6 Les fonctions

Peter Schlagheck

Université de Liège

6 Les fonctions

6.1 Un exemple

On a besoin d'effectuer une certaine action plusieurs fois dans des contextes différents:

```
include <iostream>
using namespace std;

int main()
{
    ...
    ⟨action⟩;
    ...
    if ( ... )
    {
        ⟨action⟩;
        ...
    }
    ⟨action⟩;
    ...
}
```

où ⟨action⟩ effectue p. ex.

```
⟨instruction1⟩;
⟨instruction2⟩;
while ( ⟨expression⟩ )
{
    ⟨instruction3⟩;
    ⟨instruction4⟩;
}
⟨instruction5⟩;
```

...chaque fois avec d'autres variables

6.1 Un exemple d'une fonction

Programme pour calculer des nombres entiers k, l, m qui satisfont à $k^n + l^n = m^n$ pour un exposant $n > 0$ donné

$\longrightarrow$ problème de Fermat

Calcul de la puissance n d'un nombre entier:

```
int k, n;
cout << "entrez les deux entiers k et n:";
cin >> k >> n;
int kn = 1;
for ( int i = 1; i <= n; i++ )
  kn *= k;
cout << k << "^" << n << " = " << kn << endl;
```

6.1 Un exemple d'une fonction

Programme pour calculer des nombres entiers k, l, m qui satisfont à $k^n + l^n = m^n$ pour un exposant $n > 0$ donné

$\longrightarrow$ problème de Fermat

Calcul de la puissance n d'un nombre entier:

```
int k, n;
cout << "entrez les deux entiers k et n:";
cin >> k >> n;
int kn = 1;
for ( int i = 1; i <= n; i++ )
  kn *= k;
cout << k << "^" << n << " = " << kn << endl;
```

Exécution:

```
entrez deux entiers k et n: 3
4
3^4 = 81
```

6.1 Un exemple d'une fonction

Programme pour calculer des nombres entiers k, l, m qui satisfont à $k^n + l^n = m^n$ pour un exposant $n > 0$ donné

```
int n, Nmax;
cout << "exposant:";
cin >> n;
cout << "nombre maximal:";
cin >> Nmax;
for ( int k = 1; k <= Nmax; k++ )
for ( int l = 1; l <= Nmax; l++ )
for ( int m = 1; m <= Nmax; m++ )
{
  // calcul de k^n
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  // calcul de l^n
  int ln = 1;
  for ( int i = 1; i <= n; i++ )
    ln *= l;
  // calcul de m^n
  int mn = 1;
  for ( int i = 1; i <= n; i++ )
    mn *= m;
  if ( kn + ln == mn )
    cout << k << "^" << n << " + "
         << l << "^" << n << " = "
         << m << "^" << n << endl;
}
```

6.1 Un exemple d'une fonction

Programme pour calculer des nombres entiers k, l, m qui satisfont à $k^n + l^n = m^n$ pour un exposant $n > 0$ donné

```
int n, Nmax;
cout << "exposant:";
cin >> n;
cout << "nombre maximal:";
cin >> Nmax;
for ( int k = 1; k <= Nmax; k++ )
for ( int l = 1; l <= Nmax; l++ )
for ( int m = 1; m <= Nmax; m++ )
{
  // calcul de k^n
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  // calcul de l^n
  int ln = 1;
  for ( int i = 1; i <= n; i++ )
    ln *= l;
  // calcul de m^n
  int mn = 1;
  for ( int i = 1; i <= n; i++ )
    mn *= m;
  if ( kn + ln == mn )
    cout << k << "^" << n << " + "
         << l << "^" << n << " = "
         << m << "^" << n << endl;
}
```

Exécution:

```
exposant: 2
nombre maximal: 10
3^2 + 4^2 = 5^2
4^2 + 3^2 = 5^2
6^2 + 8^2 = 10^2
8^2 + 6^2 = 10^2
```

6.1 Un exemple d'une fonction

Programme pour calculer des nombres entiers k, l, m qui satisfont à $k^n + l^n = m^n$ pour un exposant $n > 0$ donné

```
int n, Nmax;
cout << "exposant:";
cin >> n;
cout << "nombre maximal:";
cin >> Nmax;
for ( int k = 1; k <= Nmax; k++ )
for ( int l = 1; l <= Nmax; l++ )
for ( int m = 1; m <= Nmax; m++ )
{
  // calcul de k^n
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  // calcul de l^n
  int ln = 1;
  for ( int i = 1; i <= n; i++ )
    ln *= l;
  // calcul de m^n
  int mn = 1;
  for ( int i = 1; i <= n; i++ )
    mn *= m;
  if ( kn + ln == mn )
    cout << k << "^" << n << " + "
         << l << "^" << n << " = "
         << m << "^" << n << endl;
}
```

Exécution:

```
exposant: 3
nombre maximal: 10
```

6.1 Un exemple d'une fonction

Programme pour calculer des nombres entiers k, l, m qui satisfont à $k^n + l^n = m^n$ pour un exposant $n > 0$ donné

```
int n, Nmax;
cout << "exposant:";
cin >> n;
cout << "nombre maximal:";
cin >> Nmax;
for ( int k = 1; k <= Nmax; k++ )
for ( int l = 1; l <= Nmax; l++ )
for ( int m = 1; m <= Nmax; m++ )
{
  // calcul de k^n
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  // calcul de l^n
  int ln = 1;
  for ( int i = 1; i <= n; i++ )
    ln *= l;
  // calcul de m^n
  int mn = 1;
  for ( int i = 1; i <= n; i++ )
    mn *= m;
  if ( kn + ln == mn )
    cout << k << "^" << n << " + "
         << l << "^" << n << " = "
         << m << "^" << n << endl;
}
```

Exécution:

```
exposant: 3
nombre maximal: 100
```

6.1 Un exemple d'une fonction

Programme pour calculer des nombres entiers k, l, m qui satisfont à $k^n + l^n = m^n$ pour un exposant $n > 0$ donné

```
int n, Nmax;
cout << "exposant:";
cin >> n;
cout << "nombre maximal:";
cin >> Nmax;
for ( int k = 1; k <= Nmax; k++ )
for ( int l = 1; l <= Nmax; l++ )
for ( int m = 1; m <= Nmax; m++ )
{
  // calcul de k^n
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  // calcul de l^n
  int ln = 1;
  for ( int i = 1; i <= n; i++ )
    ln *= l;
  // calcul de m^n
  int mn = 1;
  for ( int i = 1; i <= n; i++ )
    mn *= m;
  if ( kn + ln == mn )
    cout << k << "^" << n << " + "
         << l << "^" << n << " = "
         << m << "^" << n << endl;
}
```

Problème:

on doit écrire trois fois la même itération

Solution:

copier & coller dans l'éditeur...
ou la définition d'une fonction

6.1 Un exemple d'une fonction

Programme pour calculer des nombres entiers k, l, m qui satisfont à $k^n + l^n = m^n$ pour un exposant $n > 0$ donné

```
int n, Nmax;
cout << "exposant:";
cin >> n;
cout << "nombre maximal:";
cin >> Nmax;
for ( int k = 1; k <= Nmax; k++ )
for ( int l = 1; l <= Nmax; l++ )
for ( int m = 1; m <= Nmax; m++ )
  if ( puissance( k, n ) +
       puissance( l, n ) ==
       puissance( m, n ) )
    cout << k << "^" << n << " + "
         << l << "^" << n << " = "
         << m << "^" << n << endl;
```

Problème:

on doit écrire trois fois la même itération

Solution:

copier & coller dans l'éditeur...
ou la définition d'une fonction
`puissance( k, n )`

6.1 Un exemple d'une fonction

```
int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}
```

6.1 Un exemple d'une fonction

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int n, Nmax;
  cout << "exposant:";
  cin >> n;
  cout << "nombre maximal:";
  cin >> Nmax;
  for ( int k = 1; k <= Nmax; k++ )
  for ( int l = 1; l <= Nmax; l++ )
  for ( int m = 1; m <= Nmax; m++ )
    if ( puissance( k, n ) +
         puissance( l, n ) ==
         puissance( m, n ) )
      cout << k << "^" << n << " + "
           << l << "^" << n << " = "
           << m << "^" << n << endl;
}
```

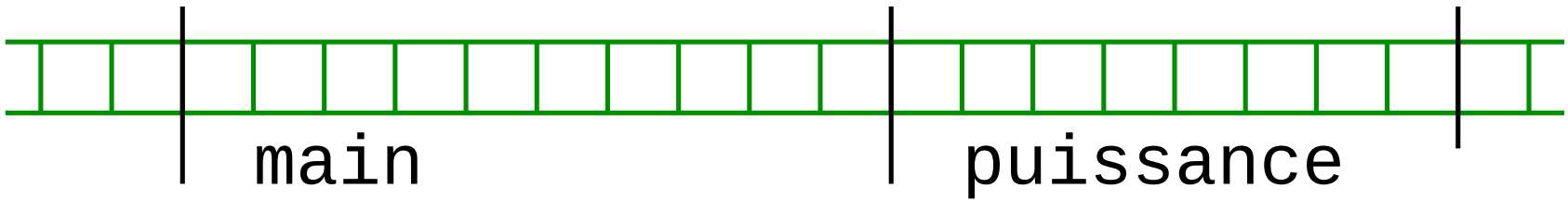
6.1 Un exemple d'une fonction

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Le flot du programme dans la mémoire:



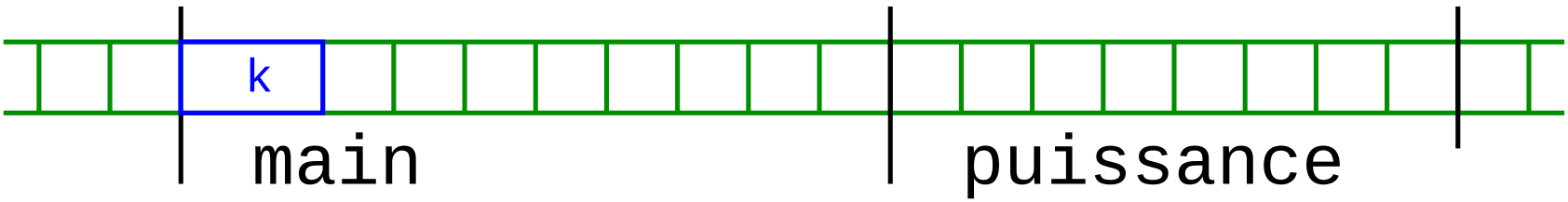
6.1 Un exemple d'une fonction

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Le flot du programme dans la mémoire:



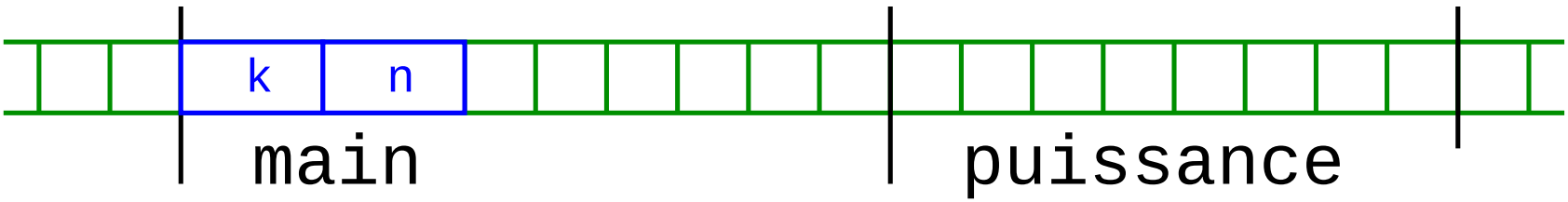
6.1 Un exemple d'une fonction

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Le flot du programme dans la mémoire:



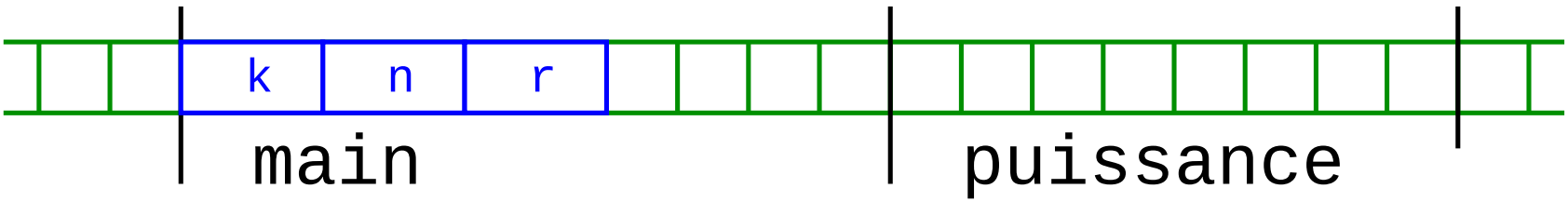
6.1 Un exemple d'une fonction

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Le flot du programme dans la mémoire:



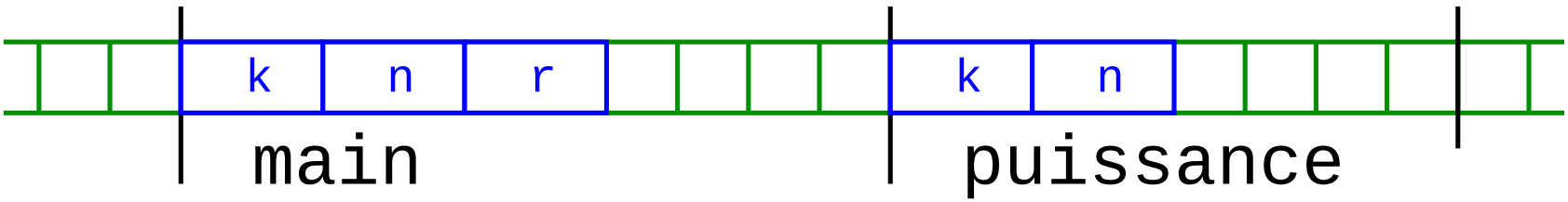
6.1 Un exemple d'une fonction

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Le flot du programme dans la mémoire:



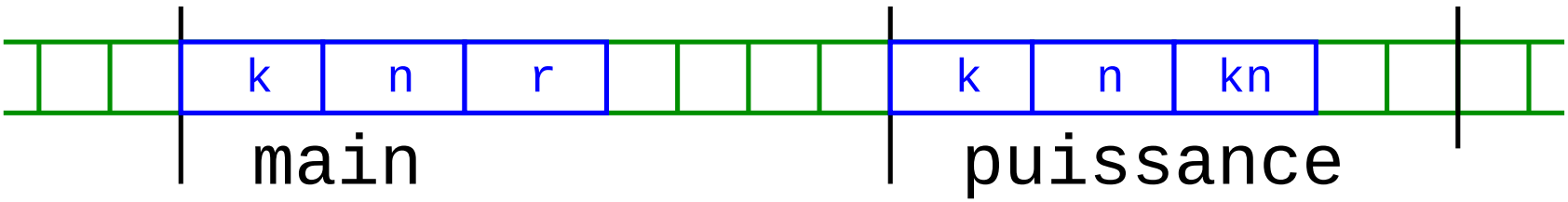
6.1 Un exemple d'une fonction

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Le flot du programme dans la mémoire:



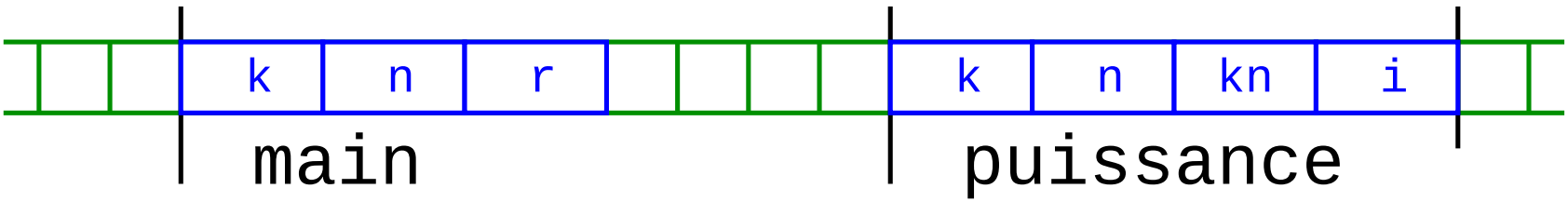
6.1 Un exemple d'une fonction

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Le flot du programme dans la mémoire:



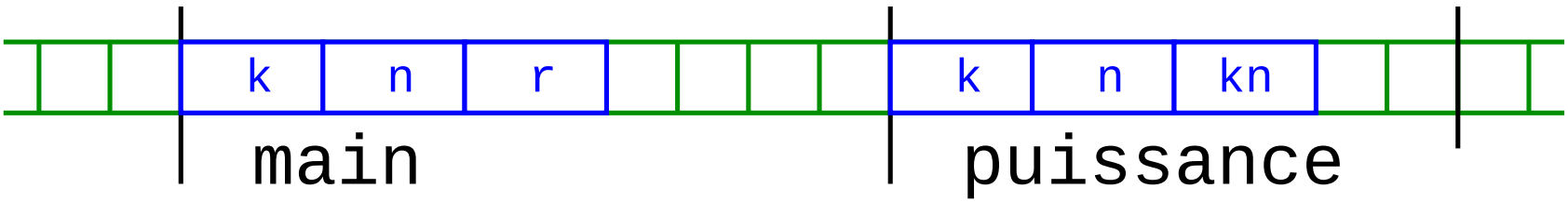
6.1 Un exemple d'une fonction

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Le flot du programme dans la mémoire:



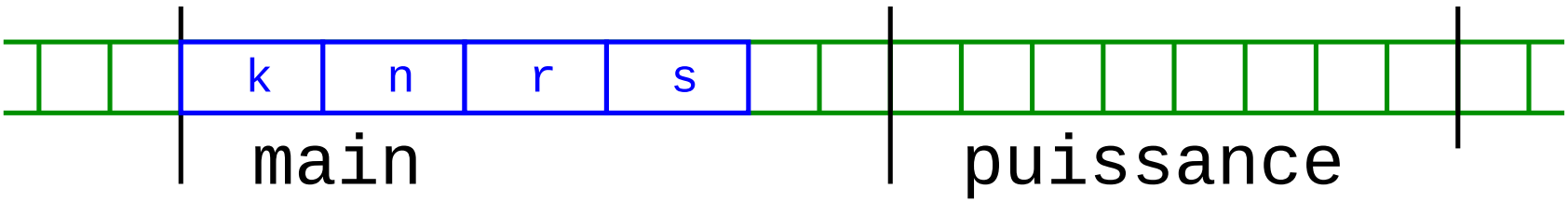
6.1 Un exemple d'une fonction

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Le flot du programme dans la mémoire:



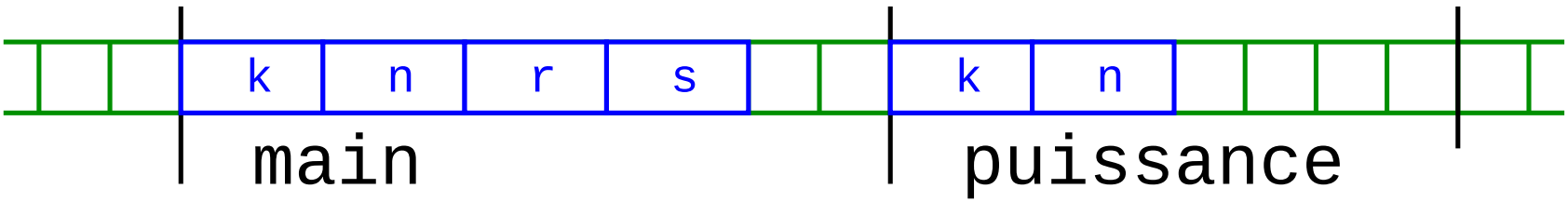
6.1 Un exemple d'une fonction

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Le flot du programme dans la mémoire:



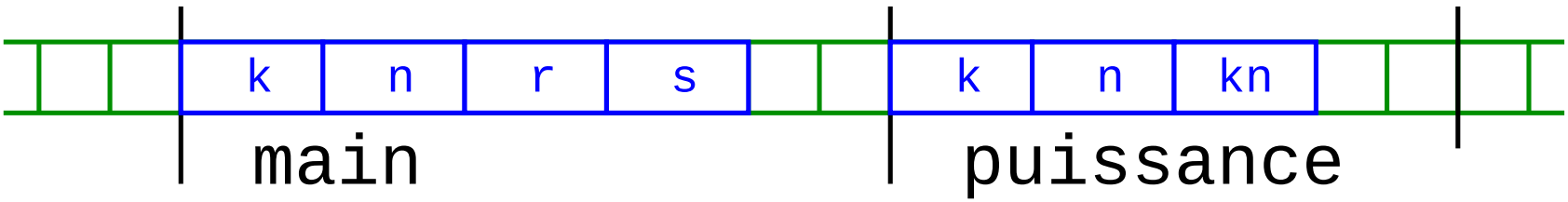
6.1 Un exemple d'une fonction

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Le flot du programme dans la mémoire:



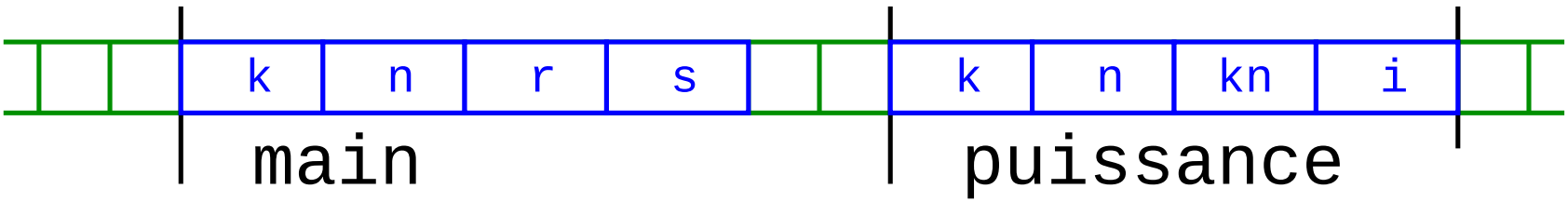
6.1 Un exemple d'une fonction

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Le flot du programme dans la mémoire:



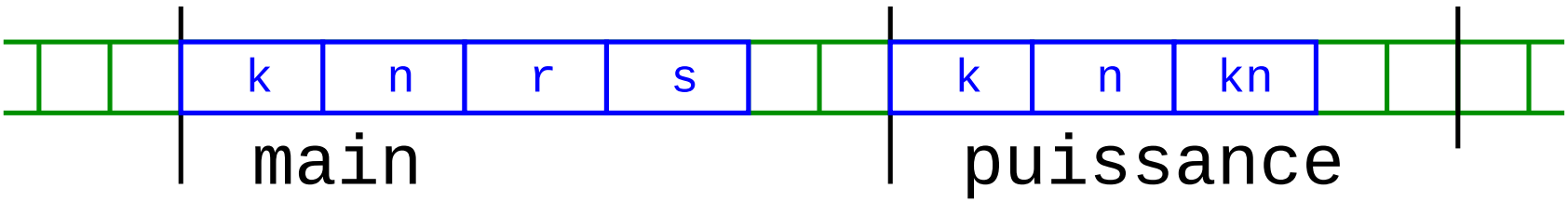
6.1 Un exemple d'une fonction

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Le flot du programme dans la mémoire:



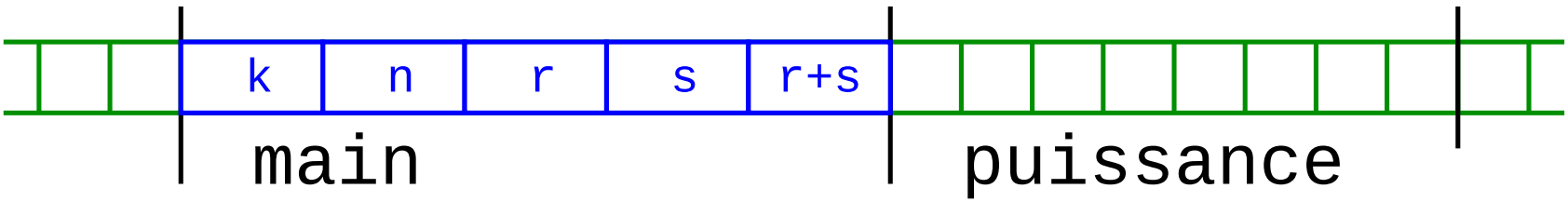
6.1 Un exemple d'une fonction

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Le flot du programme dans la mémoire:



Exécution de la fonction `puissance`

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Exécution:

```
17
```

Exécution de la fonction `puissance`

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( 2, 3 );
  int s = puissance( 3, 2 );
  cout << r + s << endl;
}
```

Exécution:

```
17
```

Exécution de la fonction `puissance`

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n + k, k );
  cout << r + s << endl;
}
```

Exécution:

```
33
```

Exécution de la fonction `puissance`

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  n = 6;
  k = 4;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
  cout << n << " " << k << endl;
}
```

Exécution:

```
17
3 2
```

Exécution de la fonction `puissance`

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

int puissance( int k, int n )
{
  int kn = m;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int m = 1;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

⟶ `erreur: 'm' was not declared in this scope`

Exécution de la fonction `puissance`

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
  kn = 0;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Exécution:

```
17
```

Exécution de la fonction `puissance`

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return 0;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Exécution:

0

6.2 La structure générale d'une fonction

La définition d'une fonction:

⟨type⟩ ⟨nom⟩ (⟨type1⟩ ⟨nom1⟩, ⟨type2⟩ ⟨nom2⟩, ...)
{
    ⟨instruction1⟩;
    ⟨instruction2⟩;
    ...
    return ⟨expression_retour⟩;
}

⟶ ⟨expression_retour⟩ doit être du type ⟨type⟩
(ou convertible dans ce type-là)

6.2 La structure générale d'une fonction

La définition d'une fonction:

⟨type⟩ ⟨nom⟩ (⟨type1⟩ ⟨nom1⟩, ⟨type2⟩ ⟨nom2⟩, ...)
{
    ⟨instruction1⟩;
    ⟨instruction2⟩;
    ...
    return ⟨expression_retour⟩;
}

Appel de la fonction:

⟨nom⟩ (⟨expression1⟩, ⟨expression2⟩, ...)
    rend ⟨expression_retour⟩ comme valeur

⟶ ⟨expression1⟩ doit être du type ⟨type1⟩,
    ⟨expression2⟩ doit être du type ⟨type2⟩, ...
    (conversions implicites possibles)

6.2 La structure générale d'une fonction

L'appel de cette fonction corresponderait au code

⟨type⟩ ⟨nom⟩;
{
    ⟨type1⟩ ⟨nom1⟩ = ⟨expression1⟩;
    ⟨type2⟩ ⟨nom2⟩ = ⟨expression2⟩;
    ...
    ⟨instruction1⟩;
    ⟨instruction2⟩;
    ...
    ⟨nom⟩ = ⟨expression_retour⟩;
}

dans la partie du programme où cette fonction est appelée

(à l'exception qu'on ne pourrait pas utiliser d'autres variables du programme principal dans ce bloc-là)

6.2 La structure générale d'une fonction

La définition d'une fonction:

⟨type⟩ ⟨nom⟩ (⟨type1⟩ ⟨nom1⟩, ⟨type2⟩ ⟨nom2⟩, ...)
{
    ⟨instruction1⟩;
    ⟨instruction2⟩;
    ...
    return ⟨expression_retour⟩;
}

Les variables ⟨nom1⟩, ⟨nom2⟩, ... sont implicitement définies au début du bloc de la fonction (définition locale)

6.2 La structure générale d'une fonction

La définition d'une fonction:

⟨type⟩ ⟨nom⟩ (⟨type1⟩ ⟨nom1⟩, ⟨type2⟩ ⟨nom2⟩, ...)
{
    ⟨instruction1⟩;
    ⟨instruction2⟩;
    ...
    return ⟨expression_retour⟩;
}

Appel de la fonction:

⟨nom⟩ (⟨expression1⟩, ⟨expression2⟩, ...)
    rend ⟨expression_retour⟩ comme valeur

⟶ ⟨expression1⟩, ⟨expression2⟩, ... sont donc données à la fonction comme des copies

6.2 La structure générale d'une fonction

La définition d'une fonction:

```
⟨type⟩ ⟨nom⟩ ( ⟨type1⟩ ⟨nom1⟩, ⟨type2⟩ ⟨nom2⟩, ... )
{
    ⟨instruction1⟩;
    ⟨instruction2⟩;
    ...
    return ⟨expression_retour⟩;
}
```

l'instruction

```
return ⟨expression_retour⟩;
```

termine l'exécution de la séquence d'instructions de la fonction et rend la valeur de l'expression ⟨expression_retour⟩

$\longrightarrow$ il peut y avoir plusieurs instructions `return` dans une fonction

6.2 La structure générale d'une fonction

La définition d'une fonction:

⟨type⟩ ⟨nom⟩ (⟨type1⟩ ⟨nom1⟩, ⟨type2⟩ ⟨nom2⟩, ...)
{
    ⟨instruction1⟩;
    ⟨instruction2⟩;
    ...
    return ⟨expression_retour⟩;
    ...
    return ⟨expression_retour2⟩;
    ...
}

Exécution de la fonction `puissance`

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = -2;
  int n = -3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Exécution:

2

Exécution de la fonction `puissance`

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

int puissance( int k, int n )
{
  if ( n < 0 )
  {
    cout << "exposant negatif!" << endl;
    return 0;
  }
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = -2;
  int n = -3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Exécution:

```
exposant negatif!
exposant negatif!
0
```

Exécution de la fonction `puissance`

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

int puissance( int k, int n )
{
  if ( n < 0 )
  {
    cerr << "exposant negatif!" << endl;
    return 0;
  }
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

int main()
{
  int k = -2;
  int n = -3;
  int r = puissance( k, n );
  int s = puissance( n, k );
  cout << r + s << endl;
}
```

Exécution:

```
exposant negatif!
exposant negatif!
0
```

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

→ dans une fonction, vous pouvez faire tout ce qui est aussi possible dans `main()`

⟶ définir des variables,
placer des branchement et des boucles,
...

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

- → dans une fonction, vous pouvez faire tout ce qui est aussi possible dans `main()`
- → vos fonctions peuvent rendre n'importe quel type

  ($\longrightarrow$ des entiers, des flottants, des caractères, ...)

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

- → dans une fonction, vous pouvez faire tout ce qui est aussi possible dans `main()`
- → vos fonctions peuvent rendre n'importe quel type
- → elles peuvent contenir autant d'arguments que vous voulez

  ⟶ même pas d'arguments de tout !

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

- → dans une fonction, vous pouvez faire tout ce qui est aussi possible dans `main()`
- → vos fonctions peuvent rendre n'importe quel type
- → elles peuvent contenir autant d'arguments que vous voulez

```
int maxint()
{
  int i = 1;
  while ( i > 0 )
    i++;
  i--;
  return i;
}
```

Appel:

```
cout << maxint() << endl;
```

```
2147483647
```

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

- → dans une fonction, vous pouvez faire tout ce qui est aussi possible dans `main()`
- → vos fonctions peuvent rendre n'importe quel type
- → elles peuvent contenir autant d'arguments que vous voulez
- → vous pouvez choisir n'importe quels noms pour vos fonctions . . .

  . . . tant qu'il n'y ait pas d'ambiguité avec d'autres fonctions

L'identification des fonctions

Au niveau du compilateur, une fonction s'identifie non seulement par son nom, mais aussi par sa liste d'arguments.

$\longrightarrow$ le "nom interne" de la fonction

```
int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}
```

au niveau du compilateur sera donc `puissance( int, int )`

$\longrightarrow$ vous pouvez définir d'autres fonctions nommées `puissance( ... )` pourvu que leurs listes d'arguments soient différentes

Deux fonctions avec le même nom

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

double puissance( double k, int n )
{
  double kn = 1;
  if ( n < 0 )
    for ( int i = -1; i >= n; i-- )
      kn /= k;
  else
    for ( int i = 1; i <= n; i++ )
      kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  cout << r << endl;
}
```

$\longrightarrow$ définition des fonctions

`puissance( int, int )`

et

`puissance( double, int )`

Deux fonctions avec le même nom

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

double puissance( double k, int n )
{
  double kn = 1;
  if ( n < 0 )
    for ( int i = -1; i >= n; i-- )
      kn /= k;
  else
    for ( int i = 1; i <= n; i++ )
      kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = 3;
  int r = puissance( k, n );
  cout << r << endl;
}
```

Exécution:

8

$\longrightarrow$ appel de la fonction

`puissance( int, int )`

Deux fonctions avec le même nom

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

double puissance( double k, int n )
{
  double kn = 1;
  if ( n < 0 )
    for ( int i = -1; i >= n; i-- )
      kn /= k;
  else
    for ( int i = 1; i <= n; i++ )
      kn *= k;
  return kn;
}

int main()
{
  double k = 2;
  int n = 3;
  double r = puissance( k, n );
  cout << r << endl;
}
```

Exécution:

8

$\longrightarrow$ appel de la fonction

`puissance( double, int )`

Deux fonctions avec le même nom

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

double puissance( double k, int n )
{
  double kn = 1;
  if ( n < 0 )
    for ( int i = -1; i >= n; i-- )
      kn /= k;
  else
    for ( int i = 1; i <= n; i++ )
      kn *= k;
  return kn;
}

int main()
{
  double k = 2;
  int n = -3;
  double r = puissance( k, n );
  cout << r << endl;
}
```

Exécution:

```
0.125
```

$\longrightarrow$ appel de la fonction

```
puissance( double, int )
```

Deux fonctions avec le même nom

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

double puissance( double k, int n )
{
  double kn = 1;
  if ( n < 0 )
    for ( int i = -1; i >= n; i-- )
      kn /= k;
  else
    for ( int i = 1; i <= n; i++ )
      kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = -3;
  double r = puissance( k, n );
  cout << r << endl;
}
```

Exécution:

1

$\longrightarrow$ appel de la fonction `puissance( int, int )`

$\longrightarrow$ pas de conversion implicite

Deux fonctions avec le même nom

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

double puissance( double k, int n )
{
  double kn = 1;
  if ( n < 0 )
    for ( int i = -1; i >= n; i-- )
      kn /= k;
  else
    for ( int i = 1; i <= n; i++ )
      kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = -3;
  double r = puissance( (double) k, n );
  cout << r << endl;
}
```

Exécution:

```
0.125
```

$\longrightarrow$ appel de la fonction

```
puissance( double, int )
```

$\longrightarrow$ conversion explicite

Deux fonctions avec le même nom

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

int puissance( int k, int n )
{
  int kn = 1;
  for ( int i = 1; i <= n; i++ )
    kn *= k;
  return kn;
}

double puissance( int k, int n )
{
  double kn = 1;
  if ( n < 0 )
    for ( int i = -1; i >= n; i-- )
      kn /= k;
  else
    for ( int i = 1; i <= n; i++ )
      kn *= k;
  return kn;
}

int main()
{
  int k = 2;
  int n = -3;
  double r = puissance( k, n );
  cout << r << endl;
}
```

$\longrightarrow$ erreur: new declaration 'double puissance(int,int)'

ambiguates old declaration 'int puissance(int,int)'

6.3 Les routines

Des routines sont des fonctions qui ne rendent pas de valeur

et qui sont appelées comme des instructions
(et non pas comme des expressions)

Structure générale:

```
void ⟨nom⟩ ( ⟨type1⟩ ⟨nom1⟩, ⟨type2⟩ ⟨nom2⟩, ... )
{
    ⟨instruction1⟩;
    ⟨instruction2⟩;
    ...
    return;
}
```

Appel de la routine:

```
⟨nom⟩ ( ⟨expression1⟩, ⟨expression2⟩, ... );
```

6.3 Les routines

Des routines sont des fonctions qui ne rendent pas de valeur

et qui sont appelées comme des instructions
(et non pas comme des expressions)

Structure générale:

```
void ⟨nom⟩ ( ⟨type1⟩ ⟨nom1⟩, ⟨type2⟩ ⟨nom2⟩, ... )
{
    ⟨instruction1⟩;
    ⟨instruction2⟩;
    ...
    return;
}
```

$\longrightarrow$ `void` signifie un type “vide”

$\longrightarrow$ pas de valeur de retour
l’instruction `return;` n’est même pas nécessaire

6.3 Les routines

Des routines sont des fonctions qui ne rendent pas de valeur

et qui sont appelées comme des instructions
(et non pas comme des expressions)

Structure générale:

```
void ⟨nom⟩ ( ⟨type1⟩ ⟨nom1⟩, ⟨type2⟩ ⟨nom2⟩, ... )
{
    ⟨instruction1⟩;
    ⟨instruction2⟩;
    ...
}
```

6.3 Les routines

Exemple: input des chiffres entre 0 et 9

```
int main()
{
  int k, l, m;
  cout << "tapez trois chiffres:";
  cin >> k >> l >> m;
  check_number( k );
  check_number( l );
  check_number( m );
  ...
}
```

$\longrightarrow$ message d'alerte dans `check_number`
si `k`, `l` ou `m` ne sont pas contenus entre 0 et 9

6.3 Les routines

```
void check_number( int n )
{
  if ( n < 0 )
    cout << "warning: " << n
         << " ne doit pas etre negatif" << endl;
  if ( n > 9 )
    cout << "warning: " << n
         << " doit etre plus petit que 10" << endl;
}

int main()
{
  int k, l, m;
  cout << "tapez trois chiffres:";
  cin >> k >> l >> m;
  check_number( k );
  check_number( l );
  check_number( m );
  ...
}
```

6.3 Les routines

Manipulation des variables dans des routines
(ou dans des fonctions):

$\longrightarrow$ utilisation de l'opérateur `&`
dans la définition de la routine (ou de la fonction)

⟨type⟩ ⟨nom⟩ (⟨type1⟩ ⟨nom1⟩,
⟨type2⟩ &⟨nom2⟩,
⟨type3⟩ ⟨nom3⟩, ...)
{
...
}

$\longrightarrow$ la fonction ⟨nom⟩ travaille avec l'original ,
et non pas avec la copie, du deuxième argument

6.3 Les routines

Manipulation des variables dans des routines
(ou dans des fonctions):

$\longrightarrow$ utilisation de l'opérateur `&`
dans la définition de la routine (ou de la fonction)

⟨type⟩ ⟨nom⟩ (⟨type1⟩ ⟨nom1⟩,
⟨type2⟩ &⟨nom2⟩,
⟨type3⟩ ⟨nom3⟩, ...)
{
...
}

Appel:

⟨nom⟩ (⟨expression1⟩, ⟨variable2⟩, ⟨expression3⟩, ...)

$\longrightarrow$ ⟨variable2⟩ peut être manipulée par la fonction ⟨nom⟩

Une routine d'échange de deux variables

```
int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  // 'echange' doit echanger a et b
  cout << a << " > " << b << endl;
}
```

Une routine d'échange de deux variables

```
void echange( int &a, int &b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres: 5
3
5 > 3
```

Une routine d'échange de deux variables

```
void echange( int &a, int &b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres: 2
6
6 > 2
```

Une routine d'échange de deux variables

```
void echange( int a, int b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

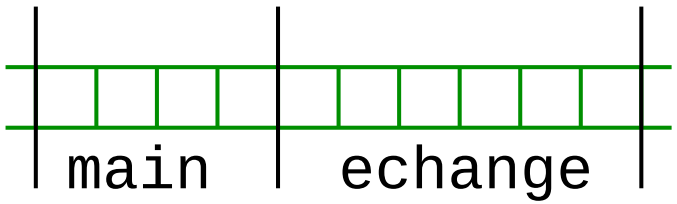
```
2 nombres: 2
6
2 > 6
```

Une routine d'échange de deux variables

```
void echange( int a, int b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres:
```

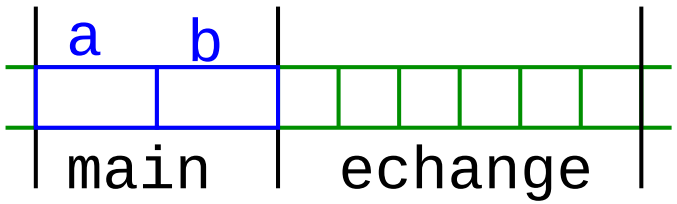


Une routine d'échange de deux variables

```
void echange( int a, int b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres:
```

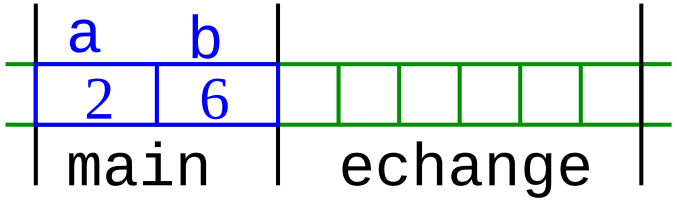


Une routine d'échange de deux variables

```
void echange( int a, int b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres: 2
6
```

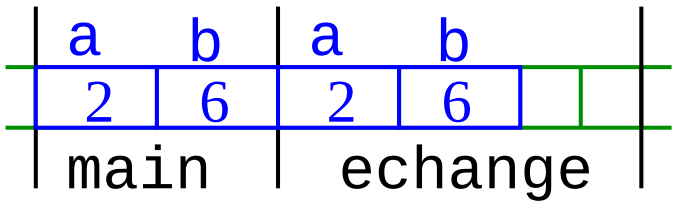


Une routine d'échange de deux variables

```
void echange( int a, int b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres: 2
6
```

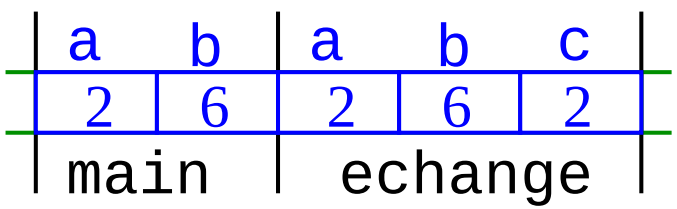


Une routine d'échange de deux variables

```
void echange( int a, int b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres: 2
6
```

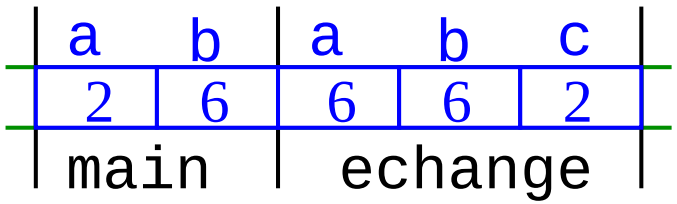


Une routine d'échange de deux variables

```
void echange( int a, int b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres: 2
6
```

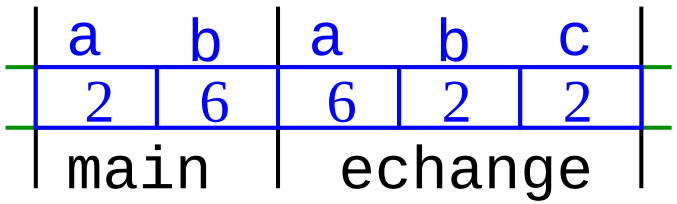


Une routine d'échange de deux variables

```
void echange( int a, int b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres: 2
6
```

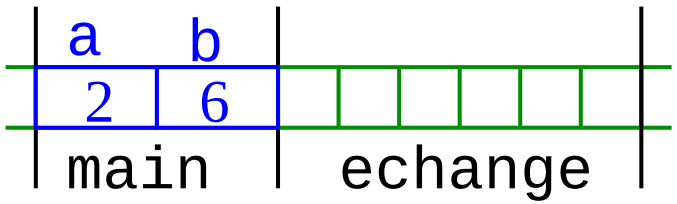


Une routine d'échange de deux variables

```
void echange( int a, int b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres: 2
6
2 > 6
```

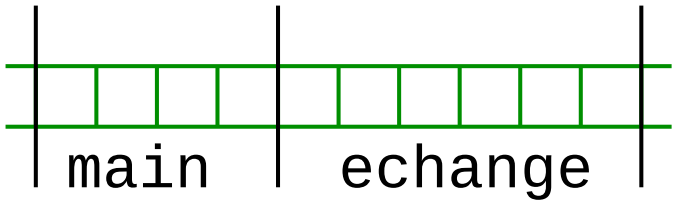


Une routine d'échange de deux variables

```
void echange( int &a, int &b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres:
```

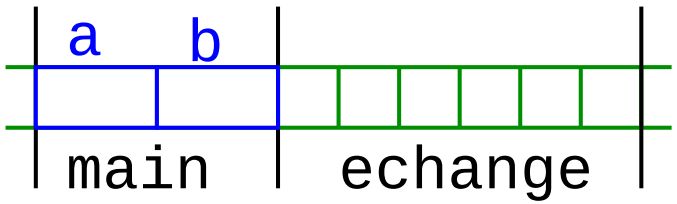


Une routine d'échange de deux variables

```
void echange( int &a, int &b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres:
```

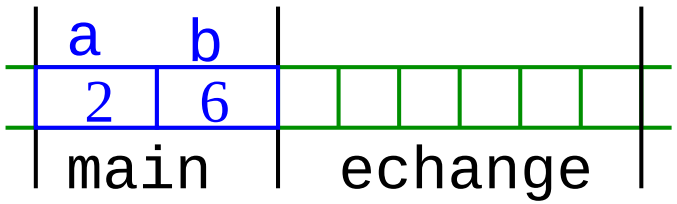


Une routine d'échange de deux variables

```
void echange( int &a, int &b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres: 2
6
```

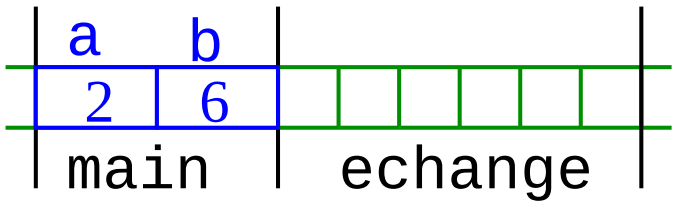


Une routine d'échange de deux variables

```
void echange( int &a, int &b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres: 2
6
```

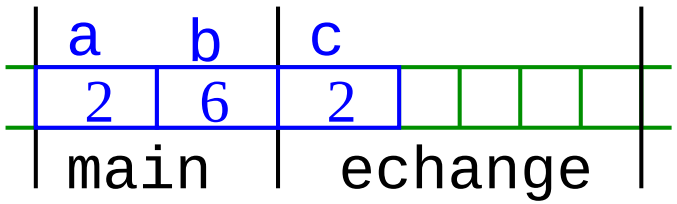


Une routine d'échange de deux variables

```
void echange( int &a, int &b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres: 2
6
```

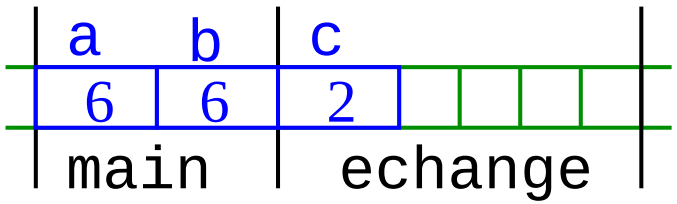


Une routine d'échange de deux variables

```
void echange( int &a, int &b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres: 2
6
```

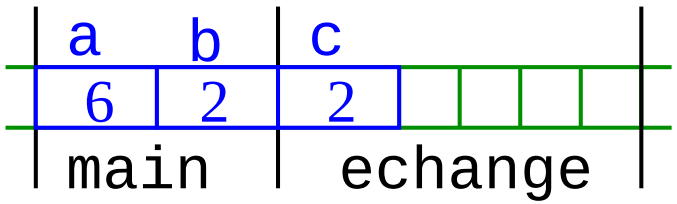


Une routine d'échange de deux variables

```
void echange( int &a, int &b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres: 2
6
```

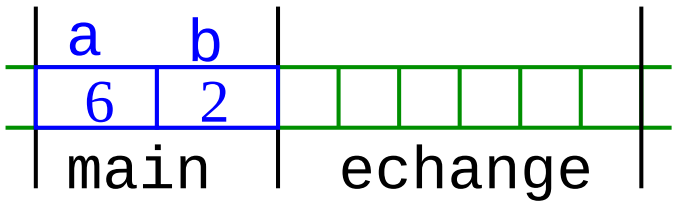


Une routine d'échange de deux variables

```
void echange( int &a, int &b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a, b );
  cout << a << " > " << b << endl;
}
```

```
2 nombres: 2
6
6 > 2
```



Une routine d'échange de deux variables

```
void echange( int &a, int &b )
{
  int c = a;
  a = b;
  b = c;
}

int main()
{
  cout << "2 nombres:";
  int a, b;
  cin >> a >> b;
  if ( a < b )
    echange( a + b, b );
  cout << a << " > " << b << endl;
}
```

⟶ erreur:

```
no matching function for call to 'echange(int, int&)'
```

Une routine de tri pour trois variables

```
void echange( int &a, int &b )
{
  int c = a;
  a = b;
  b = c;
}

void sort( int &a, int &b )
{
  if ( a < b )
    echange( a, b );
}

void sort( int &a, int &b, int &c )
{
  sort( a, b );
  sort( a, c );
  sort( b, c );
}

int main()
{
  cout << "3 nombres:" << endl;
  int k, l, m;
  cin >> k >> l >> m;
  sort( k, l, m );
  cout << k << " > " << l << " > " << m << endl;
}
```

```
3 nombres: 2
5
3
5 > 3 > 2
```

Une routine de tri pour trois variables

```
void sort( int &a, int &b )
{
  if ( a < b )
    echange( a, b );
}

void echange( int &a, int &b )
{
  int c = a;
  a = b;
  b = c;
}

void sort( int &a, int &b, int &c )
{
  sort( a, b );
  sort( a, c );
  sort( b, c );
}

int main()
{
  cout << "3 nombres:" << endl;
  int k, l, m;
  cin >> k >> l >> m;
  sort( k, l, m );
  cout << k << " > " << l << " > " << m << endl;
}
```

$\longrightarrow$ erreur: 'echange' was not declared in this scope

6.4 La récursion

Nous venons d'apprendre qu'une fonction/routine peut appeler d'autres fonctions/routines en cours de son exécution . . .

. . . mais elle peut également appeler elle-même

6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

```
int factoriel( int n )
{
  int nfac = 1;
  for ( int i = 1; i <= n; i++ )
    nfac *= i;
  return nfac;
}
```

6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  return ( n * factoriel( n - 1 ) );
}
```

6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  return ( n * factoriel( n - 1 ) );
}

int main()
{
  int k = 6;
  cout << factoriel( k ) << endl;
}
```

Exécution:

```
Segmentation fault
```

6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  if ( n == 0 )
    return 1;
  return ( n * factoriel( n - 1 ) );
}

int main()
{
  int k = 6;
  cout << factoriel( k ) << endl;
}
```

Exécution:

```
720
```

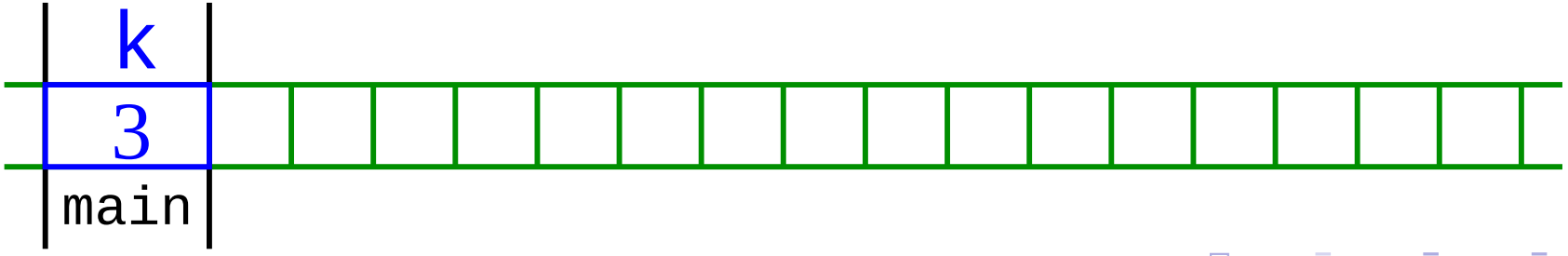
6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  if ( n == 0 )
    return 1;
  return ( n * factoriel( n - 1 ) );
}

int main()
{
  int k = 3;
  cout << factoriel( k ) << endl;
}
```



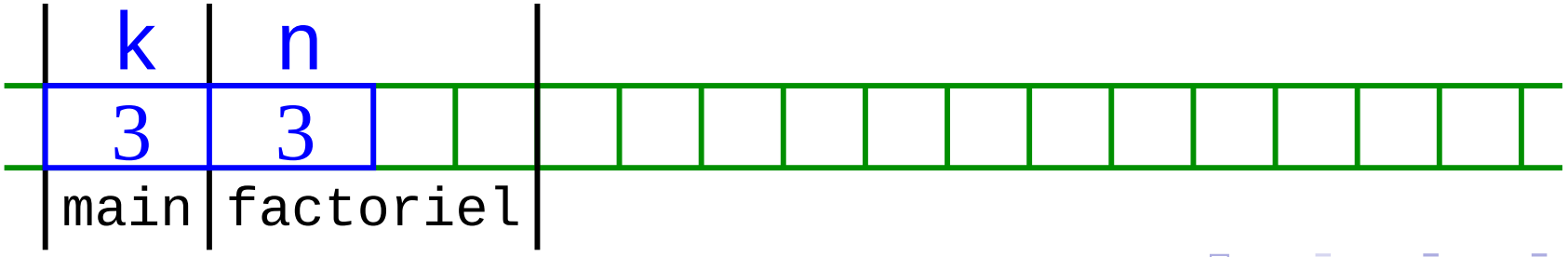
6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  if ( n == 0 )
    return 1;
  return ( n * factoriel( n - 1 ) );
}

int main()
{
  int k = 3;
  cout << factoriel( k ) << endl;
}
```



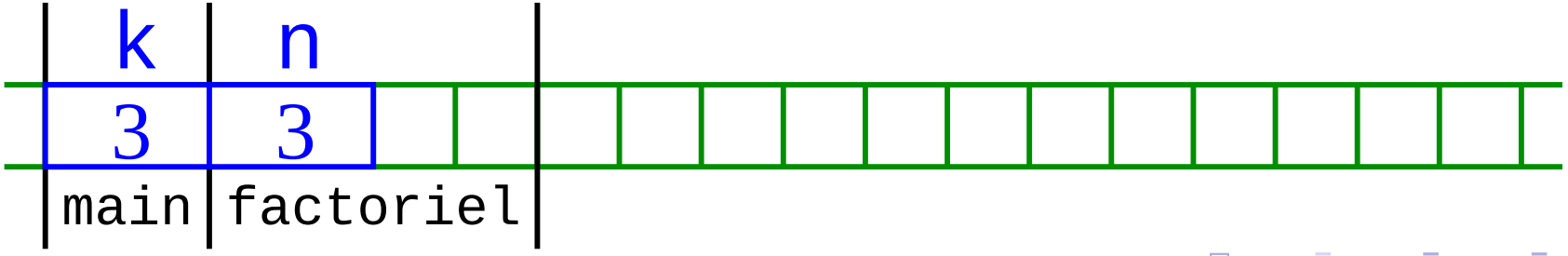
6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  if ( n == 0 )
    return 1;
  return ( n * factoriel( n - 1 )  );
}

int main()
{
  int k = 3;
  cout << factoriel( k ) << endl;
}
```



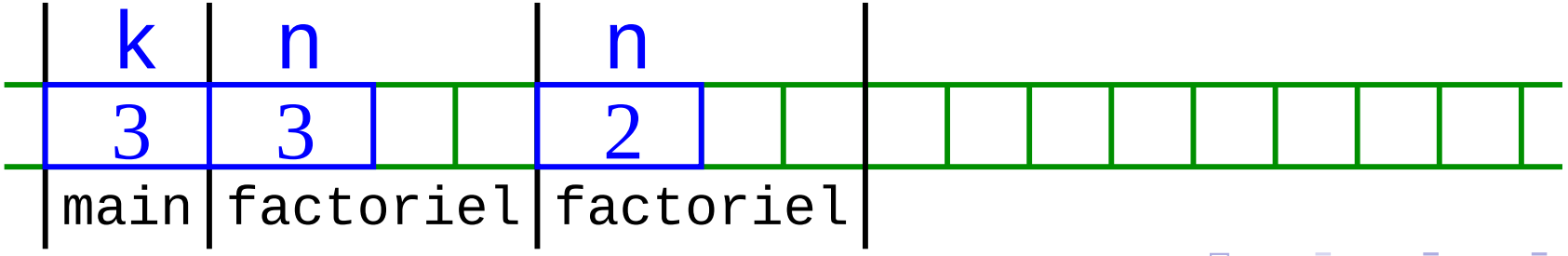
6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  if ( n == 0 )
    return 1;
  return ( n * factoriel( n - 1 ) );
}

int main()
{
  int k = 3;
  cout << factoriel( k ) << endl;
}
```



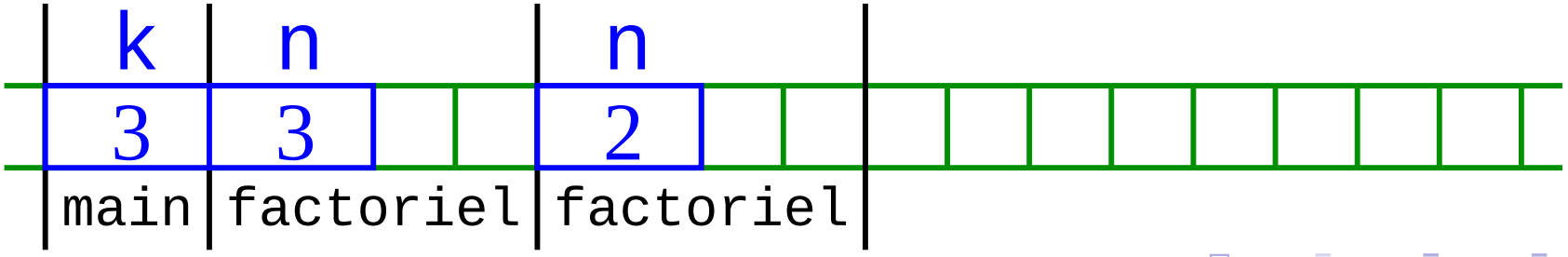
6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  if ( n == 0 )
    return 1;
  return ( n * factoriel( n - 1 )  );
}

int main()
{
  int k = 3;
  cout << factoriel( k ) << endl;
}
```



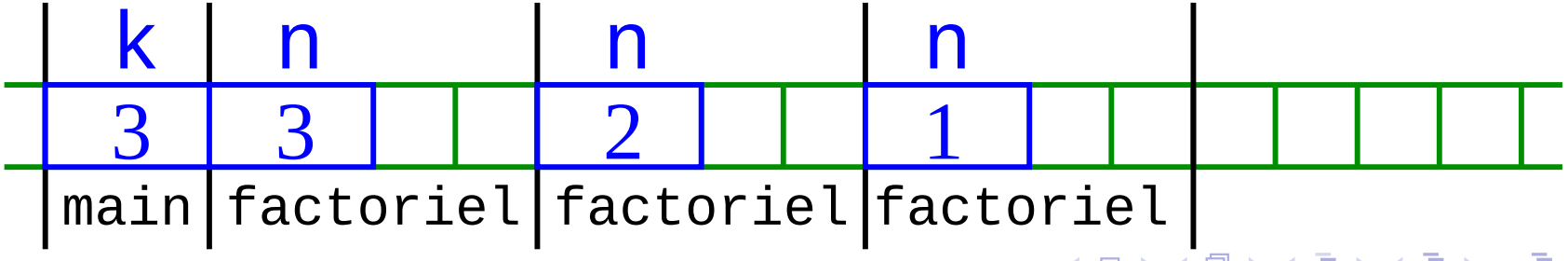
6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  if ( n == 0 )
    return 1;
  return ( n * factoriel( n - 1 ) );
}

int main()
{
  int k = 3;
  cout << factoriel( k ) << endl;
}
```



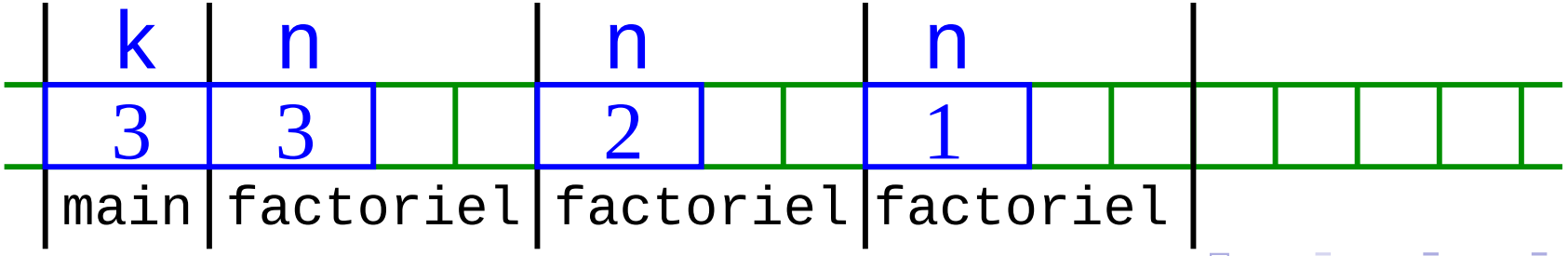
6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  if ( n == 0 )
    return 1;
  return ( n * factoriel( n - 1 )  );
}

int main()
{
  int k = 3;
  cout << factoriel( k ) << endl;
}
```



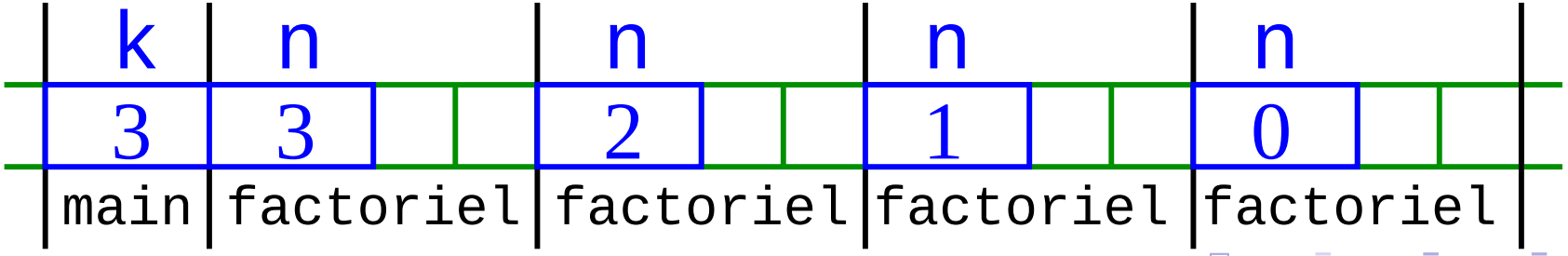
6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  if ( n == 0 )
    return 1;
  return ( n * factoriel( n - 1 ) );
}

int main()
{
  int k = 3;
  cout << factoriel( k ) << endl;
}
```



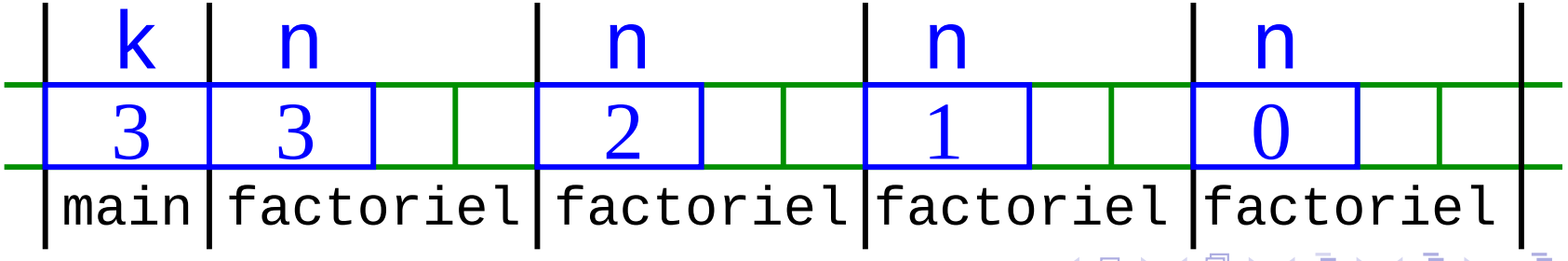
6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  if ( n == 0 )
    return 1;
  return ( n * factoriel( n - 1 ) );
}

int main()
{
  int k = 3;
  cout << factoriel( k ) << endl;
}
```



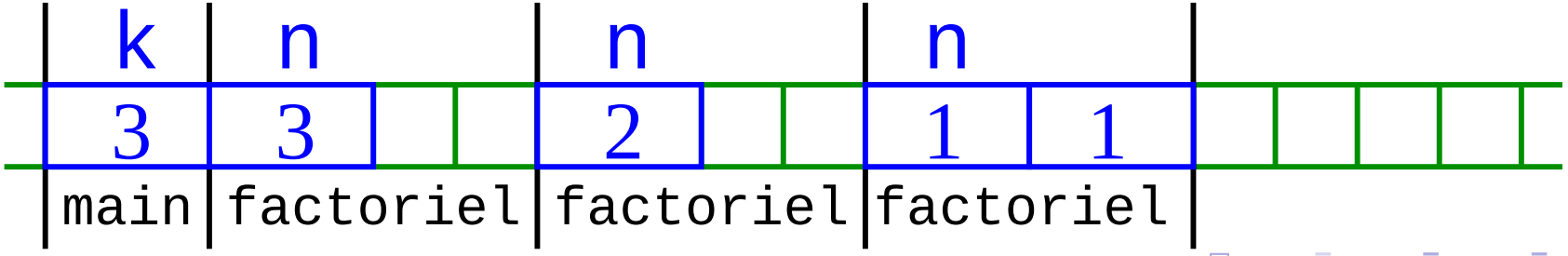
6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  if ( n == 0 )
    return 1;
  return ( n * factoriel( n - 1 ) );
}

int main()
{
  int k = 3;
  cout << factoriel( k ) << endl;
}
```



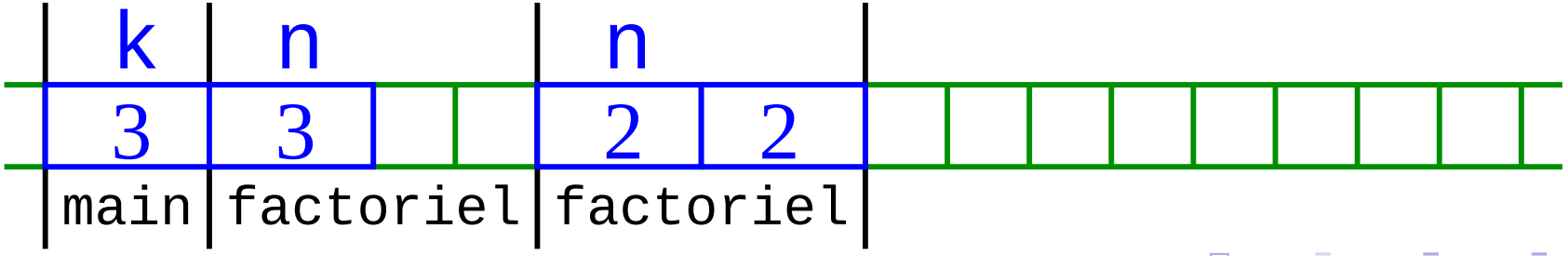
6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  if ( n == 0 )
    return 1;
  return ( n * factoriel( n - 1 ) );
}

int main()
{
  int k = 3;
  cout << factoriel( k ) << endl;
}
```



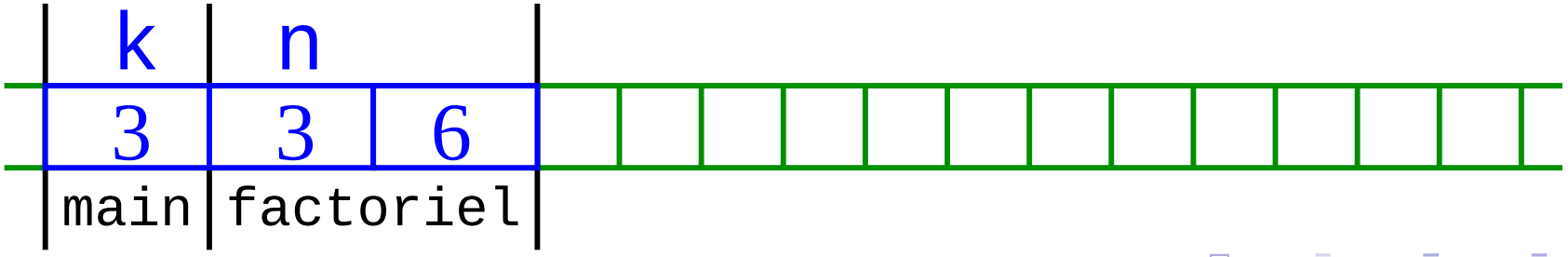
6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  if ( n == 0 )
    return 1;
  return ( n * factoriel( n - 1 ) );
}

int main()
{
  int k = 3;
  cout << factoriel( k ) << endl;
}
```



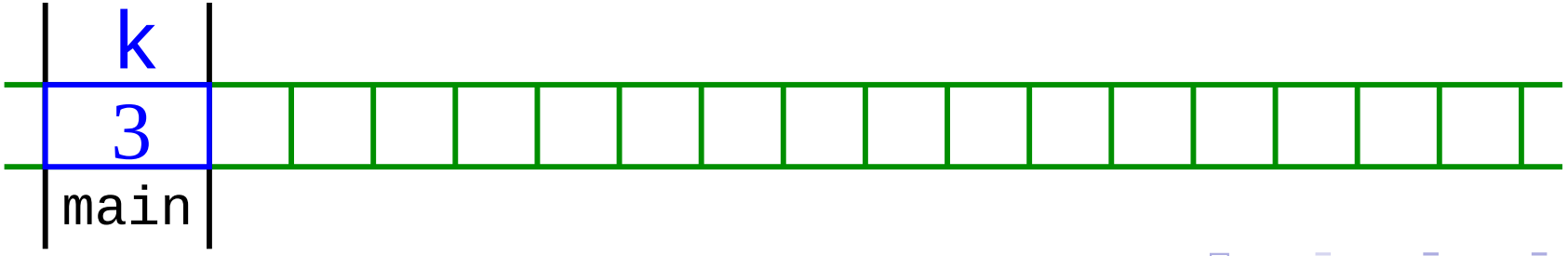
6.4 La récursion

Exemple: calcul de $n! = 1 \times 2 \times \ldots \times n$

Alternative: $n! = n \times (n-1)!$

```
int factoriel( int n )
{
  if ( n == 0 )
    return 1;
  return ( n * factoriel( n - 1 ) );
}

int main()
{
  int k = 3;
  cout << factoriel( k ) << endl;
}
```



6.4 La récursion

$\longrightarrow$ la récursion peut être très utile
dans des contextes spécifiques . . .

. . . mais ce n'est pas toujours la méthode la plus efficace

6.5 Quelques fonctions standards

`double fabs( double x )` $\longrightarrow$ valeur absolu d'un flottant
`double sqrt( double x )` $\longrightarrow$ racine $\sqrt{x}$
`double exp( double x )` $\longrightarrow$ fonction exponentielle e^x
`double log( double x )` $\longrightarrow$ logarithme naturel $\ln(x)$
`double sin( double x )` $\longrightarrow$ sinus (en radians)
`double cos( double x )` $\longrightarrow$ cosinus (en radians)
`double tan( double x )` $\longrightarrow$ tangente (en radians)
`double asin( double x )` $\longrightarrow$ arcsin (en radians)
`double acos( double x )` $\longrightarrow$ arccos (en radians)
`double atan( double x )` $\longrightarrow$ arctan (en radians)

`double atan2( double y, double x )` $\longrightarrow \varphi \in (-\pi, \pi]$
pour lequel $\sin(\varphi) = y/\sqrt{x^2+y^2}$ et $\cos(\varphi) = x/\sqrt{x^2+y^2}$

$\longrightarrow$ il faut inclure `#include <cmath>`
(ou `#include <math.h>` dans C)
pour utiliser ces fonctions mathématiques

6.5 Quelques fonctions standards

Exemple: méthode alternative pour calculer la puissance

$$x^y = e^{\ln(x^y)} = e^{y \ln(x)}$$

$\longrightarrow$ utile pour des exposants y non-entiers

6.5 Quelques fonctions standards

Exemple: méthode alternative pour calculer la puissance

$$x^y = e^{\ln(x^y)} = e^{y \ln(x)}$$

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

double puissance( double k, double n )
{
  return ( exp( n * log( k ) ) );
}
```

6.5 Quelques fonctions standards

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

double puissance( double k, double n )
{
  return ( exp( n * log( k ) ) );
}

int main()
{
  double a, b;
  cin >> a >> b;
  cout << puissance( a, b ) << endl;
}
```

Exécution:

```
4
1.5
8
```

6.5 Quelques fonctions standards

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

double puissance( double k, double n )
{
  return ( exp( n * log( k ) ) );
}

int main()
{
  double a, b;
  cin >> a >> b;
  cout << puissance( a, b ) << endl;
}
```

Exécution:

```
4
-1.5
0.125
```

6.5 Quelques fonctions standards

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

double puissance( double k, double n )
{
  return ( exp( n * log( k ) ) );
}

int main()
{
  double a, b;
  cin >> a >> b;
  cout << puissance( a, b ) << endl;
}
```

Exécution:

```
-4
1.5
nan
```

$\longrightarrow$ not a number

6.5 Quelques fonctions standards

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

double puissance( double k, double n )
{
  return ( exp( n * log( k ) ) );
}

int main()
{
  double a, b;
  cin >> a >> b;
  cout << puissance( a, b ) << endl;
}
```

Exécution:

```
-4
3
nan
```

mais

$$(-4)^3 = -64$$

6.5 Quelques fonctions standards

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

double puissance( double k, int n )
{
  double kn = 1;
  if ( n < 0 )
    for ( int i = -1; i >= n; i-- )
      kn /= k;
  else
    for ( int i = 1; i <= n; i++ )
      kn *= k;
  return kn;
}

double puissance( double k, double n )
{
  if ( k > 0 )
    return ( exp( n * log( k ) ) );
  if ( n == int( n ) )
    return ( puissance( k, int( n ) ) );
  cout << "base negative" << endl;
}

int main()
{
  double a, b;
  cin >> a >> b;
  cout << puissance( a, b ) << endl;
}
```

Exécution:

```
-4
3
-64
```

6.5 Quelques fonctions standards

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

double puissance( double k, int n )
{
  double kn = 1;
  if ( n < 0 )
    for ( int i = -1; i >= n; i-- )
      kn /= k;
  else
    for ( int i = 1; i <= n; i++ )
      kn *= k;
  return kn;
}

double puissance( double k, double n )
{
  if ( k > 0 )
    return ( exp( n * log( k ) ) );
  if ( n == int( n ) )
    return ( puissance( k, int( n ) ) );
  cout << "base negative" << endl;
}

int main()
{
  double a, b;
  cin >> a >> b;
  cout << puissance( a, b ) << endl;
}
```

Exécution:

```
-4
3.2
base negative
4.85953e-270
```

(nombre flottant aléatoire)

6.5 Quelques fonctions standards

`double fabs( double x )` $\longrightarrow$ valeur absolu d'un flottant
`double sqrt( double x )` $\longrightarrow$ racine $\sqrt{x}$
`double exp( double x )` $\longrightarrow$ fonction exponentielle e^x
`double log( double x )` $\longrightarrow$ logarithme naturel $\ln(x)$
`double sin( double x )` $\longrightarrow$ sinus (en radians)
`double cos( double x )` $\longrightarrow$ cosinus (en radians)
`double tan( double x )` $\longrightarrow$ tangente (en radians)
`double asin( double x )` $\longrightarrow$ arcsin (en radians)
`double acos( double x )` $\longrightarrow$ arccos (en radians)
`double atan( double x )` $\longrightarrow$ arctan (en radians)

`double atan2( double y, double x )` $\longrightarrow \varphi \in (-\pi, \pi]$
pour lequel $\sin(\varphi) = y/\sqrt{x^2+y^2}$ et $\cos(\varphi) = x/\sqrt{x^2+y^2}$

`double pow( double x, double y )` $\longrightarrow x^y$

Des nombres aléatoires

$\longrightarrow$ inclure `#include <cstdlib>` (ou `<stdlib.h>` pour C)

`int rand()` $\longrightarrow$ nombre aléatoire entier entre 0 et $2^{31} - 1$

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

int main()
{
  cout << rand() << endl;
  cout << rand() << endl;
}
```

Exécution:

```
1804289383
846930886
```

(toujours)

Des nombres aléatoires

$\longrightarrow$ inclure `#include <cstdlib>` (ou `<stdlib.h>` pour C)

`int rand()` $\longrightarrow$ nombre aléatoire entier entre 0 et $2^{31} - 1$

`void srand( unsigned int seed )`
$\longrightarrow$ *seed* du générateur des nombres aléatoires

combiner avec la fonction `time` dans `<ctime>` (ou `<time.h>`)
(ça rend le temps en secondes découlé depuis le 1/1/1970)

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

int main()
{
  srand( time( 0 ) );
  cout << rand() << endl;
  cout << rand() << endl;
}
```

Exécution:

```
422776193
244047141
```

Des nombres aléatoires

$\longrightarrow$ inclure `#include <cstdlib>` (ou `<stdlib.h>` pour C)

`int rand()` $\longrightarrow$ nombre aléatoire entier entre 0 et $2^{31} - 1$

`void srand( unsigned int seed )`
$\longrightarrow$ *seed* du générateur des nombres aléatoires

combiner avec la fonction `time` dans `<ctime>` (ou `<time.h>`)
(ça rend le temps en secondes découlé depuis le 1/1/1970)

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

int main()
{
  srand( time( 0 ) );
  cout << rand() << endl;
  cout << rand() << endl;
}
```

Exécution:

```
1282589832
1036426062
```

Des nombres aléatoires

$\longrightarrow$ inclure `#include <cstdlib>` (ou `<stdlib.h>` pour C)

`int rand()` $\longrightarrow$ nombre aléatoire entier entre 0 et $2^{31} - 1$

`void srand( unsigned int seed )`
$\longrightarrow$ *seed* du générateur des nombres aléatoires

combiner avec la fonction `time` dans `<ctime>` (ou `<time.h>`)
(ça rend le temps en secondes découlé depuis le 1/1/1970)

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

int main()
{
  srand( time( 0 ) );
  cout << rand() << endl;
  cout << rand() << endl;
}
```

Exécution:

```
1318789015
568117324
```

Des nombres aléatoires

$\longrightarrow$ inclure `#include <cstdlib>` (ou `<stdlib.h>` pour C)

`int rand()` $\longrightarrow$ nombre aléatoire entier entre 0 et $2^{31} - 1$

```
void srand( unsigned int seed )
```
$\longrightarrow$ *seed* du générateur des nombres aléatoires

combiner avec la fonction `time` dans `<ctime>` (ou `<time.h>`)
(ça rend le temps en secondes découlé depuis le 1/1/1970)

$\longrightarrow$ très utile pour des calculs stochastiques
(et pour la programmation des jeux . . .)

La fonction `assert`

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

double puissance( double k, int n )
{
  double kn = 1;
  if ( n < 0 )
    for ( int i = -1; i >= n; i-- )
      kn /= k;
  else
    for ( int i = 1; i <= n; i++ )
      kn *= k;
  return kn;
}

double puissance( double k, double n )
{
  if ( k > 0 )
    return ( exp( n * log( k ) ) );
  assert( n == int( n ) );
  return ( puissance( k, int( n ) ) );
}

int main()
{
  double a, b;
  cin >> a >> b;
  cout << puissance( a, b ) << endl;
}
```

Exécution:

```
-4
3.2
Assertion 'n == int( n )'
failed
```

La fonction `assert`

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

double puissance( double k, int n )
{
  double kn = 1;
  if ( n < 0 )
    for ( int i = -1; i >= n; i-- )
      kn /= k;
  else
    for ( int i = 1; i <= n; i++ )
      kn *= k;
  return kn;
}

double puissance( double k, double n )
{
  if ( k > 0 )
    return ( exp( n * log( k ) ) );
  assert( n == int( n ) );
  return ( puissance( k, int( n ) ) );
}

int main()
{
  double a, b;
  cin >> a >> b;
  cout << puissance( a, b ) << endl;
}
```

Exécution:

```
-4
3
-64
```

Des commandes pour terminer un programme

`assert(` ⟨expression⟩ `);`

⟶ termine le programme si ⟨expression⟩ vaut zéro
après avoir affiché le message
`Assertion '`⟨expression⟩`' failed.`

disponible avec `#include <cassert>` (ou `<assert.h>`)

`exit(` ⟨expression⟩ `);`

⟶ termine le programme et rend la valeur de ⟨expression⟩
comme “valeur de retour” de `main()`

disponible avec `#include <cstdlib>` (ou `<stdlib.h>`)