4 La programmation élémentaire en C/C++

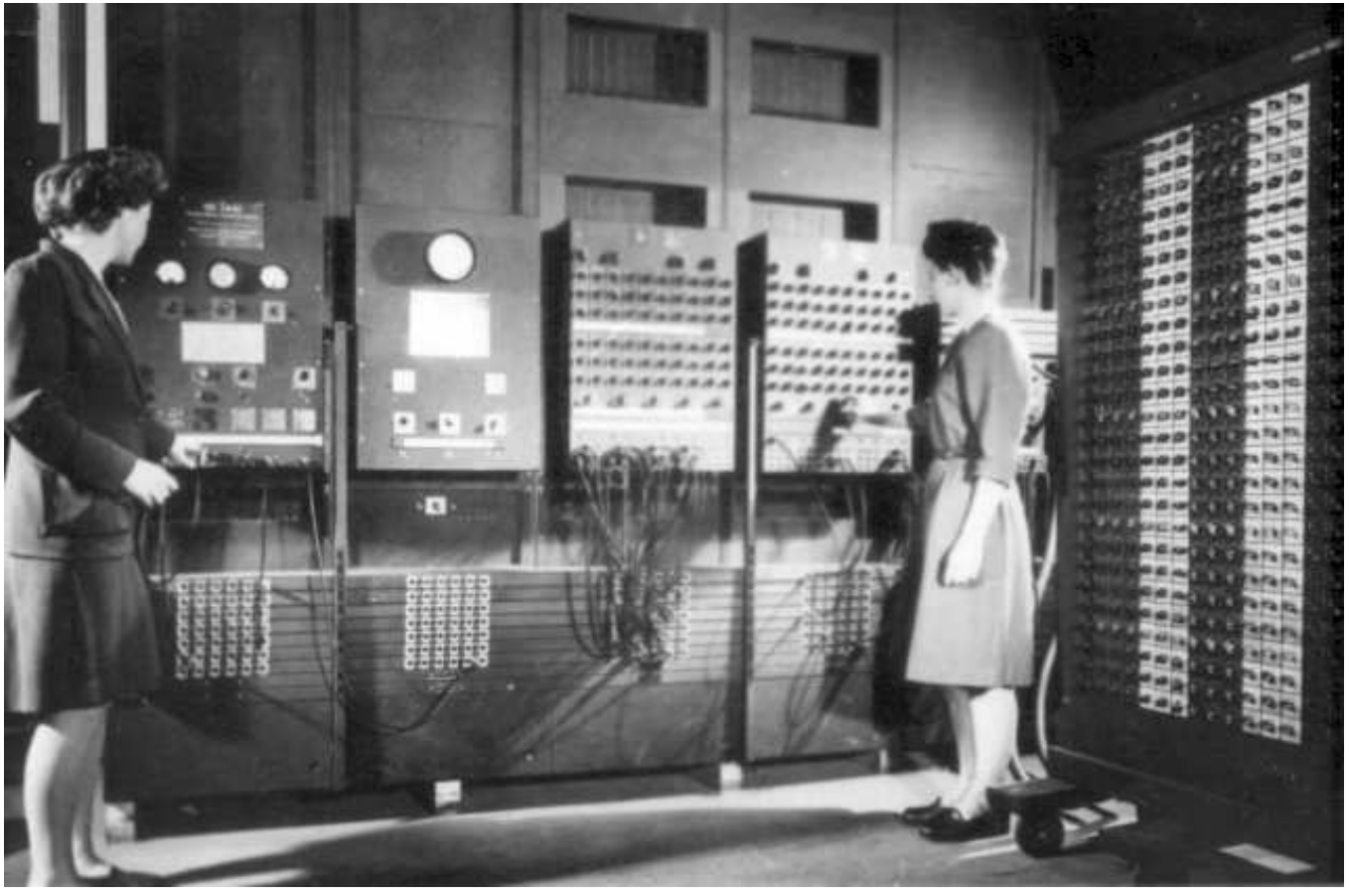
Peter Schlagheck

Université de Liège

4 La programmation élémentaire en C/C++

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:

  $\longrightarrow$ création d'un fichier de texte (ASCII) avec un tel code

  $\longrightarrow$ 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:

$\longrightarrow$ création d'un fichier de texte (ASCII) avec un tel code

$\longrightarrow$ 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:

$\longrightarrow$ création d'un fichier de texte (ASCII) avec un tel code

$\longrightarrow$ traduction en langage de machine par un programme "assembleur" et placement dans la mémoire

$\longrightarrow$ 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: ...
```

$\longrightarrow$ représentation directe du langage de machine sous une forme "lisible" (par des humains)

$\longrightarrow$ difficile d'encoder des opérations composées (comme $a = b + \sqrt{c - d/2}$)

$\longrightarrow$ 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

  $\longrightarrow$ introduit en 1956 par IBM

  $\longrightarrow$ 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

  $\longrightarrow$ introduit en 1956 par IBM

  $\longrightarrow$ fortement utilisé pour des programmes mathématiques et des simulations numériques

  $\longrightarrow$ plusieurs révisions et évolutions:

  $\rightarrow$ Fortran 66 (1966)
  $\rightarrow$ Fortran 77 (1978)
  $\rightarrow$ Fortran 90
  $\rightarrow$ Fortran 95
  $\rightarrow$ Fortran 2003
  $\rightarrow$ Fortran 2008
  $\rightarrow$ ...

4.1 Les langages de programmation

- les langages d'assembleur
- Fortran
- BASIC
  (Beginner's All-purpose Symbolic Instruction Code)

  $\longrightarrow$ conçu en 1963 (Dartmouth College, USA) comme langage d'apprentissage de la programmation

  $\longrightarrow$ très facile à écrire des programmes

  $\longrightarrow$ grande popularité dans les années 70/80

  $\longrightarrow$ début du succès de la société Microsoft
  ($\rightarrow$ 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

  $\longrightarrow$ introduit en 1970 (Niklaus Wirth)
  basé sur le langage ALGOL (1958)

  $\longrightarrow$ conçu pour permettre de structurer des algorithmes
  (`if`, `while`, `for`, `case`) et des données
  (tableaux, pointeurs, ...)

  $\longrightarrow$ grande popularité dans les années 80
  ("Turbo Pascal" de Borland)

4.1 Les langages de programmation

- les langages d'assembleur
- Fortran
- BASIC
- Pascal
- C

  $\longrightarrow$ développé en 1972 (Dennis Ritchie, Bell Labs)
  utilisé pour implémenter le système d'exploitation Unix

  $\longrightarrow$ combine les avantages d'un langage structuré
  avec l'efficacité d'un langage assembleur

  $\longrightarrow$ 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

  $\longrightarrow$ 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++

  $\longrightarrow$ développé en 1979 (Bjarne Stroustrup, Bell Labs)
  comme une espèce de "révision" de C
  avec des outils supplémentaires puissants

  $\longrightarrow$ langage de programmation "orientée objet"
  (influencé par le langage Simula)

  $\longrightarrow$ fortement utilisé par des professionnels

4.1 Les langages de programmation

- les langages d'assembleur
- Fortran
- BASIC
- Pascal
- C
- C++
- Java

  $\longrightarrow$ publié en 1995 (James Gosling, Sun Microsystems)

  $\longrightarrow$ langage de programmation orientée objet
  (influencé par C++, avec plus de "sécurité")

  $\longrightarrow$ 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

  $\longrightarrow$ première version en 1989

  $\longrightarrow$ langage de programmation multi-style

  $\longrightarrow$ 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

$$\longrightarrow \text{C++ dans le style C}$$

4.2 Un programme simple

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

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

ceci est le contenu (plain text/ASCII) 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
```

$\longrightarrow$ 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):

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

$\longrightarrow$ 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;
```

$\longrightarrow$ 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()
{
    ...
}
```

$\longrightarrow$ 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;
```

$\longrightarrow$ 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;
```

$\longrightarrow$ 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;
```

$\longrightarrow$ 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;
```

$\longrightarrow$ 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;
```

$\longrightarrow$ 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;
```

$\longrightarrow$ 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;
```

$\longrightarrow$ 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;
```

$\longrightarrow$ 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
```

$\longrightarrow$ 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 */
```

$\longrightarrow$ un autre commentaire, ignoré par le compilateur
il commence par `/*` et finit par `*/`

```
}
```

4.2 Un programme simple

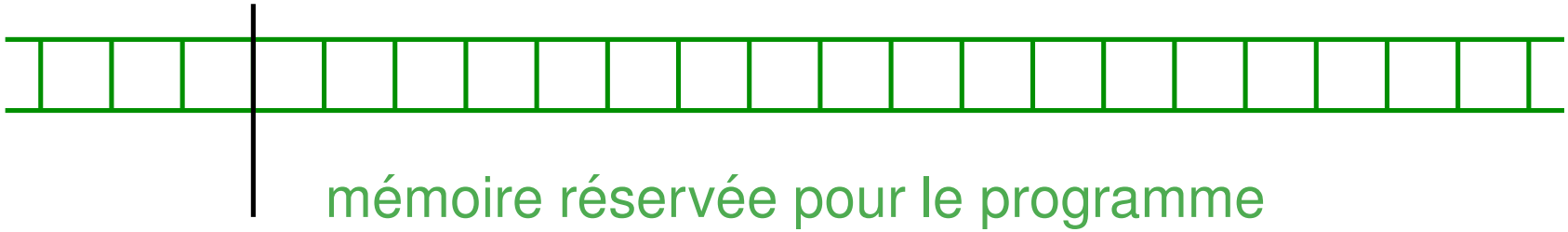
Pour l'instant on voit la structure suivante de notre programme:

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

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

    /*******************\
    |* premiere partie *|
    \*******************/
```

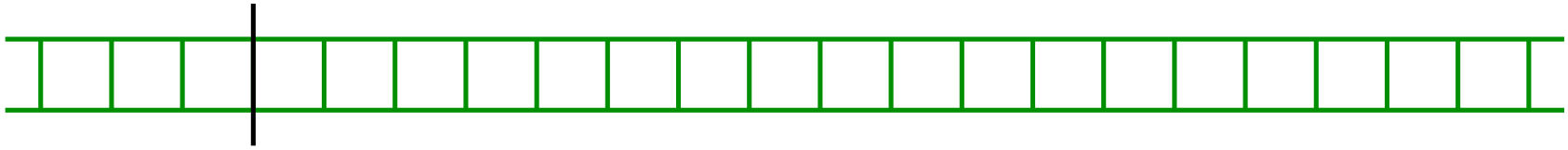
$\longrightarrow$ il peut s'étendre sur plusieurs lignes:
tout est ignoré entre `/*` et `*/`

```
}
```

Le flot du programme dans la mémoire

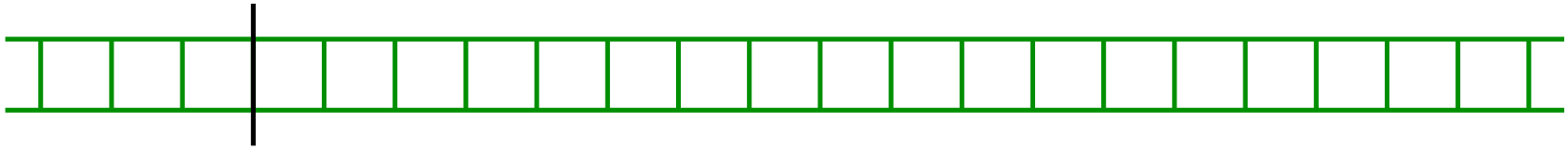


Le flot du programme dans la mémoire



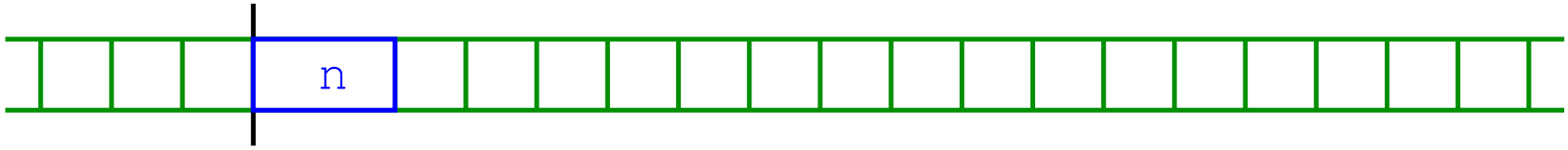
```
#include <iostream>
using namespace std;

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



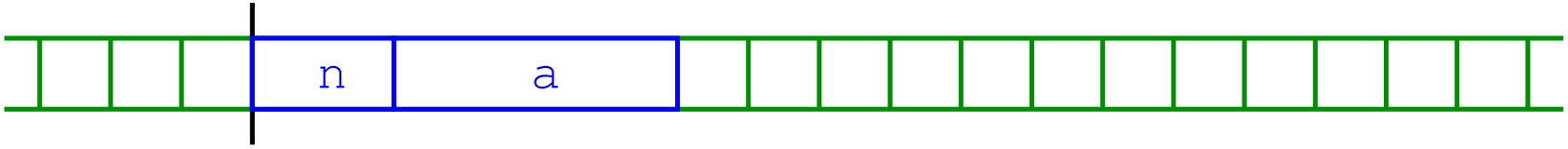
```
#include <iostream>
using namespace std;

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



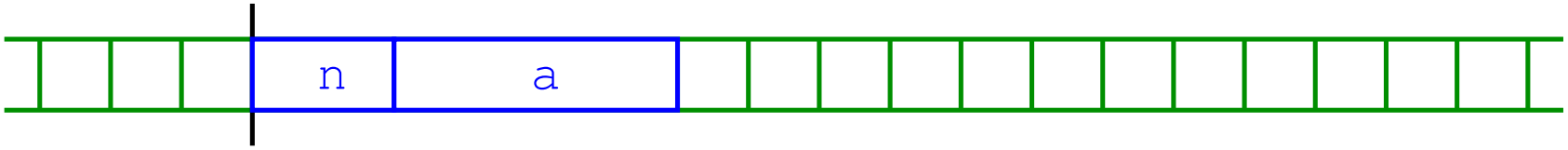

```
#include <iostream>
using namespace std;

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



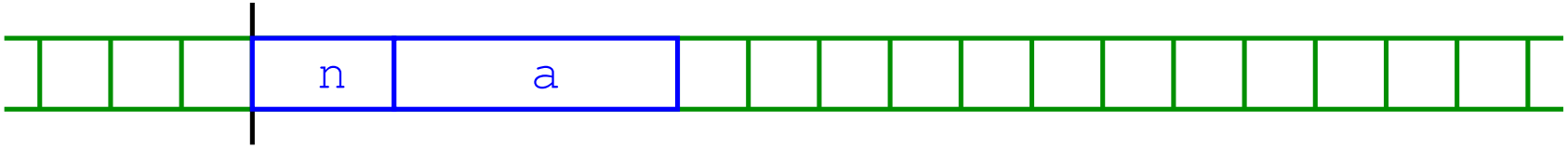

```
#include <iostream>
using namespace std;

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



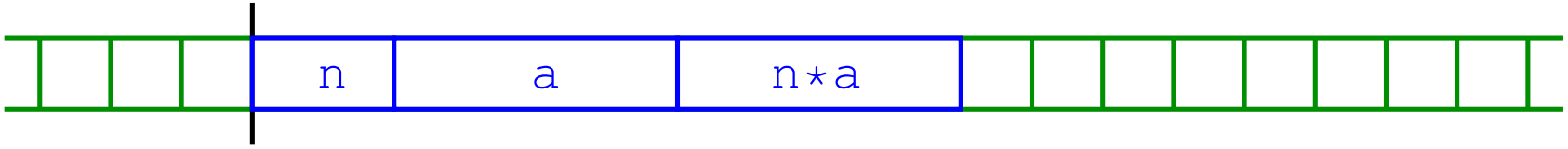

```
#include <iostream>
using namespace std;

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



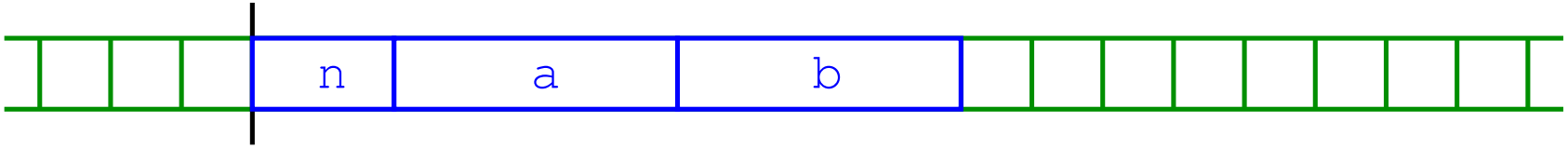

```
#include <iostream>
using namespace std;

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



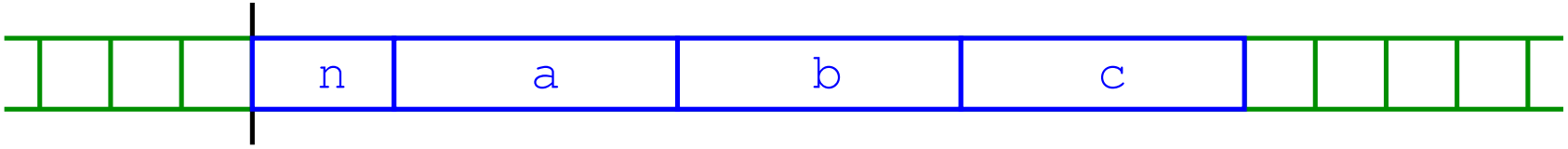

```
#include <iostream>
using namespace std;

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



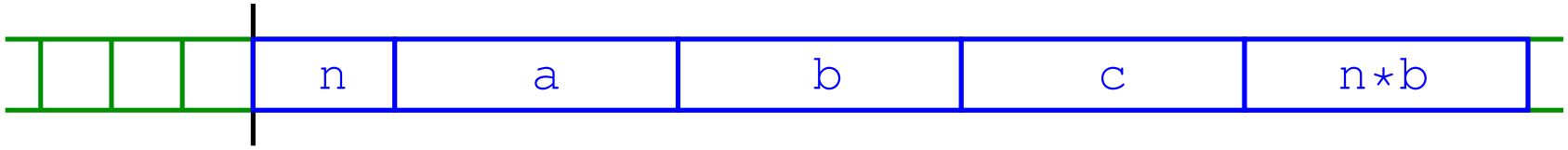

```
#include <iostream>
using namespace std;

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



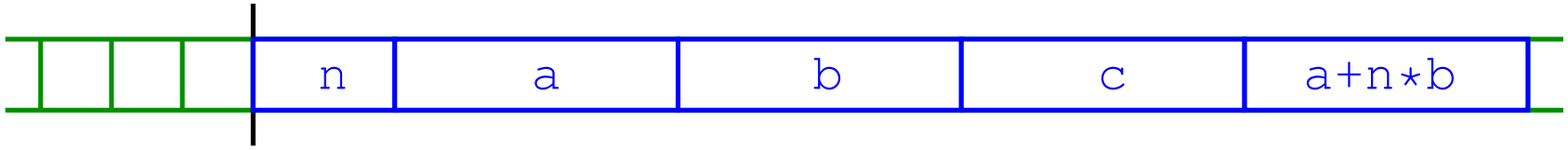

```
#include <iostream>
using namespace std;

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



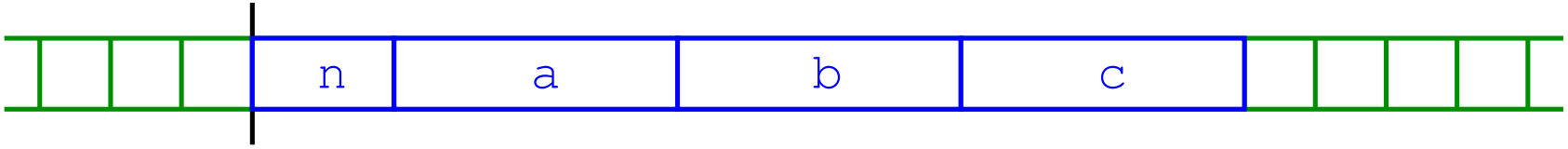

```
#include <iostream>
using namespace std;

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



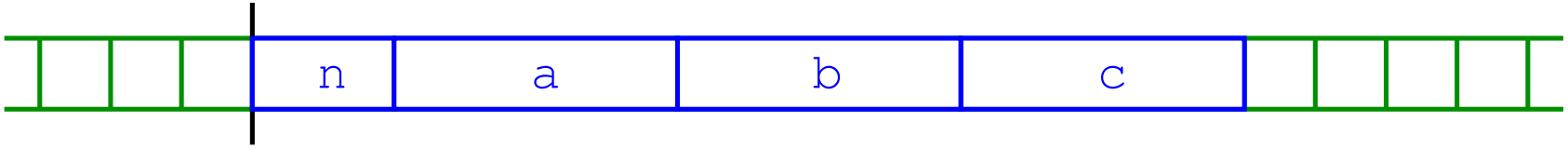

```
#include <iostream>
using namespace std;

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
```

$\longrightarrow$ 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;
```

$\longrightarrow$ `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 consécutifs

  que 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'ambiguité

```
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
- 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'ambiguité

```
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 consécutifs

  que 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'ambiguité

```
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 consécutifs

  que 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'ambiguité

```
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 consécutifs

  que 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'ambiguité

```
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 consécutifs

  que 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'ambiguité

```
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 consécutifs

  que 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'ambiguité

```
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 consécutifs

  que 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'ambiguité
- utiliser toutes les lettres de l'alphabete `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

  $\longrightarrow$ ambiguité 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 consécutifs

  que 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'ambiguité
- utiliser toutes les lettres de l'alphabete `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}	$2^{31} - 1$
`int`	4 octets	-2^{31}	$2^{31} - 1$

$\longrightarrow$ 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	$\pm 2,2 \times 10^{-308}$	$\pm 1,8 \times 10^{308}$

$\longrightarrow$ on utilise plus souvent `double`

4.3 Les variables élémentaires

Expression explicite des nombres flottants

⟨±⟩⟨chiffres avant la virgule⟩ . ⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

Expression explicite des nombres flottants

⟨±⟩⟨chiffres avant la virgule⟩.⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

```
#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;

}
```

Expression explicite des nombres flottants

⟨±⟩⟨chiffres avant la virgule⟩.⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

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

Expression explicite des nombres flottants

⟨±⟩⟨chiffres avant la virgule⟩.⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

```
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
```

Expression explicite des nombres flottants

⟨±⟩⟨chiffres avant la virgule⟩ . ⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

```
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
```

$\longrightarrow$ affichage avec 15 chiffres significatifs

4.3 Les variables élémentaires

Expression explicite des nombres flottants

⟨±⟩⟨chiffres avant la virgule⟩ . ⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

Expression explicite des nombres entiers

⟨±⟩⟨chiffres⟩

4.3 Les variables élémentaires

Expression explicite des nombres flottants

⟨±⟩⟨chiffres avant la virgule⟩ . ⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

Expression explicite des nombres entiers

⟨±⟩⟨chiffres⟩

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

Exécution:

```
23
```

4.3 Les variables élémentaires

Expression explicite des nombres flottants

⟨±⟩⟨chiffres avant la virgule⟩ . ⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

Expression explicite des nombres entiers

⟨±⟩⟨chiffres⟩

```
int a = 23.0;
```
← nombre flottant

```
cout << a << endl;
```

Exécution:

erreur ?

4.3 Les variables élémentaires

Expression explicite des nombres flottants

⟨±⟩⟨chiffres avant la virgule⟩ . ⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

Expression explicite des nombres entiers

⟨±⟩⟨chiffres⟩

```
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

⟨±⟩⟨chiffres avant la virgule⟩ . ⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

Expression explicite des nombres entiers

⟨±⟩⟨chiffres⟩

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

Exécution:

```
23
```

4.3 Les variables élémentaires

Expression explicite des nombres flottants

⟨±⟩⟨chiffres avant la virgule⟩ . ⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

Expression explicite des nombres entiers

⟨±⟩⟨chiffres⟩

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

Exécution:

```
23
```

4.3 Les variables élémentaires

Expression explicite des nombres flottants

⟨±⟩⟨chiffres avant la virgule⟩ . ⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

Expression explicite des nombres entiers

⟨±⟩⟨chiffres⟩

```
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	$2^8 - 1 = 255$

$\longrightarrow$ les caractères se comportent comme des nombres entiers dans toutes les opérations arithmétiques

$\longrightarrow$ ils représentent des lettres selon le code ASCII

4.3 Les variables élémentaires

Expression explicite des nombres flottants

⟨±⟩⟨chiffres avant la virgule⟩ . ⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

Expression explicite des nombres entiers

⟨±⟩⟨chiffres⟩

Expression explicite des caractères

' ⟨lettre⟩ '

4.3 Les variables élémentaires

Expression explicite des nombres flottants

⟨±⟩⟨chiffres avant la virgule⟩ . ⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

Expression explicite des nombres entiers

⟨±⟩⟨chiffres⟩

Expression explicite des caractères

'⟨lettre⟩'

```
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

⟨±⟩⟨chiffres avant la virgule⟩ . ⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

Expression explicite des nombres entiers

⟨±⟩⟨chiffres⟩

Expression explicite des caractères

'⟨lettre⟩'

```
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

⟨±⟩⟨chiffres avant la virgule⟩ . ⟨chiffres après la virgule⟩e⟨±⟩⟨exposant⟩

Expression explicite des nombres entiers

⟨±⟩⟨chiffres⟩

Expression explicite des caractères

'⟨lettre⟩'

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

Exécution: `DEF`

Le code ASCII

bin.	d.	
1000001	65	A
1000010	66	B
1000011	67	C
1000100	68	D
⋮	⋮	
1011010	90	Z

bin.	d.	
1100001	97	a
1100010	98	b
1100011	99	c
1100100	100	d
⋮	⋮	
1111010	122	z

bin.	d.	
0110000	48	0
0110001	49	1
0110010	50	2
0110011	51	3
⋮	⋮	
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'`	*beep*	
`0001000`	8	`'\b'`	*backspace*	(retour arrière)
`0001001`	9	`'\t'`	*tab*	(tabulation)
`0001010`	10	`'\n'`	*line feed*	(nouvelle ligne)
`0001011`	11	`'\v'`	*vertical tab*	
`0001100`	12	`'\f'`	*form feed*	(nouvelle page)
`0001101`	13	`'\r'`	*carriage return*	(retour chariot)

Le code ASCII

Quelques caractères de contrôle utiles:

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

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

Exécution:

```
texteest un
```

Le code ASCII

Quelques caractères de contrôle utiles:

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

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

Exécution:

```
ceci est untexte
```

Le code ASCII

Quelques caractères de contrôle utiles:

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

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

Exécution:

```
ceci est un
texte
```

Le code ASCII

Quelques caractères de contrôle utiles:

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

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

Exécution:

```
ceci est un     texte
```

Le code ASCII

Quelques caractères de contrôle utiles:

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

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

Exécution:

```
ceci est
         un texte
```

Le code ASCII

Quelques caractères de contrôle utiles:

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

La déclaration des variables

```
double a;
```

$\longrightarrow$ déclaration d'une variable nommée "`a`"
elle est du type `double`

La déclaration des variables

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

$\longrightarrow$ déclaration d'une variable nommée "`n`"
elle est du type `int`

La déclaration des variables

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

$\longrightarrow$ 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;
```

$\longrightarrow$ 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;
```

$\longrightarrow$ `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;
```

$\longrightarrow$ 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;
```

$\longrightarrow$ 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 de 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:

$\longrightarrow$ définition des nombres comme $\pi = 3.14159\ldots$
$\longrightarrow$ définition des tailles des tableaux
$\longrightarrow$ définition des limites de précision
...

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

```
int n = 5;
```

$\longrightarrow$ la valeur 5 est copiée dans `n`

4.4 Les opérations arithmétiques élémentaires

= opérateur de copie

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

$\longrightarrow$ 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;
```

$\longrightarrow$ 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;
```

$\longrightarrow$ 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;
```

$\longrightarrow$ 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 !

⟨variable⟩ = ⟨expression⟩;

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;
```

$\longrightarrow$ 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;
```

$\longrightarrow$ 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;
```

$\longrightarrow$ 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 );
```

$\longrightarrow$ 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 );
```

$\longrightarrow$ 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	
$\star$	opérateur de multiplication	
$/$	opérateur de division	$\rightarrow$ 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` ?!?

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

$\frac{n}{p} = \frac{2}{3} = 0.666\ldots$ $\longleftrightarrow$ `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 $\rightarrow$ différent pour des entiers

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

Exécution:
`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 $\rightarrow$ 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 $\rightarrow$ 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 $\rightarrow$ 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 $\rightarrow$ 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)` :

`1/2*(a+b)`
$= 0 * 4.4 = 0$

4.4 Les opérations arithmétiques élémentaires

- $=$ opérateur de copie
- $+$ opérateur d'addition
- $-$ opérateur de soustraction
- $*$ opérateur de multiplication
- $/$ opérateur de division $\rightarrow$ 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 $\rightarrow$ 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 $\rightarrow$ 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
- $\star$ opérateur de multiplication
- $/$ opérateur de division $\rightarrow$ différent pour des entiers
- $\%$ opérateur "modulo" $\rightarrow$ seulement pour des entiers
  $\rightarrow$ 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

$\longrightarrow$ 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

$\longrightarrow$ 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

$\longrightarrow$ 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
```

$\longrightarrow$ 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
```

$\longrightarrow$ 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

$\longrightarrow$ la valeur de `n` subit une opération modulo par 3

Les opérations d'affectation

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

⟨variable⟩`+=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`+`⟨expression⟩`;`

⟨variable⟩`-=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`-`⟨expression⟩`;`

⟨variable⟩`*=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`*`⟨expression⟩`;`

⟨variable⟩`/=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`/`⟨expression⟩`;`

⟨variable⟩`%=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`%`⟨expression⟩`;`

Les opérations d'affectation

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

⟨variable⟩+=⟨expression⟩; ⇔ ⟨variable⟩=⟨variable⟩+⟨expression⟩;
⟨variable⟩−=⟨expression⟩; ⇔ ⟨variable⟩=⟨variable⟩−⟨expression⟩;
⟨variable⟩*=⟨expression⟩; ⇔ ⟨variable⟩=⟨variable⟩*⟨expression⟩;
⟨variable⟩/=⟨expression⟩; ⇔ ⟨variable⟩=⟨variable⟩/⟨expression⟩;
⟨variable⟩%=⟨expression⟩; ⇔ ⟨variable⟩=⟨variable⟩%⟨expression⟩;

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: `+=` `-=` `*=` `/=` `%=`

⟨variable⟩+=⟨expression⟩; ⇔ ⟨variable⟩=⟨variable⟩+⟨expression⟩;
⟨variable⟩−=⟨expression⟩; ⇔ ⟨variable⟩=⟨variable⟩−⟨expression⟩;
⟨variable⟩*=⟨expression⟩; ⇔ ⟨variable⟩=⟨variable⟩*⟨expression⟩;
⟨variable⟩/=⟨expression⟩; ⇔ ⟨variable⟩=⟨variable⟩/⟨expression⟩;
⟨variable⟩%=⟨expression⟩; ⇔ ⟨variable⟩=⟨variable⟩%⟨expression⟩;

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: `+=` `-=` `*=` `/=` `%=`

⟨variable⟩+=⟨expression⟩; ⇔ ⟨variable⟩=⟨variable⟩+⟨expression⟩;
⟨variable⟩−=⟨expression⟩; ⇔ ⟨variable⟩=⟨variable⟩−⟨expression⟩;
⟨variable⟩*=⟨expression⟩; ⇔ ⟨variable⟩=⟨variable⟩*⟨expression⟩;
⟨variable⟩/=⟨expression⟩; ⇔ ⟨variable⟩=⟨variable⟩/⟨expression⟩;
⟨variable⟩%=⟨expression⟩; ⇔ ⟨variable⟩=⟨variable⟩%⟨expression⟩;

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: `+=` `-=` `*=` `/=` `%=`

⟨variable⟩`+=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`+`⟨expression⟩`;`
⟨variable⟩`-=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`-`⟨expression⟩`;`
⟨variable⟩`*=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`*`⟨expression⟩`;`
⟨variable⟩`/=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`/`⟨expression⟩`;`
⟨variable⟩`%=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`%`⟨expression⟩`;`

Les opérateurs d'incrément: `++` `--`

⟨variable⟩`++;` ⇔ ⟨variable⟩`+=1;` ⇔ ⟨variable⟩`=`⟨variable⟩`+1;`
⟨variable⟩`--;` ⇔ ⟨variable⟩`-=1;` ⇔ ⟨variable⟩`=`⟨variable⟩`-1;`

Les opérations d'affectation

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

⟨variable⟩`+=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`+`⟨expression⟩`;`
⟨variable⟩`-=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`-`⟨expression⟩`;`
⟨variable⟩`*=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`*`⟨expression⟩`;`
⟨variable⟩`/=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`/`⟨expression⟩`;`
⟨variable⟩`%=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`%`⟨expression⟩`;`

Les opérateurs d'incrément: `++` `--`

⟨variable⟩`++;` ⇔ ⟨variable⟩`+=1;` ⇔ ⟨variable⟩`=`⟨variable⟩`+1;`
⟨variable⟩`--;` ⇔ ⟨variable⟩`-=1;` ⇔ ⟨variable⟩`=`⟨variable⟩`-1;`

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

Exécution: `4`

Les opérations d'affectation

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

⟨variable⟩`+=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`+`⟨expression⟩`;`
⟨variable⟩`-=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`-`⟨expression⟩`;`
⟨variable⟩`*=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`*`⟨expression⟩`;`
⟨variable⟩`/=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`/`⟨expression⟩`;`
⟨variable⟩`%=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`%`⟨expression⟩`;`

Les opérateurs d'incrément: `++` `--`

`++`⟨variable⟩`;` ⇔ ⟨variable⟩`+=1;` ⇔ ⟨variable⟩`=`⟨variable⟩`+1;`
`--`⟨variable⟩`;` ⇔ ⟨variable⟩`-=1;` ⇔ ⟨variable⟩`=`⟨variable⟩`-1;`

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

Exécution: `4`

Les opérations d'affectation

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

⟨variable⟩`+=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`+`⟨expression⟩`;`
⟨variable⟩`-=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`-`⟨expression⟩`;`
⟨variable⟩`*=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`*`⟨expression⟩`;`
⟨variable⟩`/=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`/`⟨expression⟩`;`
⟨variable⟩`%=`⟨expression⟩`;` ⇔ ⟨variable⟩`=`⟨variable⟩`%`⟨expression⟩`;`

Les opérateurs d'incrément: `++` `--`

⟨variable⟩`++;` ⇔ ⟨variable⟩`+=1;` ⇔ ⟨variable⟩`=`⟨variable⟩`+1;`
⟨variable⟩`--;` ⇔ ⟨variable⟩`-=1;` ⇔ ⟨variable⟩`=`⟨variable⟩`-1;`

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