibliothèque standard des entrées/sortie

fonction nommée `main` sans arguments

Début du programme principal

La fonction main fait appel à la fonction printf de la bibliothèque `stdio` pour afficher la séquence de caractères.
\n : permet d'effecteur un passage à la ligne.

Fin du programme principal

Le plus petit programme

- Le petit programme qu'on peut écrire :

```
main()
{
}
```

Structure d'un programme

- Programme C:
  une ou plusieurs **fonctions** dont l'une doit s'appeler **main** stockées dans un ou plusieurs fichiers.

- Fonction :
  - entête (type et nom de la **fonction** suivis d'une liste d'arguments entre parenthèses),
  - instruction composée constituant le corps de la **fonction**.

```
int somme(int a, int b)
{
   return a+b;
}
```

Structure d'un programme

- Instruction composée : délimitée par les caractères *{* et *}*

- Instruction simple : se termine par **;**

- Commentaire : encadré par les délimiteurs **/*** et ***/**

- Instruction préprocesseur : commence par **#**

Structure générale d'un programme

(Programme de factorisation d'une équation second

```
/* entête : fichiers inclus */
#include <stdio.h>
#include <math.h> /* etc. */
```

(1)

```
/* déclarations de constantes
et de variables globales */
#define FALSE 0
#define TRUE 1
float a, b, c;
```

(2)

```
/* prototypes de fonctions */
void factorisation(float a, float b, float c);
```

(3)

Structure générale d'un programme

```
main () /* programme principal */
{
printf("Factorisation d'un polynome réel de degre 2\n");
printf("Entrez trois nombres reels a, b, c\n");
scanf("%f %f %f", &a, &b, &c);
if (a==0)
    {
    printf("Erreur de saisie\n");
    exit(0);
    }
factorisation(a, b, c);
} /* fin du programme */
```

(4)

Structure générale d'un programme

```
/* définition des fonctions */
void factorisation(float a, float b, float c)
/* on assume que a est non nul */
{
float delta = b*b - 4*a*c;
/* delta est le discriminant */
/* c'est une variable locale */
printf("La factorisation donne \n");
if (delta==0)
printf(" (%f)(x-(%f))(x-(%f))\n",
a,-b/(2*a),-b/(2*a));
else if (delta>0)
printf(" (%f)(x-(%f))(x-(%f))\n",
a,(-b-sqrt(delta))/2*a, (-b+sqrt(delta))/2*a);
else
printf(" (%f)(x+(%f)+(%f)i)(x+(%f)-(%f)i)\n",
a, b/(2*a), sqrt(-delta)/2*a, b/(2*a),
sqrt(-delta)/2*a);
}
```

(5)

Bloc (1)

- Le bloc (1) correspond aux fichiers inclure. Sa première ligne, `#include <stdio.h>`, invoque la bibliothèque des fonctions d'entrée-sortie (saisie au clavier `scanf()` et affichage à l'écran `printf()`).
- La deuxième ligne, `#include<math.h>`, fait appel aux fonctions mathématiques (`sqrt()` : racine carrée).
- En général, cette partie comporte l'inclusion des bibiothèques contenant les fonctions appelées dans le programme principal (fonction `main`).

Blocs (2) et (3)

- Vient ensuite —bloc (2)— la définition des constantes et des variables *globales*, c'est-à-dire vues depuis tous les points du programme :

```
#define FALSE 0
#define TRUE 1
float a, b, c;
```

- Puis on trouve le *prototype* de la fonction *`factorisation()`* :

```
void factorisation (float a, float b, float
 c);
```

- Celle-ci prend trois paramètres —a, b, c : les coefficients du polynôme à factoriser— de type float; mais comme elle ne retourne aucune valeur, elle est typée `void`.

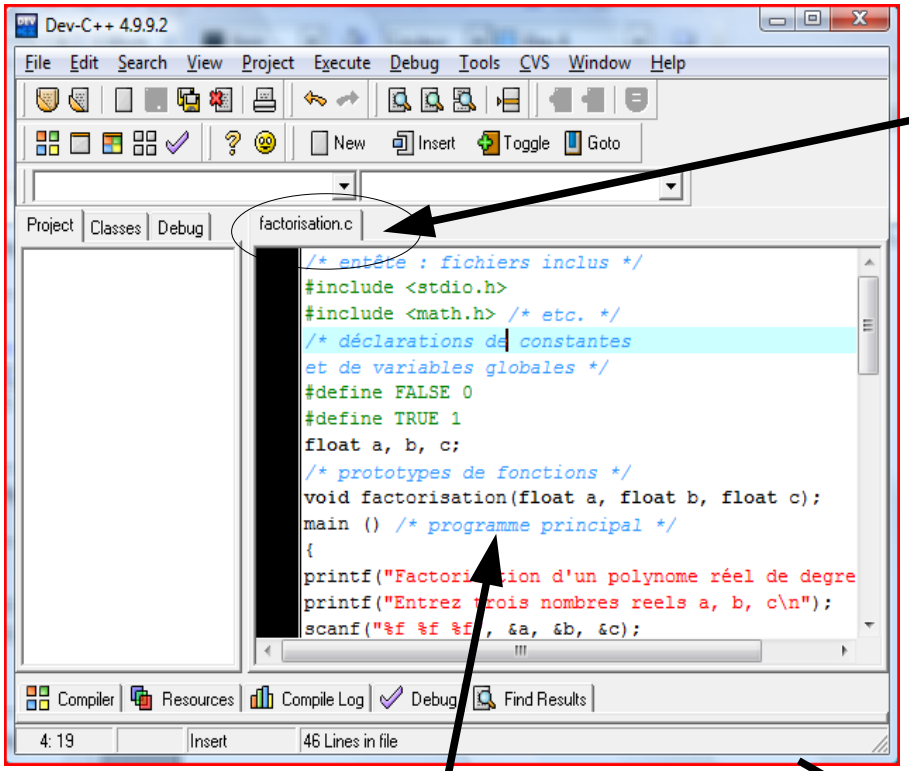
Blocs (4)

- C'est le point d'entrée du programme.
- Tout programme en Langage C a une fonction `main()` et une seule.
- La fonction main affiche deux lignes de message et lit ensuite à partir du clavier —`scanf("%f %f %f", &a, &b, &c);`— jusqu'à l'entrée de trois nombres 'flottants'.
- Après avoir testé la condition `(a==0)` la fonction `factorisation()` est appelée et le programme se termine.

Blocs (5)

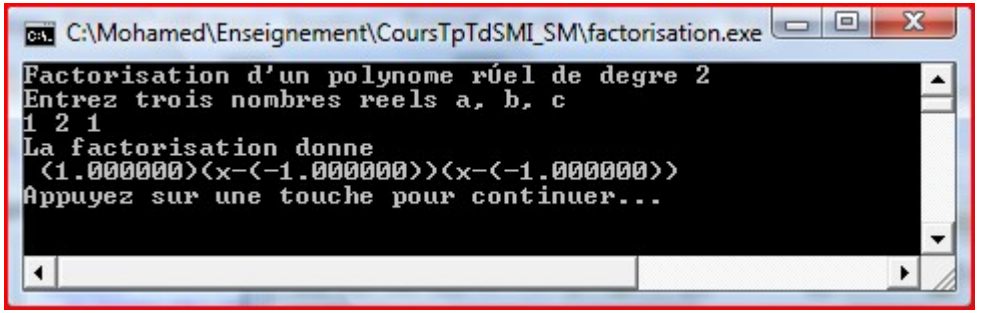
- Le dernier bloc (5) est le code de la fonction factorisation().
- Le lecteur reconnaîtra la définition du discriminant et l'expression des différentes factorisations selon la valeur de delta (le discriminant).
- Nous n'entrerons pas maintenant dans le détail de la syntaxe des fonctions printf() et scanf(), qui sont parmi les plus compliquées du langage C.

Exécution du programme factorisation

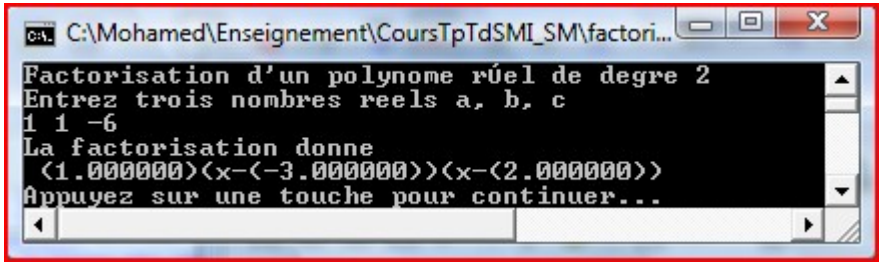


Nom du programme

Première exécution (Résultat)



Première exécution



Programme C (factorisation)

Jeu de caractères utilisés

- 26 lettres de l'alphabet (minuscules, majuscules)
- chiffres 0 à 9
- caractères spéciaux :

  ! * + \ " < # (= | { > %) ;] / - [: , ? & } . (espace) ...etc
- Séquences d'échappement telles :
  - passage à la ligne (\n),
  - tabulation (\t),
  - backspace (\b),
  - Sonnette (\a),
  - ...etc

.

Identificateurs et mots-clés

- Identificateur : nom donné aux diverses composantes d'un programme ; variables, tableaux, fonctions.
- Formé de lettres et de chiffres ainsi que du caractère _ permettant une plus grande lisibilité.
- Le 1er caractère doit obligatoirement être une lettre ou bien le caractère _.
- L'identificateur peut contenir jusqu'à 31 caractères minuscules et majuscules.
- Il est d'usage de réserver les identificateurs
- entièrement en majuscules aux variables du préprocesseur.

Identificateurs et mots-clés

Exemples:

- Identificateurs valides :

  x y12 somme_1 _temperature

  noms surface fin_de_fichier TABLE

- Identificateurs invalides :

4eme	commence par un chiffre
x#y	caractère non autorisé (#)
no-commande	caractère non autorisé (-)
taux change	caractère non autorisé (espace)

Identificateurs et mots-clés

Mots réservés (mots-clés) :

auto	extern	sizeof	break	float	static
case	for	struct	char	goto	switch
const	if	typedef	continue	int	union
default	long	unsigned	do	register	void
double	return	volatile	else	short	while
enum	signed				

Les types de base

- Le langage C contient des types de base qui sont les **entiers**, les **réels** simple et double précision et les caractères que l'on identifie à l'aide des mots-clés **int**, **float**, **double** et **char** respectivement.
- De plus il existe un type **ensemble vide**: le type **void**.
- Les mots-clés **short** et **long** permettent d'influer sur la taille mémoire des entiers et des réels.

Les types de base

La liste des différents types de base:

Syntaxe	Type
`void`	ensemble vide
`char`	caractère
`short int`	entier court
`int`	entier par défaut
`long int`	entier long
`float`	réel simple précision
`double`	réel double précision
`long double`	réel précision étendue

Les types de base

- La taille d'un **entier** par défaut est soit 2 soit 4 octets (dépend de la machine). 4 octets est la taille la plus courante.
- La taille d'un **entier court** est en général 2 octets et celle d'un **entier long** est 4 octets.
- La taille d'un *entier court* est inférieure ou égale à la taille d'un *entier par défaut* qui est elle-même inférieure ou égale à celle d'un *entier long*.
- Les types **short int** et **long int** peuvent être abrégés en **short** et **long**.
- Le type **char** occupe un octet. Un caractère est considéré comme un entier qu'on pourra donc utiliser dans une expression arithmétique.

Les types de base

Les deux mots-clés **unsigned** et **signed** peuvent s'appliquer aux types caractère et entier pour indiquer si le bit de poids fort doit être considéré ou non comme un **bit de signe**.

Les entiers sont **signés** par défaut, tandis que les caractères peuvent l'être ou pas suivant le compilateur utilisé.

Une déclaration telle que **unsigned char** permettra de désigner une quantité comprise entre **0** et **255** , tandis que **signed char** désignera une quantité comprise entre **-128** et **+127**.

De même **unsigned long** permettra de désigner une quantité comprise entre **0** et 232 −1, et **long** une quantité comprise entre -2^{31} et $2^{31} - 1$.

Taille en mémoire

- Le langage C est doté de la fonction **sizeof()** qui permet de fournir la taille des types de base.
- Exemple: taille en octet d'un double.

```
#include<stdio.h>
main()
{
  printf(" %d octet(s)\n",sizeof(double));
       system("pause");
}
```