

4 La programmation élémentaire en C/C++

Peter Schlagheck

Université de Liège

Ces notes ont pour seule vocation d'être utilisées par les étudiants dans le cadre de leur cursus au sein de l'Université de Liège. Aucun autre usage ni diffusion n'est autorisé, sous peine de constituer une violation de la Loi du 30 juin 1994 relative au droit d'auteur.

4 La programmation élémentaire en C/C++

4.1 Les langages de programmation

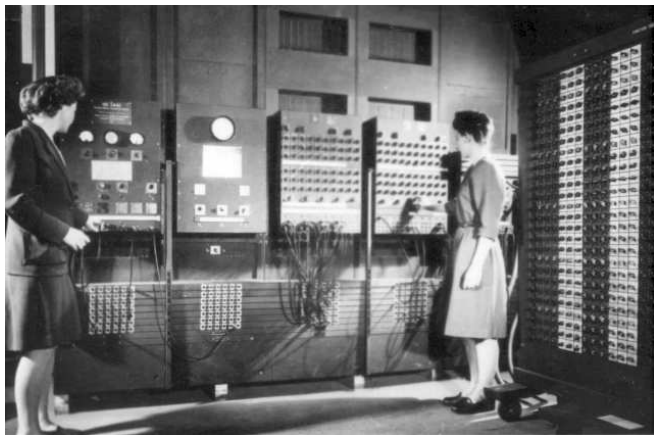
4.2 Un programme simple en C/C++

4.3 Les variables élémentaires

4.4 Les opérations arithmétiques élémentaires

4.1 Les langages de programmation

La programmation avant les années 50 ...



4.1 Les langages de programmation

- ▶ les langages d'assembleur (introduits dans les années 50)

```
16: LOAD *10
17: ADD *15
18: STORE *4
19: IF (ACC NOT 0) GOTO 17
20: ...
```

Principe de base:

- création d'un fichier de texte avec un tel code
- traduction en langage de machine par un programme "assembleur" et placement dans la mémoire

4.1 Les langages de programmation

- ▶ les langages d'assembleur (introduits dans les années 50)

16: 01001010

17: 10001111

18: 01100100

19: 00010001

20: ...

Principe de base:

- création d'un fichier de texte avec un tel code
- traduction en langage de machine par un programme "assembleur" et placement dans la mémoire

4.1 Les langages de programmation

- ▶ les langages d'assembleur (introduits dans les années 50)

16: 01001010

17: 10001111

18: 01100100

19: 00010001

20: ...

Principe de base:

- création d'un fichier de texte avec un tel code
- traduction en langage de machine par un programme "assembleur" et placement dans la mémoire
- exécution

4.1 Les langages de programmation

- ▶ les langages d'assembleur (introduits dans les années 50)

```
16: LOAD *10
17: ADD *15
18: STORE *4
19: IF (ACC NOT 0) GOTO 17
20: ...
```

- représentation directe du langage de machine sous une forme “lisible” (par des humains)
- difficile d'encoder des opérations composées (comme $a = b + \sqrt{c - d/2}$)
- difficile d'écrire des programmes plus compliqués ...

4.1 Les langages de programmation

- ▶ les langages d'assembleur
- ▶ Fortran (Formula translation)

```
program test
integer a, b

a = 84
b = -28
do while (a.ne.0)
    a = a + b
    print *, a
end do
end
```


4.1 Les langages de programmation

- ▶ les langages d'assembleur
- ▶ Fortran
 - introduit en 1956 par IBM
 - fortement utilisé pour des programmes mathématiques et des simulations numériques
(physique/chimie/biologie/climatologie numérique)

4.1 Les langages de programmation

- ▶ les langages d'assembleur
- ▶ Fortran
 - introduit en 1956 par IBM
 - fortement utilisé pour des programmes mathématiques et des simulations numériques
 - plusieurs révisions et évolutions:
 - Fortran 66 (1966)
 - Fortran 77 (1978)
 - Fortran 90
 - Fortran 95
 - Fortran 2003
 - Fortran 2008
 - ...

4.1 Les langages de programmation

- ▶ les langages d'assembleur
- ▶ Fortran
- ▶ BASIC
(Beginner's All-purpose Symbolic Instruction Code)
 - conçu en 1963 (Dartmouth College, USA) comme langage d'apprentissage de la programmation
 - très facile à écrire des programmes
 - grande popularité dans les années 70/80
 - début du succès de la société Microsoft
(→ Microsoft BASIC pour les PCs de IBM)

4.1 Les langages de programmation

- ▶ les langages d'assembleur
- ▶ Fortran
- ▶ BASIC

```
10 N = 84  
20 K = -28  
30 N = N + K  
40 IF N > 0 THEN GOTO 30
```

4.1 Les langages de programmation

- ▶ les langages d'assembleur
- ▶ Fortran
- ▶ BASIC
- ▶ Pascal
 - introduit en 1970 (Niklaus Wirth)
basé sur le langage ALGOL (1958)
 - conçu pour permettre de structurer des algorithmes
(`if`, `while`, `for`, `case`) et des données
(tableaux, pointeurs, ...)
 - grande popularité dans les années 80
("Turbo Pascal" de Borland)

4.1 Les langages de programmation

- ▶ les langages d'assembleur
- ▶ Fortran
- ▶ BASIC
- ▶ Pascal
- ▶ C
 - développé en 1972 (Dennis Ritchie, Bell Labs)
utilisé pour implémenter le système d'exploitation Unix
 - combine les avantages d'un langage structuré
avec l'efficacité d'un langage assembleur
 - utilisé comme langage "intermédiaire"
pour créer des systèmes d'exploitation,
des logiciels standards, des compilateurs ...

4.1 Les langages de programmation

- ▶ les langages d'assembleur
- ▶ Fortran
- ▶ BASIC
- ▶ Pascal
- ▶ C
 - B. Kernighan & D. Ritchie:
"The C programming language"
(Prentice Hall, 2nd edition)

4.1 Les langages de programmation

- ▶ les langages d'assembleur
- ▶ Fortran
- ▶ BASIC
- ▶ Pascal
- ▶ C
- ▶ C++
 - développé en 1979 (Bjarne Stroustrup, Bell Labs) comme une espèce de “révision” de C avec des outils supplémentaires puissants
 - langage de programmation “orientée objet” (influencé par le langage Simula)
 - fortement utilisé par des professionnels

4.1 Les langages de programmation

- ▶ les langages d'assembleur
- ▶ Fortran
- ▶ BASIC
- ▶ Pascal
- ▶ C
- ▶ C++
- ▶ Java
 - publié en 1995 (James Gosling, Sun Microsystems)
 - langage de programmation orientée objet (influencé par C++, avec plus de “sécurité”)
 - fortement utilisé pour des applications sur le web

4.1 Les langages de programmation

- ▶ les langages d'assembleur
- ▶ Fortran
- ▶ BASIC
- ▶ Pascal
- ▶ C
- ▶ C++
- ▶ Java
- ▶ Python
 - première version en 1989
 - langage de programmation multi-style
 - très facile à apprendre

4.1 Les langages de programmation

“Introduction à la programmation en C/C++” veut signifier:

- ▶ on utilise un **compilateur C++**
- ▶ on passe la plupart du temps avec la discussion des **outils élémentaires** de C

→ C++ dans le style C

4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;
}
```

ceci est le contenu d'un fichier nommé `prog1.cpp`
qui se trouve quelque part dans un répertoire de l'utilisateur

4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;
}
```

Compilation et traduction en programme de machine
(sous Linux/Unix/Mac):

```
> c++ prog1.cpp
```

→ création d'un fichier `a.out`

Exécution:

```
> ./a.out
bonjour
```

4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;
}
```

**Compilation et traduction en programme de machine
(sous Linux/Unix/Mac):**

```
> g++ prog1.cpp -o prog1
```

→ création d'un fichier prog1

Exécution:

```
> ./prog1
bonjour
```

4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;
}
```

4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;
}
```

Exécution:

```
bonjour
tapez un nombre reel:
```


4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;
}
```

Exécution:

```
bonjour
tapez un nombre reel: 1.2
```

4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;
}
```

Exécution:

```
bonjour
tapez un nombre reel: 1.2
voila le resultat d'une operation arithmetique: 2.4
```

4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
        << "operation arithmetique: " << n*a << endl;

    double b = 1.5e-3;
    double c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Exécution:

bonjour

tapez un nombre reel:

4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
        << "operation arithmetique: " << n*a << endl;

    double b = 1.5e-3;
    double c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Exécution:

bonjour

tapez un nombre reel: 1.2

4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
        << "operation arithmetique: " << n*a << endl;

    double b = 1.5e-3;
    double c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Exécution:

bonjour

tapez un nombre reel: 1.2

voila le resultat d'une operation arithmetique: 2.4

voila une autre operation:1.203

4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
        << "operation arithmetique: " << n*a << endl;

    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Exécution:

bonjour

tapez un nombre reel:

4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
        << "operation arithmetique: " << n*a << endl;

    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Exécution:

bonjour

tapez un nombre reel: 4

4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;

    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Exécution:

bonjour

tapez un nombre reel: 4

voila le resultat d'une operation arithmetique: 8

voila une autre operation:4.003

4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Exécution:

bonjour

tapez un nombre reel:

4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Exécution:

bonjour

tapez un nombre reel: -0.1

4.2 Un programme simple

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
        << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Exécution:

bonjour

tapez un nombre reel: -0.1

voila le resultat d'une operation arithmetique: -0.2

voila une autre operation:-0.097

4.2 Un programme simple

Pour l'instant on voit la structure suivante de notre programme:

```
#include <iostream>
using namespace std;
```

→ on ne comprend pas encore ce que ça veut dire

4.2 Un programme simple

Pour l'instant on voit la structure suivante de notre programme:

```
#include <iostream>
using namespace std;
```

```
int main()
{
    ...
}
```

→ ça contient toutes les instructions du programme
séparées par des point-virgules ;

4.2 Un programme simple

Pour l'instant on voit la structure suivante de notre programme:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;
    → affichage (output) sur l'écran
}
```

4.2 Un programme simple

Pour l'instant on voit la structure suivante de notre programme:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;
    ...
    cin >> a;
    → lecture (input) par le clavier
}
```

4.2 Un programme simple

Pour l'instant on voit la structure suivante de notre programme:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;
    ...
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;
    → affichage consécutif du texte et d'une valeur
}
```


4.2 Un programme simple

Pour l'instant on voit la structure suivante de notre programme:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    int n = 2;
    → définition d'une variable n
      initialisation avec la valeur 2
}
```

4.2 Un programme simple

Pour l'instant on voit la structure suivante de notre programme:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    int n = 2;
    double a;
    → définition d'une variable a
      pas d'initialisation
}
```

4.2 Un programme simple

Pour l'instant on voit la structure suivante de notre programme:

```
#include <iostream>
using namespace std;
```

```
int main()
{
    cout << "bonjour" << endl;
```

```
    int n = 2;
```

```
    double a;
```

```
    ...
```

```
    double b = 1.5e-3;
```

→ **définition d'une variable b**

initialisation avec la valeur $1.5 \times 10^{-3} = 0.0015$

```
}
```

4.2 Un programme simple

Pour l'instant on voit la structure suivante de notre programme:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    int n = 2;
    double a;
    ...
    double b = 1.5e-3;
    double c = a + n * b;
    → définition d'une variable c
      initialisation avec le résultat d'une opération arithmétique:
        
$$c = a + n \times b$$

}
```

4.2 Un programme simple

Pour l'instant on voit la structure suivante de notre programme:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    int n = 2;
    double a;
    ...
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    → opération arithmétique et copie du résultat dans c
}
```

4.2 Un programme simple

Pour l'instant on voit la structure suivante de notre programme:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
→ commentaire:
    cette ligne est ignorée par le compilateur
}
```

4.2 Un programme simple

Pour l'instant on voit la structure suivante de notre programme:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;
```

```
    /* premiere partie */
```

→ un autre commentaire, ignoré par le compilateur
il commence par /* et finit par */

```
}
```

4.2 Un programme simple

Pour l'instant on voit la structure suivante de notre programme:

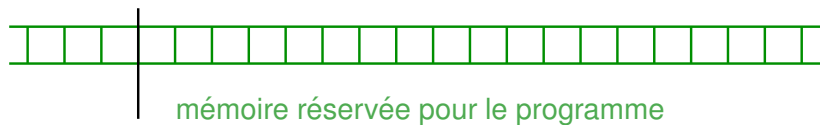
```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

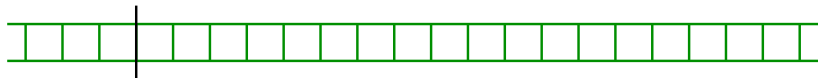
    /* *****\
    |* premiere partie *|
    \***** */
    → il peut s'étendre sur plusieurs lignes:
    tout est ignoré entre /* et */

}
```


Le flot du programme dans la mémoire



Le flot du programme dans la mémoire



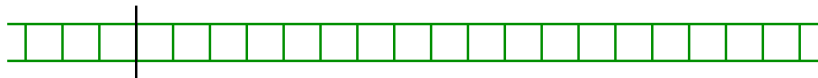
```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Le flot du programme dans la mémoire



```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Le flot du programme dans la mémoire



```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Le flot du programme dans la mémoire



```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Le flot du programme dans la mémoire



```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Le flot du programme dans la mémoire



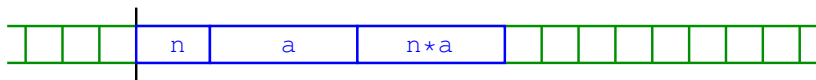
```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
        << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Le flot du programme dans la mémoire



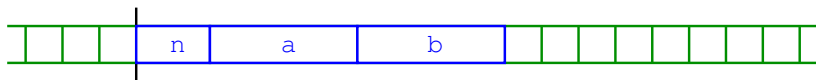
```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```


Le flot du programme dans la mémoire



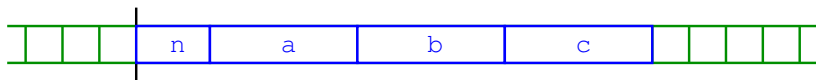
```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Le flot du programme dans la mémoire



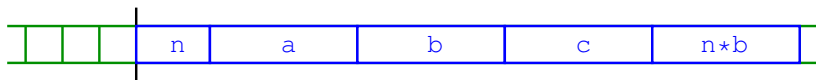
```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Le flot du programme dans la mémoire



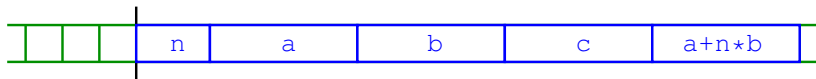
```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Le flot du programme dans la mémoire



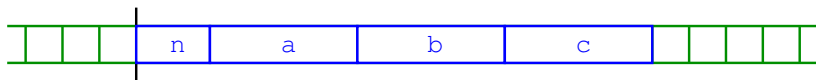
```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Le flot du programme dans la mémoire



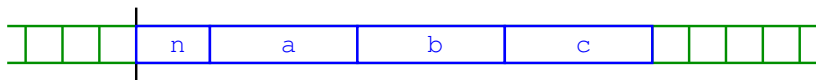
```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Le flot du programme dans la mémoire



```
#include <iostream>
using namespace std;

int main()
{
    cout << "bonjour" << endl;

    // premiere partie
    int n = 2;
    double a;
    cout << "tapez un nombre reel: ";
    cin >> a;
    cout << "voila le resultat d'une "
         << "operation arithmetique: " << n*a << endl;

    // deuxieme partie
    double b = 1.5e-3;
    double c ;
    c = a + n * b;
    cout << "voila une autre operation:" << c << endl;
}
```

Quelques règles de syntaxe

- ▶ chaque instruction est terminée par un point-virgule
- ▶ les instructions sont exécutées l'une après l'autre dans l'ordre de leur apparition
- ▶ une instruction peut contenir plusieurs sous-instructions :

$$c = a + n * b$$

→ multiplication, addition, copie du résultat dans c

Quelques règles de syntaxe

- ▶ chaque instruction est terminée par un point-virgule
- ▶ les instructions sont exécutées l'une après l'autre dans l'ordre de leur apparition
- ▶ une instruction peut contenir plusieurs sous-instructions
- ▶ des commentaires sont ignorés
- ▶ tout ce qui est inclus entre guillemets " . . . " est considéré comme un "string" (une chaîne de caractères)

Vos libertés en tant que programmeurs en C/C++

Vous avez la liberté de

- ▶ laisser autant d'espace
 - ▶ entre deux instructions voisines et
 - ▶ entre deux mots/opérateurs consécutifs
- que vous voulez

```
int a = 10;  
int b = a + 2;
```

Vos libertés en tant que programmeurs en C/C++

Vous avez la liberté de

- ▶ laisser autant d'espace
 - ▶ entre deux instructions voisines et
 - ▶ entre deux mots/opérateurs consécutifs

que vous voulez

```
int
a =
10
; int b= a    +2;
```

Vos libertés en tant que programmeurs en C/C++

Vous avez la liberté de

- ▶ laisser autant d'espace
 - ▶ entre deux instructions voisines et
 - ▶ entre deux mots/opérateurs consécutifs
- que vous voulez

```
int a=10;int b=a+2;
```

Vos libertés en tant que programmeurs en C/C++

Vous avez la liberté de

- ▶ laisser autant d'espace
 - ▶ entre deux instructions voisines et
 - ▶ entre deux mots/opérateurs consécutifs
- que vous voulez

```
inta = 10;  
int b = a + 2;
```

→ erreur:
'inta' not declared

Vos libertés en tant que programmeurs en C/C++

Vous avez la liberté de

- ▶ laisser autant d'espace
 - ▶ entre deux instructions voisines et
 - ▶ entre deux mots/opérateurs consécutifsque vous voulez
- ▶ choisir n'importe quels noms de n'importe quelle longueur pour les variables utilisées (à des exceptions près) tant qu'il n'y ait pas d'ambiguïté

```
int a = 10;  
int b = a + 2;
```

Vos libertés en tant que programmeurs en C/C++

Vous avez la liberté de

- ▶ laisser autant d'espace
 - ▶ entre deux instructions voisines et
 - ▶ entre deux mots/opérateurs consécutifsque vous voulez
- ▶ choisir n'importe quels noms de n'importe quelle longueur pour les variables utilisées (à des exceptions près) tant qu'il n'y ait pas d'ambiguïté

```
int abc123 = 10;  
int i1j2 = abc123 + 2;
```

Vos libertés en tant que programmeurs en C/C++

Vous avez la liberté de

- ▶ laisser autant d'espace
 - ▶ entre deux instructions voisines et
 - ▶ entre deux mots/opérateurs consécutifsque vous voulez
- ▶ choisir n'importe quels noms de n'importe quelle longueur pour les variables utilisées (à des exceptions près) tant qu'il n'y ait pas d'ambiguïté

```
int une_variable = 10;
```

```
int uneAutreVariable = une_variable + 2;
```

Vos libertés en tant que programmeurs en C/C++

Vous avez la liberté de

- ▶ laisser autant d'espace
 - ▶ entre deux instructions voisines et
 - ▶ entre deux mots/opérateurs consécutifsque vous voulez
- ▶ choisir n'importe quels noms de n'importe quelle longueur pour les variables utilisées (à des exceptions près) tant qu'il n'y ait pas d'ambiguïté

```
int une_variable = 10;  
int double = une_variable + 2;
```

→ **erreur:**
'double' déjà réservé

Vos libertés en tant que programmeurs en C/C++

Vous avez la liberté de

- ▶ laisser autant d'espace
 - ▶ entre deux instructions voisines et
 - ▶ entre deux mots/opérateurs consécutifsque vous voulez
- ▶ choisir n'importe quels noms de n'importe quelle longueur pour les variables utilisées (à des exceptions près) tant qu'il n'y ait pas d'ambiguïté

```
int une_variable = 10;  
int if = une_variable + 2;
```

→ **erreur:**
'if' déjà réservé

Vos libertés en tant que programmeurs en C/C++

Vous avez la liberté de

- ▶ laisser autant d'espace
 - ▶ entre deux instructions voisines et
 - ▶ entre deux mots/opérateurs consécutifsque vous voulez
- ▶ choisir n'importe quels noms de n'importe quelle longueur pour les variables utilisées (à des exceptions près) tant qu'il n'y ait pas d'ambiguïté

```
int a = 10;
```

```
int a = 2;
```

→ **erreur:**
redeclaration of 'int a'

Vos libertés en tant que programmeurs en C/C++

Vous avez la liberté de

- ▶ laisser autant d'espace
 - ▶ entre deux instructions voisines et
 - ▶ entre deux mots/opérateurs consécutifsque vous voulez
- ▶ choisir n'importe quels noms de n'importe quelle longueur pour les variables utilisées (à des exceptions près) tant qu'il n'y ait pas d'ambiguïté

```
int a = 10;
```

```
int 2a = 2;
```

→ **erreur:**

invalid suffix "a" on integer constant

Vos libertés en tant que programmeurs en C/C++

Vous avez la liberté de

- ▶ laisser autant d'espace
 - ▶ entre deux instructions voisines et
 - ▶ entre deux mots/opérateurs consécutifsque vous voulez
- ▶ choisir n'importe quels noms de n'importe quelle longueur pour les variables utilisées (à des exceptions près) tant qu'il n'y ait pas d'ambiguïté

- ▶ utiliser toutes les lettres de l'alphabet

A B ... Z a b ... z, tous les chiffres 0 1 ... 9

et la souligne _ pour composer des noms

... avec la restriction que les noms ne doivent pas commencer avec un chiffre

→ ambiguïté avec des valeurs numériques

Vos libertés en tant que programmeurs en C/C++

Vous avez la liberté de

- ▶ laisser autant d'espace
 - ▶ entre deux instructions voisines et
 - ▶ entre deux mots/opérateurs consécutifsque vous voulez
- ▶ choisir n'importe quels noms de n'importe quelle longueur pour les variables utilisées (à des exceptions près) tant qu'il n'y ait pas d'ambiguïté
- ▶ utiliser toutes les lettres de l'alphabet
A B ... Z a b ... z, tous les chiffres 0 1 ... 9
et la souligne _ pour composer des noms
- ▶ écrire tout ce que vous voulez dans des commentaires

Vos libertés en tant que programmeurs en C/C++

Il est néanmoins conseillé

- ▶ de maintenir une structure cohérente dans l'écriture des instructions de vos programmes
- ▶ de choisir des noms de vos variables d'une manière évidente
- ▶ de placer des commentaires d'une manière intelligente

afin de rendre vos programmes bien lisibles pour vous

4.3 Les variables élémentaires

Les nombres entiers

type	taille	nombre minimal	nombre maximal
short	2 octets	$-2^{15} = -32768$	$2^{15} - 1 = 32767$
long	4 octets	$-2^{31} =$ $-2\,147\,483\,648$	$2^{31} - 1 =$ $2\,147\,483\,647$

4.3 Les variables élémentaires

Les nombres entiers

type	taille	nombre minimal	nombre maximal
short	2 octets	$-2^{15} = -32768$	$2^{15} - 1 = 32767$
long	4 octets	-2^{31}	$2^{31} - 1$
<code>int</code>	4 octets	-2^{31}	$2^{31} - 1$

→ on utilise plus souvent `int`

4.3 Les variables élémentaires

Les nombres entiers

type	taille	nombre minimal	nombre maximal
unsigned short	2 octets	0	$2^{16} - 1 = 65525$
unsigned long	4 octets	0	$2^{32} - 1 =$ 4 294 967 295

4.3 Les variables élémentaires

Les nombres entiers

type	taille	nombre minimal	nombre maximal
unsigned short	2 octets	0	$2^{16} - 1 = 65525$
unsigned long	4 octets	0	$2^{32} - 1$
unsigned int	4 octets	0	$2^{32} - 1$

4.3 Les variables élémentaires

Les nombres réels / flottants

type	taille	nombre minimal	nombre maximal
float	4 octets	$\pm 1,2 \times 10^{-38}$	$\pm 3,4 \times 10^{38}$
double	8 octets	$\pm 2,2 \times 10^{-308}$	$\pm 1,8 \times 10^{308}$

→ on utilise plus souvent double

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

```
#include <iostream>
using namespace std;

int main()
{
    double a = -13.272e-7;
    double b = 0.0012e2;
    double c = 113450002.4;
    cout << a << endl << b << endl << c << endl;
}
```

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

```
double a = -13.272e-7;  
double b = 0.0012e2;  
double c = 113450002.4;  
cout << a << endl << b << endl << c << endl;
```

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

```
double a = -13.272e-7;  
double b = 0.0012e2;  
double c = 113450002.4;  
cout << a << endl << b << endl << c << endl;
```

Exécution:

```
-1.3272e-06  
0.12  
1.1345e+08
```

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

```
double a = -13.272e-7;  
double b = 0.0012e2;  
double c = 113450002.4;  
cout.precision(15);  
cout << a << endl << b << endl << c << endl;
```

Exécution:

```
-1.3272e-06  
0.12  
113450002.4
```

→ affichage avec 15 chiffres significatifs

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

Expression explicite des nombres entiers

$\langle \pm \rangle \langle \text{chiffres} \rangle$

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

Expression explicite des nombres entiers

$\langle \pm \rangle \langle \text{chiffres} \rangle$

```
int a = 23;  
cout << a << endl;
```

Exécution:

23

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

Expression explicite des nombres entiers

$\langle \pm \rangle \langle \text{chiffres} \rangle$

```
int a = 23.0; ← nombre flottant  
cout << a << endl;
```

Exécution:

erreur ?

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

Expression explicite des nombres entiers

$\langle \pm \rangle \langle \text{chiffres} \rangle$

```
int a = 23.0; ← nombre flottant  
cout << a << endl;
```

Exécution:

23

→ conversion implicite des flottants en entiers
avec une perte de précision

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

Expression explicite des nombres entiers

$\langle \pm \rangle \langle \text{chiffres} \rangle$

```
int a = 23.4;  
cout << a << endl;
```

Exécution:

23

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

Expression explicite des nombres entiers

$\langle \pm \rangle \langle \text{chiffres} \rangle$

```
int a = 23.9;  
cout << a << endl;
```

Exécution:

23

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

Expression explicite des nombres entiers

$\langle \pm \rangle \langle \text{chiffres} \rangle$

```
double a = 23; ← nombre entier  
cout << a << endl;
```

Exécution:

23

→ conversion implicite des entiers en flottants
sans perte de précision

4.3 Les variables élémentaires

Les caractères

type	taille	nombre minimal	nombre maximal
char	1 octet	0	$2^8 - 1 = 255$

- les caractères se comportent comme des nombres entiers dans toutes les opérations arithmétiques
- ils représentent des **lettres** selon le code ASCII (American Standard Code for Information Interchange)

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

Expression explicite des nombres entiers

$\langle \pm \rangle \langle \text{chiffres} \rangle$

Expression explicite des caractères

$\langle \text{lettre} \rangle$

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

Expression explicite des nombres entiers

$\langle \pm \rangle \langle \text{chiffres} \rangle$

Expression explicite des caractères

$\langle \text{lettre} \rangle$

```
char x = 'A';  
char y = 'B';  
char z = 'C';  
cout << x << y << z << endl;
```

Exécution: ABC

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

Expression explicite des nombres entiers

$\langle \pm \rangle \langle \text{chiffres} \rangle$

Expression explicite des caractères

$\langle \text{lettre} \rangle$

```
char x = 65;  
char y = 66;  
char z = 67;  
cout << x << y << z << endl;
```

Exécution: ABC

4.3 Les variables élémentaires

Expression explicite des nombres flottants

$\langle \pm \rangle \langle \text{chiffres avant la virgule} \rangle . \langle \text{chiffres après la virgule} \rangle e \langle \pm \rangle \langle \text{exposant} \rangle$

Expression explicite des nombres entiers

$\langle \pm \rangle \langle \text{chiffres} \rangle$

Expression explicite des caractères

$\langle \text{lettre} \rangle$

```
char x = 68;  
char y = 69;  
char z = 70;  
cout << x << y << z << endl;
```

Exécution: DEF

Le code ASCII

(American Standard Code for Information Interchange)

bin.	d.		bin.	d.		bin.	d.	
1000001	65	A	1100001	97	a	0110000	48	0
1000010	66	B	1100010	98	b	0110001	49	1
1000011	67	C	1100011	99	c	0110010	50	2
1000100	68	D	1100100	100	d	0110011	51	3
⋮	⋮		⋮	⋮		⋮	⋮	
1011010	90	Z	1111010	122	z	0111001	57	9

Tous les caractères du code ASCII entre 32 et 126:

!"#\$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

les autres nombres sont réservés pour des

caractères de contrôle

Le code ASCII

Quelques caractères de contrôle utiles:

binaire	déc.	C/C++	effet
0000111	7	'\a'	<i>beep</i>
0001000	8	'\b'	<i>backspace</i> (retour arrière)
0001001	9	'\t'	<i>tab</i> (tabulation)
0001010	10	'\n'	<i>line feed</i> (nouvelle ligne)
0001011	11	'\v'	<i>vertical tab</i>
0001100	12	'\f'	<i>form feed</i> (nouvelle page)
0001101	13	'\r'	<i>carriage return</i> (retour chariot)

```
cout << "ceci \test \nun \ttexte" << endl;
```

Exécution:

```
ceci      est
un        texte
```

Le code ASCII

Quelques caractères de contrôle utiles:

binaire	déc.	C/C++	effet
0000111	7	'\a'	<i>beep</i>
0001000	8	'\b'	<i>backspace</i> (retour arrière)
0001001	9	'\t'	<i>tab</i> (tabulation)
0001010	10	'\n'	<i>line feed</i> (nouvelle ligne)
0001011	11	'\v'	<i>vertical tab</i>
0001100	12	'\f'	<i>form feed</i> (nouvelle page)
0001101	13	'\r'	<i>carriage return</i> (retour chariot)

```
cout << "ceci est \"un texte\" " << endl;
```

Exécution:

```
ceci est "un texte"
```

Le code ASCII

Quelques caractères de contrôle utiles:

binaire	déc.	C/C++	effet
0000111	7	'\a'	<i>beep</i>
0001000	8	'\b'	<i>backspace</i> (retour arrière)
0001001	9	'\t'	<i>tab</i> (tabulation)
0001010	10	'\n'	<i>line feed</i> (nouvelle ligne)
0001011	11	'\v'	<i>vertical tab</i>
0001100	12	'\f'	<i>form feed</i> (nouvelle page)
0001101	13	'\r'	<i>carriage return</i> (retour chariot)

```
cout << "ceci est "un texte" " << endl;
```

→ erreur:

expected ';' before 'un'

Le code ASCII

Quelques caractères de contrôle utiles:

binaire	déc.	C/C++	effet
0000111	7	'\a'	<i>beep</i>
0001000	8	'\b'	<i>backspace</i> (retour arrière)
0001001	9	'\t'	<i>tab</i> (tabulation)
0001010	10	'\n'	<i>line feed</i> (nouvelle ligne)
0001011	11	'\v'	<i>vertical tab</i>
0001100	12	'\f'	<i>form feed</i> (nouvelle page)
0001101	13	'\r'	<i>carriage return</i> (retour chariot)

```
cout << "ceci est \"un texte\" " << endl;
```

Exécution:

```
ceci est 'un texte'
```

Le code ASCII

Quelques caractères de contrôle utiles:

binaire	déc.	C/C++	effet
0000111	7	'\a'	<i>beep</i>
0001000	8	'\b'	<i>backspace</i> (retour arrière)
0001001	9	'\t'	<i>tab</i> (tabulation)
0001010	10	'\n'	<i>line feed</i> (nouvelle ligne)
0001011	11	'\v'	<i>vertical tab</i>
0001100	12	'\f'	<i>form feed</i> (nouvelle page)
0001101	13	'\r'	<i>carriage return</i> (retour chariot)

```
cout << "ceci est \\ un texte" << endl;
```

Exécution:

```
ceci est \ un texte
```

Le code ASCII

Quelques caractères de contrôle utiles:

binaire	déc.	C/C++	effet
0000111	7	'\a'	<i>beep</i>
0001000	8	'\b'	<i>backspace</i> (retour arrière)
0001001	9	'\t'	<i>tab</i> (tabulation)
0001010	10	'\n'	<i>line feed</i> (nouvelle ligne)
0001011	11	'\v'	<i>vertical tab</i>
0001100	12	'\f'	<i>form feed</i> (nouvelle page)
0001101	13	'\r'	<i>carriage return</i> (retour chariot)
0100010	34	'\"'	"
0100111	39	'\''	'
1011100	92	'\\'	\

La déclaration des variables

```
double a;
```

→ déclaration d'une variable nommée "a"
elle est du type `double`

La déclaration des variables

```
double a;  
int n;
```

→ déclaration d'une variable nommée "n"
elle est du type `int`

La déclaration des variables

```
double a;  
int n;  
unsigned int k1;
```

→ déclaration d'une variable nommée "k1"
elle est du type `unsigned int`

La déclaration des variables

```
double a;  
int n;  
unsigned int k1;  
float long_name;
```

→ **déclaration d'une variable nommée "long_name"**
elle est du type `float`

La déclaration des variables

La définition d'une variable peut être combinée avec son initialisation ...

```
double a = 3.1;
```

est équivalent à

```
double a;
```

```
a = 3.1;
```


La déclaration des variables

La définition d'une variable peut être combinée avec son initialisation ...

```
double a = 3.1;
```

...et on peut même définir plusieurs variables avec une seule instruction:

```
double a, b, c;
```

est équivalent à

```
double a;
```

```
double b;
```

```
double c;
```

La déclaration des variables

La définition d'une variable peut être combinée avec son initialisation ...

```
double a = 3.1;
```

... et on peut même définir plusieurs variables avec une seule instruction:

```
double a = 3.1, b = -1.5e-3, c = 35e5;
```

est équivalent à

```
double a;  
double b;  
double c;  
a = 3.1;  
b = -1.5e-3;  
c = 35e5;
```

La déclaration des variables

La définition d'une variable peut être combinée avec son initialisation ...

```
double a = 3.1;
```

...et on peut même définir plusieurs variables avec une seule instruction:

```
double a = 3.1, b = -1.5e-3, c = 35e5;
```

Il faut d'abord définir une variable avant de l'utiliser ...

```
double a;
```

```
a = 3;
```

La déclaration des variables

La définition d'une variable peut être combinée avec son initialisation ...

```
double a = 3.1;
```

...et on peut même définir plusieurs variables avec une seule instruction:

```
double a = 3.1, b = -1.5e-3, c = 35e5;
```

Il faut d'abord définir une variable avant de l'utiliser ...

```
a = 3;
```

```
double a;
```

→ `erreur: 'a' was not declared`

La déclaration des variables

La définition d'une variable peut être combinée avec son initialisation ...

```
double a = 3.1;
```

...et on peut même définir plusieurs variables avec une seule instruction:

```
double a = 3.1, b = -1.5e-3, c = 35e5;
```

Il faut d'abord définir une variable avant de l'utiliser ...

...et on ne peut pas donner le même nom à deux variables différentes:

```
double a;
```

```
int a = 1;
```

→ **erreur: 'a' has a previous declaration as 'double a'**

La définition des constantes

```
const int n = 100;
```

→ déclaration d'une constante nommée "n"
elle est du type `int` et elle porte la valeur 100

La définition des constantes

```
const int n = 100;  
const double b = 1.5e-14;
```

→ déclaration d'une constante nommée "b"
elle est du type `double` et elle porte la valeur 1.5×10^{-14}

La définition des constantes

```
const int n = 100;  
const double b = 1.5e-14;  
n = 3;
```

→ `erreur: assignment of read-only variable 'n'`

→ on n'a pas le droit de changer la valeur d'une constante après sa déclaration

La définition des constantes

```
const int n = 100;  
const double b = 1.5e-14;  
const double pi = 3.14159265359;
```

Des constantes peuvent être utiles dans des contextes différents:

- définition des nombres comme $\pi = 3.14159\dots$
- définition des tailles des tableaux
- définition des limites de précision
- ...

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

```
int n = 5;
```

→ la valeur 5 est copiée dans `n`

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

```
int n = 5;  
double a = 0.2;
```

→ la valeur 0.2 est copiée dans a

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

```
int n = 5;  
double a = 0.2;  
double b = 1.5e-3;
```

→ la valeur 0.0015 est copiée dans b

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

```
int n = 5;  
double a = 0.2;  
double b = 1.5e-3;  
b = -2.01;
```

→ maintenant b porte la valeur -2.01

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

```
int n = 5;  
double a = 0.2;  
double b = 1.5e-3;  
b = -2.01;  
double c;  
c = a + n * b;
```

→ la valeur de l'expression $a + n \times b$ est copiée dans c
(= -9.85)

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

```
int n = 5;  
double a = 0.2;  
double b = 1.5e-3;  
b = -2.01;  
double c;  
a + n * b = c;
```

→ erreur:

lvalue required as left operand of assignment

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

```
int n = 5;  
double a = 0.2;  
double b = 1.5e-3;  
b = -2.01;  
double c;  
-2.01 = b;
```

→ erreur:

lvalue required as left operand of assignment

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

```
int n = 5;  
double a = 0.2;  
double b = 1.5e-3;  
b = -2.01;  
double c;  
-2.01 = b;
```

→ **erreur:**

lvalue required as left operand of assignment

→ l'opérateur de copie n'est pas symétrique !

$\langle \text{variable} \rangle = \langle \text{expression} \rangle;$

4.4 Les opérations arithmétiques élémentaires

- = opérateur de copie
- + opérateur d'addition
- opérateur de soustraction
- * opérateur de multiplication
- / opérateur de division

4.4 Les opérations arithmétiques élémentaires

- = opérateur de copie
- + opérateur d'addition
- opérateur de soustraction
- * opérateur de multiplication
- / opérateur de division

```
int n = 2;  
double a = 3.2;  
double b = 1.5;  
double c;  
c = a + n * b;
```

→ multiplication de n avec b (conversion implicite en `double`)
addition du résultat avec a
copie du résultat final dans c :
 $c = a + n \times b = 6.2$

4.4 Les opérations arithmétiques élémentaires

- = opérateur de copie
- + opérateur d'addition
- opérateur de soustraction
- * opérateur de multiplication
- / opérateur de division

```
int n = 2;  
double a = 3.2;  
double b = 1.5;  
double c;  
c = a + n * b;
```

→ et si on voulait faire autrement: $c = (a + n) \times b$?

4.4 Les opérations arithmétiques élémentaires

- = opérateur de copie
- + opérateur d'addition
- opérateur de soustraction
- * opérateur de multiplication
- / opérateur de division

```
int n = 2;  
double a = 3.2;  
double b = 1.5;  
double c;  
c = ( a + n ) * b;
```

→ addition de n avec a (conversion implicite en double)
multiplication du résultat avec b
copie du résultat final dans c :
 $c = (a + n) \times b = 7.8$

4.4 Les opérations arithmétiques élémentaires

- = opérateur de copie
- + opérateur d'addition
- opérateur de soustraction
- * opérateur de multiplication
- / opérateur de division

```
int n = 2;  
double a = 3.2;  
double b = 1.5;  
double c;  
c = b * ( a + n );
```

→ addition de n avec a (conversion implicite en double)
multiplication du résultat avec b
copie du résultat final dans c :
 $c = (a + n) \times b = 7.8$

4.4 Les opérations arithmétiques élémentaires

- = opérateur de copie
- + opérateur d'addition
- opérateur de soustraction
- * opérateur de multiplication
- / opérateur de division

```
int n = 2;  
double a = 3.2;  
double b = 1.5;  
double c;  
c = ( a + n ) * ( b + 3 - ( a - 2 ) / 5 );
```

→ n'utilisez pas les crochets `[]` ni les accolades `{ }`
dans ce contexte-là !

4.4 Les opérations arithmétiques élémentaires

- = opérateur de copie
- + opérateur d'addition
- opérateur de soustraction
- * opérateur de multiplication
- / opérateur de division → différent pour des entiers

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

```
int n = 2;
```

```
int p = 3;
```

```
cout << n + p << endl;
```

Exécution:

5

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

```
int n = 2;
```

```
int p = 3;
```

```
cout << n - p << endl;
```

Exécution:

-1

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

```
int n = 2;
```

```
int p = 3;
```

```
cout << n * p << endl;
```

Exécution:

6

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

```
int n = 2;
```

```
int p = 3;
```

```
cout << n / p << endl;
```

Exécution:

0 0

n / p = l'entier **devant la virgule** du nombre rationnel $\frac{n}{p}$

$$\frac{n}{p} = \frac{2}{3} = 0.666 \dots \quad \longleftrightarrow \quad n / p \Rightarrow 0$$

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

```
int n = 2;
```

```
int p = 3;
```

```
cout << n / p << endl;
```

Exécution:

0 0

n / p = l'entier **devant la virgule** du nombre rationnel $\frac{n}{p}$

$$\frac{p}{n} = \frac{3}{2} = 1.5 \quad \longleftrightarrow \quad p / n \Rightarrow 1$$

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

```
int n = 2;
```

```
int p = 3;
```

```
cout << p / n << endl;
```

Exécution:

1

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

```
int n = 2;
```

```
int p = 3;
```

```
cout << 2 / 3 << endl;
```

Exécution:

0

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

```
int n = 2;  
int p = 3;  
cout << 2.0 / 3 << endl;
```

Exécution:

0.666667

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

```
int n = 2;  
int p = 3;  
cout << 2 / 3.0 << endl;
```

Exécution:

0.666667

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

```
int n = 2;
```

```
int p = 3;
```

```
cout << double( n ) / double( p ) << endl;
```

Exécution:

0.666667

→ conversion **explicite** entre entiers et flottants

(double) n / (double) p marcherait également

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

Attention !

```
double a = 1.2;  
double b = 3.2;  
double c = 1 / 2 * ( a + b ) ;  
cout << c << endl;
```

Exécution:

0

4.4 Les opérations arithmétiques élémentaires

- = opérateur de copie
- + opérateur d'addition
- opérateur de soustraction
- * opérateur de multiplication
- / opérateur de division → différent pour des entiers

Attention !

```
double a = 1.2;  
double b = 3.2;  
double c = 1 / 2 * ( a + b ) ;  
cout << c << endl;
```

→ division des entiers avant la multiplication avec $(a+b)$:

$$\begin{aligned} & 1/2 * (a+b) \\ &= 0 * 4.4 = 0 \end{aligned}$$

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

Attention !

```
double a = 1.2;  
double b = 3.2;  
double c = 1.0 / 2 * ( a + b ) ;  
cout << c << endl;
```

Exécution:

2.2

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

Attention !

```
double a = 1.2;  
double b = 3.2;  
double c = 0.5 * ( a + b ) ;  
cout << c << endl;
```

Exécution:

2.2

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

Attention !

```
double a = 1.2;  
double b = 3.2;  
double c = ( a + b ) / 2 ;  
cout << c << endl;
```

Exécution:

2.2

4.4 Les opérations arithmétiques élémentaires

- = opérateur de copie
- + opérateur d'addition
- opérateur de soustraction
- * opérateur de multiplication
- / opérateur de division → différent pour des entiers
- % opérateur "modulo" → seulement pour des entiers
→ le reste de la division

4.4 Les opérations arithmétiques élémentaires

- = opérateur de copie
- + opérateur d'addition
- opérateur de soustraction
- * opérateur de multiplication
- / opérateur de division → différent pour des entiers
- % opérateur "modulo" → seulement pour des entiers
→ le reste de la division

```
int n = 25;  
int m = 4;  
cout << n / m << endl;
```

Exécution:

6

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

% opérateur "modulo" → seulement pour des entiers
→ le reste de la division

```
int n = 25;  
int m = 4;  
cout << n % m << endl;
```

Exécution:

1

$n \% m$ est synonyme à $n - m * (n / m)$

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

+ opérateur d'addition

- opérateur de soustraction

* opérateur de multiplication

/ opérateur de division → différent pour des entiers

% opérateur "modulo" → seulement pour des entiers
→ le reste de la division

```
int n = 27;  
int m = 4;  
cout << n % m << endl;
```

Exécution:

3

$n \% m$ est synonyme à $n - m * (n / m)$

Les opérations d'affectation

= copie et manipulation: += -= *= /= %=

Les opérations d'affectation

= copie et manipulation: += -= *= /= %=

```
int n = 5;  
n = n + 3;  
cout << n << endl;
```

Exécution:

8

→ la valeur de `n` est augmentée par 3

Les opérations d'affectation

= copie et manipulation: += -= *= /= %=

```
int n = 5;  
n += 3;  
cout << n << endl;
```

Exécution:

8

→ la valeur de `n` est augmentée par 3

Les opérations d'affectation

= copie et manipulation: += -= *= /= %=

```
int n = 5;  
n -= 3;  
cout << n << endl;
```

Exécution:

2

→ la valeur de `n` est diminuée par 3

Les opérations d'affectation

= copie et manipulation: += -= *= /= %=

```
int n = 5;  
n *= 3;  
cout << n << endl;
```

Exécution:

15

→ la valeur de `n` est multipliée par 3

Les opérations d'affectation

= copie et manipulation: += -= *= /= %=

```
int n = 5;  
n /= 3;  
cout << n << endl;
```

Exécution:

1

→ la valeur de `n` est divisée par 3 (division des entiers)

Les opérations d'affectation

= copie et manipulation: += -= *= /= %=

```
int n = 5;  
n %= 3;  
cout << n << endl;
```

Exécution:

2

→ la valeur de `n` subit une opération modulo par 3

Les opérations d'affectation

= copie et manipulation: += -= *= /= %=

$\langle \text{variable} \rangle += \langle \text{expression} \rangle ; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle + \langle \text{expression} \rangle ;$

$\langle \text{variable} \rangle -= \langle \text{expression} \rangle ; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle - \langle \text{expression} \rangle ;$

$\langle \text{variable} \rangle *= \langle \text{expression} \rangle ; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle * \langle \text{expression} \rangle ;$

$\langle \text{variable} \rangle /= \langle \text{expression} \rangle ; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle / \langle \text{expression} \rangle ;$

$\langle \text{variable} \rangle \% = \langle \text{expression} \rangle ; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle \% \langle \text{expression} \rangle ;$

Les opérations d'affectation

= copie et manipulation: += -= *= /= %=

$\langle \text{variable} \rangle += \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle + \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle -= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle - \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle *= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle * \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle /= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle / \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle \% = \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle \% \langle \text{expression} \rangle;$

Ça marche également pour des nombres flottants:

```
double a = 3.2;
double b = 2.5;
a *= b;
cout << a << endl;
```

Exécution: 8

Les opérations d'affectation

= copie et manipulation: += -= *= /= %=

$\langle \text{variable} \rangle += \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle + \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle -= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle - \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle *= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle * \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle /= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle / \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle \% = \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle \% \langle \text{expression} \rangle;$

Ça marche également pour des nombres flottants:

```
double a = 3.2;  
double b = 2.5;  
a *= b;  
a += 2.0;  
cout << a << endl;
```

Exécution: 10

Les opérations d'affectation

= copie et manipulation: += -= *= /= %=

$\langle \text{variable} \rangle += \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle + \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle -= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle - \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle *= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle * \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle /= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle / \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle \% = \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle \% \langle \text{expression} \rangle;$

Ça marche également pour des nombres flottants:

```
double a = 3.2;
double b = 2.5;
a *= b;
a += 2.0;
a /= 4;
cout << a << endl;
```

Exécution: 2.5 → la “vraie” division

Les opérations d'affectation

= copie et manipulation: += -= *= /= %=

$\langle \text{variable} \rangle += \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle + \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle -= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle - \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle *= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle * \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle /= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle / \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle \% = \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle \% \langle \text{expression} \rangle;$

Les opérateurs d'incrément: ++ --

$\langle \text{variable} \rangle ++; \Leftrightarrow \langle \text{variable} \rangle += 1; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle + 1;$

$\langle \text{variable} \rangle --; \Leftrightarrow \langle \text{variable} \rangle -= 1; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle - 1;$

Les opérations d'affectation

= copie et manipulation: += -= *= /= %=

$\langle \text{variable} \rangle += \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle + \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle -= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle - \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle *= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle * \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle /= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle / \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle \% = \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle \% \langle \text{expression} \rangle;$

Les opérateurs d'incrément: ++ --

$\langle \text{variable} \rangle ++; \Leftrightarrow \langle \text{variable} \rangle += 1; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle + 1;$

$\langle \text{variable} \rangle --; \Leftrightarrow \langle \text{variable} \rangle -= 1; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle - 1;$

```
int n = 3;  
n++;  
cout << n << endl;
```

Exécution: 4

Les opérations d'affectation

= copie et manipulation: += -= *= /= %=

$\langle \text{variable} \rangle += \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle + \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle -= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle - \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle *= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle * \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle /= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle / \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle \% = \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle \% \langle \text{expression} \rangle;$

Les opérateurs d'incrément: ++ --

$++\langle \text{variable} \rangle; \Leftrightarrow \langle \text{variable} \rangle += 1; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle + 1;$

$--\langle \text{variable} \rangle; \Leftrightarrow \langle \text{variable} \rangle -= 1; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle - 1;$

```
int n = 3;  
++n;  
cout << n << endl;
```

Exécution: 4

Les opérations d'affectation

= copie et manipulation: += -= *= /= %=

$\langle \text{variable} \rangle += \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle + \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle -= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle - \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle *= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle * \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle /= \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle / \langle \text{expression} \rangle;$

$\langle \text{variable} \rangle \% = \langle \text{expression} \rangle; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle \% \langle \text{expression} \rangle;$

Les opérateurs d'incrément: ++ --

$\langle \text{variable} \rangle ++; \Leftrightarrow \langle \text{variable} \rangle += 1; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle + 1;$

$\langle \text{variable} \rangle --; \Leftrightarrow \langle \text{variable} \rangle -= 1; \Leftrightarrow \langle \text{variable} \rangle = \langle \text{variable} \rangle - 1;$

C++ signifie: "le langage de programmation C avancé par 1"