7 Les fichiers et la compilation

Peter Schlagheck

Université de Liège

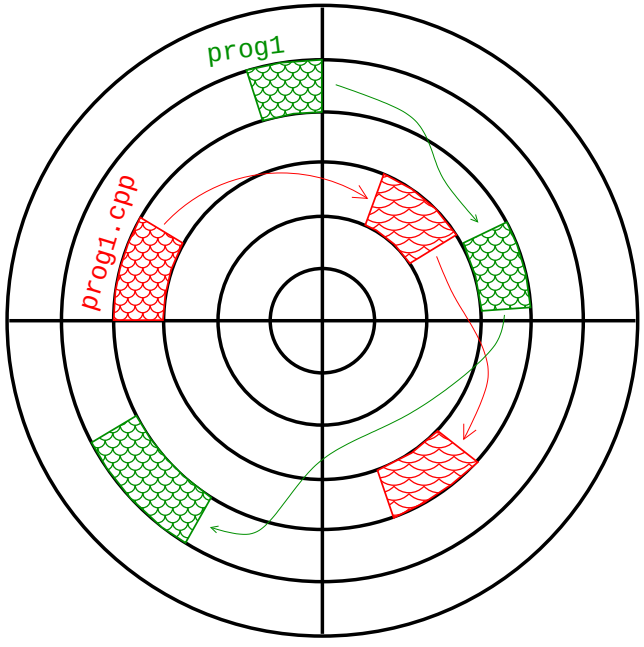
7 Les fichiers et la compilation

7.1 Les fichiers

Un fichier est une collection de données informatiques, représentées en octets, qui se trouve sur le disque dur (ou sur une clé USB / un CD . . .):

`/home/peter/cours/prog/prog1.cpp`

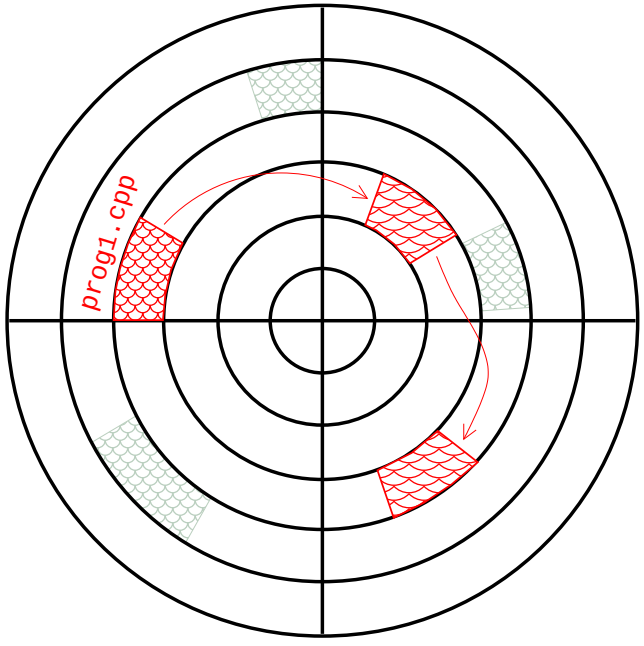
`/home/peter/cours/prog/prog1`



7.1 Les fichiers

Un fichier est une collection de données informatiques, représentées en octets, qui se trouve sur le disque dur (ou sur une clé USB / un CD . . .):

```
/home/peter/cours/prog/prog1.cpp
/home/peter/cours/prog/prog1
rm prog1
```



$\longrightarrow$ enlever des fichiers fait toujours laisser des traces sur le disque (que les professionnels pourront accéder . . .)

Les fichiers ASCII

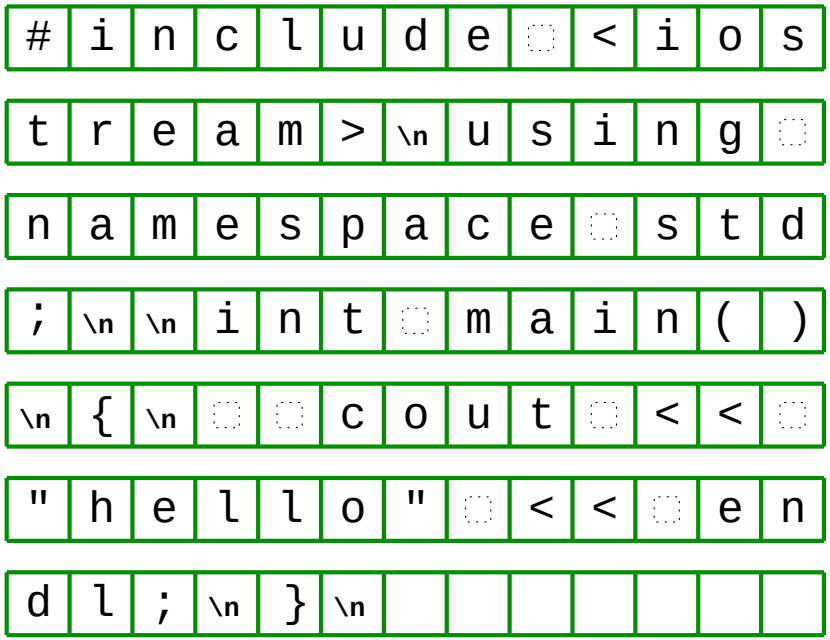
Un fichier ASCII est un fichier qui contient de l'information lexicale, c.-à-d., des caractères encodés selon le code ASCII.

Le fichier `prog1.cpp` ...

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

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

... enregistré sur le disque dur :



Les fichiers ASCII

Un fichier ASCII est un fichier qui contient de l'information lexicale, c.-à-d., des caractères encodés selon le code ASCII.

Le fichier `prog1.cpp` ...

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

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

... enregistré sur le disque dur :

35	105	110	99	108	117	100	101	32	60	105	111	115
116	114	101	97	109	62	10	117	115	105	110	103	32
110	97	109	101	115	112	97	99	101	32	115	116	100
59	10	10	105	110	116	32	109	97	105	110	40	41
10	123	10	32	32	99	111	117	116	32	60	60	32
34	104	101	108	108	111	34	32	60	60	32	101	110
100	108	59	10	125	10							

$\longrightarrow$ un fichier ASCII contient des octets entre 32 et 126 (et éventuellement quelques "octets de contrôle", comme 10 $=$ \n)

Input/output avec des fichiers ASCII

```
#include <iostream>
#include <fstream>
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;
  ofstream out( "fichier.dat" );
  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 ) )
      out << k << "^" << n << " + "
          << l << "^" << n << " = "
          << m << "^" << n << endl;
}
```

Exécution:

```
exposant: 2
nombre maximal: 20
```

$\longrightarrow$ création d'un fichier `fichier.dat` dans le répertoire actuel

Input/output avec des fichiers ASCII

```
#include <iostream>
#include <fstream>
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;
  ofstream out( "fichier.dat" );
  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 ) )
      out << k << "^" << n << " + "
          << l << "^" << n << " = "
          << m << "^" << n << endl;
}
```

Le contenu de `fichier.dat`:

```
3^2 + 4^2 = 5^2
4^2 + 3^2 = 5^2
5^2 + 12^2 = 13^2
6^2 + 8^2 = 10^2
8^2 + 6^2 = 10^2
8^2 + 15^2 = 17^2
9^2 + 12^2 = 15^2
12^2 + 5^2 = 13^2
12^2 + 9^2 = 15^2
12^2 + 16^2 = 20^2
15^2 + 8^2 = 17^2
16^2 + 12^2 = 20^2
```

Input/output avec des fichiers ASCII

```
#include <iostream>
#include <fstream>
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;
  ifstream inp( "param.inp");
  inp >> n;
  inp >> Nmax;
  ofstream out( "fichier.dat" );
  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 ) )
      out << k << "^" << n << " + "
          << l << "^" << n << " = "
          << m << "^" << n << endl;
}
```

$\longrightarrow$ lecture des paramètres `n` et `Nmax` depuis le fichier `param.inp`

ceci pourrait contenir

```
2
20
```

$\longrightarrow$ sortie dans `fichier.dat`

Input/output avec des fichiers ASCII

disponible avec `#include <fstream>`

`ifstream` ⟨nom⟩`(` ⟨string⟩ `);`

⟶ définition d'une "variable" ⟨nom⟩ du type `ifstream`
pour la lecture depuis le fichier ⟨string⟩ (entre guillemets)

`ofstream` ⟨nom⟩`(` ⟨string⟩ `);`

⟶ définition d'une "variable" ⟨nom⟩ du type `ofstream`
pour l'écriture dans le fichier ⟨string⟩

⟶ utilisation comme `cin` et `cout`

Input/output avec des fichiers binaires

Fichier “binaire” $=$ fichier qui contient de l’information binaire générique, sans la restriction aux caractères ASCII.

$\longrightarrow$ fichiers d’exécution de programme (`prog1`)

$\longrightarrow$ fichiers Word, Excel, PowerPoint, pdf, . . .

Input/output avec des fichiers binaires

Fichier “binaire” = fichier qui contient de l’information binaire générique, sans la restriction aux caractères ASCII.

Lecture d’un fichier binaire :

```
ifstream inp( "file.dat" );
char c;
inp.get( c );
```

$\longrightarrow$ lecture d’un caractère (contenu entre 0 et 255) depuis le fichier `file.dat`

Input/output avec des fichiers binaires

Fichier “binaire” = fichier qui contient de l’information binaire générique, sans la restriction aux caractères ASCII.

Ecriture dans un fichier binaire :

```
ofstream out( "file.dat" );
char c = 'A';
out.put( c );
out.put( c + 100 );
out.put( 27 );
out.put( 'e' );
```

$\longrightarrow$ écriture des caractères 65 (’A’), 165, 27, 99 (’e’) dans le fichier `file.dat`

Input/output avec des fichiers binaires

Affichage du contenu d'un fichier binaire :

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

int main()
{
  ifstream inp( "prog1.cpp" );
  char c;
  while ( inp.get( c ) )
    cout << int( c ) << " ";
}
```

Le contenu de `prog1.cpp` :

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

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

Input/output avec des fichiers binaires

Affichage du contenu d'un fichier binaire :

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

int main()
{
  ifstream inp( "prog1.cpp" );
  char c;
  while ( inp.get( c ) )
    cout << int( c ) << " ";
}
```

Le contenu de `prog1.cpp` :

35	105	110	99	108	117	100	101	32	60	105	111	115
116	114	101	97	109	62	10	117	115	105	110	103	32
110	97	109	101	115	112	97	99	101	32	115	116	100
59	10	10	105	110	116	32	109	97	105	110	40	41
10	123	10	32	32	99	111	117	116	32	60	60	32
34	104	101	108	108	111	34	32	60	60	32	101	110
100	108	59	10	125	10							

Exécution:

```
35 105 110 99 108 117 100 101 32 60 105 111 115 116 114
101 97 109 62 10 117 115 105 110 103 32 110 97 109 101
115 112 97 99 101 32 115 116 100 59 10 10 105 110 116 32
109 97 105 110 40 41 10 123 10 32 32 99 111 117 116 32
60 60 32 34 104 101 108 108 111 34 32 60 60 32 101 110
100 108 59 10 125 10
```

Input/output avec des fichiers binaires

Affichage du contenu d'un fichier binaire :

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

int main()
{
  ifstream inp( "prog1" );
  char c;
  while ( inp.get( c ) )
    cout << int( c ) << " ";
}
```

Le contenu de `prog1.cpp` :

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

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

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

Exécution:

```
127 69 76 70 2 1 1 0 0 0 0 0 0 0 0 0 3 0 62 0 1 0 0 0
-80 7 0 0 0 0 0 0 64 0 0 0 0 0 0 0 -104 27 0 0 0 0 0 0 0
0 0 0 64 0 56 0 9 0 64 0 29 0 28 0 6 0 0 0 4 0 0 0 64 0
0 0 0 0 0 0 64 0 0 0 0 0 0 0 64 0 0 0 0 0 0 0 -8 1 0 0 0
0 0 0 -8 1 0 0 0 0 0 0 8 0 0 0 0 0 0 0 3 0 0 0 4 0 0 0
56 2 0 0 0 0 0 0 56 2 0 0 0 0 0 0 56 2 0 0 0 0 0 0 28 0
0 0 0 0 0 0 28 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 5 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 112
11 0 0 0 0 0 0 112 11 0 0 0 0 0 0 0 32 0 0 0 0 0 1 0 0
0 6 0 0 0 120 13 0 0 0 0 0 0 120 13 32 0 0 0 0 0 120 13
32 0 0 0 0 0 -104 2 0 0 0 0 0 0 -64 3 0 0 0 0 0 0 0 0 32
0 0 0 0 0 2 0 0 0 6 0 0 0 -112 13 0 0 0 0 0 0 -112 13 32
0 0 0 0 0 -112 13 32 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2 0 0 0
0 0 0 8 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 84 2 0 0 0 0 0 0
```

7.2 Le processus de la compilation

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

int fonction1( ... )
// utilise sin(..) et log(..) de <cmath>
{
  ...
}

void fonction2( ... )
// appelle fonction1
{
  ...
}

double fonction3( ... )
// appelle fonction1 et fonction2
{
  ...
}

int main()
// appelle fonction2 et fonction3
// utilise cout/cin et ifstream/ofstream
{
  ...
}
```

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

⟶ fonction1:

```
01101011
11010001
...
```

⟶ fonction2:

```
11100100
01010011
...
```

⟶ fonction3:

```
00011010
10000111
...
```

⟶ main:

```
11011101
00110010
...
```

7.2 Le processus de la compilation

La commande

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

traduit toutes les fonctions définies dans `prog1.cpp`
en langage de machine.

De plus, un fichier prêt à exécuter est créé avec le nom `prog1`

La commande `./prog1` commence avec l'exécution de la fonction `main()`

7.2 Le processus de la compilation

Problème:

Pour le fonctionnement d'un programme ordinaire
on a besoin de beaucoup de fonctions/routines standards. . .

$\longrightarrow$ des fonctions mathématiques

```
#include <cmath>
```

7.2 Le processus de la compilation

Problème:

Pour le fonctionnement d'un programme ordinaire
on a besoin de beaucoup de fonctions/routines standards. . .

$\longrightarrow$ des fonctions mathématiques

$\longrightarrow$ des routines (et types) entrée / sortie

```
#include <iostream>
```

7.2 Le processus de la compilation

Problème:

Pour le fonctionnement d'un programme ordinaire
on a besoin de beaucoup de fonctions/routines standards. . .

$\longrightarrow$ des fonctions mathématiques

$\longrightarrow$ des routines (et types) entrée / sortie

$\longrightarrow$ des routines (et types) pour le traitement des fichiers

```
#include <fstream>
```

7.2 Le processus de la compilation

Problème:

Pour le fonctionnement d'un programme ordinaire
on a besoin de beaucoup de fonctions/routines standards. . .

$\longrightarrow$ des fonctions mathématiques

$\longrightarrow$ des routines (et types) entrée / sortie

$\longrightarrow$ des routines (et types) pour le traitement des fichiers

$\longrightarrow$ . . .

. . . et beaucoup d'autres fonctions élémentaires "invisibles"
que de telles fonctions (ou routines/types) utilisent dans leur définition interne

$\longrightarrow$ ça ferait un lourd processus de compilation . . .

7.2 Le processus de la compilation

Problème:

Pour le fonctionnement d'un programme ordinaire
on a besoin de beaucoup de fonctions/routines standards. . .

Solution:

De telles fonctions standards sont déjà précompilées en langage de machine.

Elles sont disponibles dans des bibliothèques standards qui sont automatiquement prises en compte lors de l'exécution du programme.

7.2 Le processus de la compilation

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

int fonction1( ... )
// utilise sin(..) et log(..) de <cmath>
{
  ...
}

void fonction2( ... )
// appelle fonction1
{
  ...
}

double fonction3( ... )
// appelle fonction1 et fonction2
{
  ...
}

int main()
// appelle fonction2 et fonction3
// utilise cout/cin et ifstream/ofstream
{
  ...
}
```

`c++ prog1.cpp -o prog1`

compile d'abord les fonctions `fonction1`, `fonction2`, `fonction3`, `main` (processus de compilation)

et ajoute ensuite un lien avec les bibliothèques standards (processus de *linking*) avant de créer l'exécutable `prog1`

7.2 Le processus de la compilation

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

int fonction1( ... )
// utilise sin(..) et log(..) de <cmath>
{
  ...
}

void fonction2( ... )
// appelle fonction1
{
  ...
}

double fonction3( ... )
// appelle fonction1 et fonction2
{
  ...
}

int main()
// appelle fonction2 et fonction3
// utilise cout/cin et ifstream/ofstream
{
  ...
}
```

`c++ -c prog1.cpp`

compilation de
`fonction1`, `fonction2`,
`fonction3`, `main`

et création d'un "fichier d'objet"
`prog1.o` qui contient
la traduction de ces fonctions
en langage de machine

`c++ prog1.o -o prog1`

processus de *linking* avec
les bibliothèques standards et
création de l'exécutable `prog1`

7.2 Le processus de la compilation

Comment peut-on appeler des fonctions standards si leurs définitions ne sont pas inclues dans le code à compiler ?

$\longrightarrow$ on inclut leur déclaration dans le programme

7.3 Déclaration et définition des fonctions

La définition de la fonction `puissance`:

```
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;
}
```

La déclaration de la fonction `puissance`:

```
double puissance( double k, int n );
```

7.3 Déclaration et définition des fonctions

La déclaration

⟨type⟩ ⟨nom⟩ (⟨type1⟩ ⟨nom1⟩, ⟨type2⟩ ⟨nom2⟩, ...);

dans un fichier C/C++ informe le compilateur
qu'une fonction nommée

⟨nom⟩ (⟨type1⟩, ⟨type2⟩, ...)

qui rend une valeur du type ⟨type⟩
sera disponible dans le programme final

Le compilateur compile ce fichier donc
comme si cette fonction était explicitement définie là-dedans

Déclaration et définition de la fonction `puissance`

Le contenu du fichier `prog1.cpp`:

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

double puissance( double k, int n );
// declaration de puissance

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

`c++ -c prog1.cpp`

$\longrightarrow$ création du fichier objet `prog1.o` contenant
la définition de la fonction `main`
en langage de machine

Déclaration et définition de la fonction `puissance`

Le contenu du fichier `prog1.cpp`:

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

double puissance( double k, int n );
// declaration de puissance

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

`c++ prog1.cpp -o prog1`

$\longrightarrow$ erreur:

```
undefined reference to 'puissance(double, int)'
```

Déclaration et définition de la fonction `puissance`

Le contenu du fichier `prog1.cpp`:

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

double puissance( double k, int n );
// declaration de puissance

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

Le contenu du fichier `puis.cpp`:

```
double puissance( double k, int n )
// definition de puissance
{
  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;
}
```

`c++ -c puis.cpp`

$\longrightarrow$ création du fichier objet `puis.o` contenant la définition de la fonction `puissance` en langage de machine

Déclaration et définition de la fonction `puissance`

Le contenu du fichier `prog1.cpp`:

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

double puissance( double k, int n );
// declaration de puissance

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

Le contenu du fichier `puis.cpp`:

```
double puissance( double k, int n )
// definition de puissance
{
  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;
}
```

`c++ puis.cpp -o puissance`

$\longrightarrow$ erreur:

undefined reference to 'main'

Déclaration et définition de la fonction `puissance`

Le contenu du fichier `prog1.cpp`:

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

double puissance( double k, int n );
// declaration de puissance

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

Le contenu du fichier `puis.cpp`:

```
double puissance( double k, int n )
// definition de puissance
{
  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;
}
```

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

$\longrightarrow$ création de l'exécutable `prog1`:
linking des deux fonctions `puissance` et `main`

Déclaration et définition de la fonction `puissance`

Le contenu du fichier `prog1.cpp`:

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

double puissance( double k, int n );
// declaration de puissance

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

Le contenu du fichier `puis.cpp`:

```
double puissance( double k, int n )
// definition de puissance
{
  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;
}
```

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

$\longrightarrow$ création de l'exécutable `prog1`:
compilation de `prog1.cpp` et linking avec `puis.o`

Déclaration et définition de la fonction `puissance`

Le contenu du fichier `prog1.cpp`:

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

double puissance( double k, int n );
// declaration de puissance

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

Le contenu du fichier `puis.cpp`:

```
double puissance( double k, int n )
// definition de puissance
{
  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;
}
```

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

$\longrightarrow$ création de l'exécutable `prog1`:
compilation et linking de `prog1.cpp` et `puis.cpp`

Déclaration et définition de la fonction `puissance`

Le contenu du fichier `prog1.cpp`:

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

double puissance( double a, int b );
// declaration de puissance

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

Le contenu du fichier `puis.cpp`:

```
double puissance( double k, int n )
// definition de puissance
{
  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;
}
```

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

$\longrightarrow$ pas de problème

Déclaration et définition de la fonction `puissance`

Le contenu du fichier `prog1.cpp`:

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

double puissance( double k, double n );
// declaration de puissance

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

Le contenu du fichier `puis.cpp`:

```
double puissance( double k, int n )
// definition de puissance
{
  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;
}
```

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

$\longrightarrow$ `erreur:`

`undefined reference to 'puissance(double, double)'`

$\longrightarrow$ pas de conversion implicite entre déclarations et définitions

Déclaration et définition de la fonction `puissance`

Le contenu du fichier `prog1.cpp`:

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

double puissance( double &k, int &n );
// declaration de puissance

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

Le contenu du fichier `puis.cpp`:

```
double puissance( double k, int n )
// definition de puissance
{
  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;
}
```

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

$\longrightarrow$ erreur:

```
undefined reference to 'puissance(double&, int&)'
```

Déclaration et définition de la fonction `puissance`

Le contenu du fichier `prog1.cpp`:

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

double puissance( double &k, int &n );
// declaration de puissance

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

Le contenu du fichier `puis.cpp`:

```
double puissance( double &k, int &n )
// definition de puissance
{
  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;
}
```

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

$\longrightarrow$ pas de problème

Déclaration et définition dans un seul fichier

Le contenu du fichier `prog1.cpp`:

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

double puissance( double k, int n );
// declaration de puissance

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

double puissance( double k, int n )
// definition de puissance
{
  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;
}
```

$\longrightarrow$ ça peut rendre les programmes plus lisibles

7.4 Les headers

le fichier `prog1.cpp`:

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

double puissance( double k, int n );

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

le fichier `puis.cpp`:

```
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;
}
```

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

crée l'exécutable `prog1` contenant les fonctions `main` et `puissance(double, int)`

7.4 Les headers

le fichier `prog1.cpp`:

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

double puissance( double k, int n );
int puissance( int k, unsigned int n );

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

le fichier `puis.cpp`:

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

7.4 Les headers

le fichier `prog2.cpp`:

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

int puissance( int k, unsigned int n );
void sort( int &a, int &b );

int main()
{
  int k, n;
  cin >> k >> n;
  sort( k, n );
  cout << puissance( k, n ) << endl;
}
```

le fichier `sort.cpp`:

```
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 );
}
```

`c++ prog2.cpp puis.cpp sort.cpp -o prog2`

$\longrightarrow$ output de k^n si $k > n$, et de n^k sinon

7.4 Les headers

le fichier `prog3.cpp`:

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

int factoriel( int n );
void sort( int &a, int &b );

int main()
{
  int n, m;
  cin >> n >> m;
  sort( n, m );
  int n_m = factoriel( n )
          / factoriel( m )
          / factoriel( n - m );
  cout << n_m << endl;
}
```

le fichier `fact.cpp`:

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

```
c++ prog3.cpp sort.cpp fact.cpp -o prog3
```

$\longrightarrow$ calcul de $\binom{n}{m} = \frac{n!}{m!(n-m)!}$

7.4 Les headers

le fichier `prog3.cpp`:

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

int n_sur_m( int n, int m );
void sort( int &a, int &b );

int main()
{
  int n, m;
  cin >> n >> m;
  sort( n, m );
  int n_m = n_sur_m( n, m );
  cout << n_m << endl;
}
```

le fichier `fact.cpp`:

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

int n_sur_m( int n, int m )
{
  return ( factoriel( n ) /
           factoriel ( m ) /
           factoriel( n - m ) );
}
```

Comment éviter d'écrire toujours les mêmes déclarations dans des différents programmes ?

$\longrightarrow$ utilisation des headers

7.4 Les headers

le fichier `prog3.cpp`:

```
#include <iostream>
#include "fact.h"
using namespace std;

void sort( int &a, int &b );

int main()
{
  int n, m;
  cin >> n >> m;
  sort( n, m );
  int n_m = n_sur_m( n, m );
  cout << n_m << endl;
}
```

le fichier `fact.cpp`:

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

int n_sur_m( int n, int m )
{
  return ( factoriel( n ) /
           factoriel ( m ) /
           factoriel( n - m ) );
}
```

le fichier `fact.h`:

```
int factoriel( int n );
int n_sur_m( int n, int m );
```

$\longrightarrow$ pour le compilateur, `#include "fact.h"` fait inclure le contenu de `fact.h` dans `prog3.cpp`

7.4 Les headers

le fichier `prog3.cpp`:

```
#include <iostream>
#include "fact.h"
using namespace std;

void sort( int &a, int &b );

int main()
{
  int n, m;
  cin >> n >> m;
  sort( n, m );
  int n_m = n_sur_m( n, m );
  cout << n_m << endl;
}
```

le fichier `fact.cpp`:

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

int n_sur_m( int n, int m )
{
  return ( factoriel( n ) /
           factoriel ( m ) /
           factoriel( n - m ) );
}
```

le fichier `fact.h`:

```
int factoriel( int n );
int n_sur_m( int n, int m );
```

$\longrightarrow$ `fact.h` contient l'interface des fonctions `factoriel(int)` et `n_sur_m(int, int)`

$\longrightarrow$ `fact.cpp` contient leur implémentation

7.4 Les headers

le fichier `prog3.cpp`:

```
#include <iostream>
#include "fact.h"
#include "sort.h"
using namespace std;


int main()
{
  int n, m;
  cin >> n >> m;
  sort( n, m );
  int n_m = n_sur_m( n, m );
  cout << n_m << endl;
}
```

le fichier `sort.cpp`:

```
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 );
}
```

le fichier `sort.h`:

```
void echange( int &a, int &b );
void sort( int &a, int &b );
void sort( int &a, int &b, int &c );
```

7.4 Les headers

le fichier `prog2.cpp`:

```
#include <iostream>
#include "sort.h"
using namespace std;

int puissance( int k, unsigned int n );

int main()
{
  int k, n;
  cin >> k >> n;
  sort( k, n );
  cout << puissance( k, n ) << endl;
}
```

le fichier `sort.cpp`:

```
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 );
}
```

le fichier `sort.h`:

```
void echange( int &a, int &b );
void sort( int &a, int &b );
void sort( int &a, int &b, int &c );
```

7.4 Les headers

le fichier `prog2.cpp`:

```
#include <iostream>
#include "puis.h"
#include "sort.h"
using namespace std;


int main()
{
  int k, n;
  cin >> k >> n;
  sort( k, n );
  cout << puissance( k, n ) << endl;
}
```

le fichier `puis.cpp`:

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

le fichier `puis.h`:

```
double puissance( double k, int n );
int puissance( int k, unsigned int n );
```

7.4 Les headers

le fichier `prog1.cpp`:

```
#include <iostream>
#include "puis.h"
using namespace std;

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

le fichier `puis.cpp`:

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

le fichier `puis.h`:

```
double puissance( double k, int n );
int puissance( int k, unsigned int n );
```

7.4 Les headers

Des headers peuvent contenir

- des déclarations des fonctions

```
int factoriel( int n );
```

7.4 Les headers

Des headers peuvent contenir

- des déclarations des fonctions
- des déclarations des constantes

```
const double pi = 3.14159265359;
```

$\longrightarrow$ `M_PI` dans `<cmath>`

7.4 Les headers

Des headers peuvent contenir

- des déclarations des fonctions
- des déclarations des constantes
- des nouveaux types / des définitions des classes

  $\longrightarrow$ `ifstream` et `ofstream` dans `<fstream>`

7.4 Les headers

Des headers peuvent contenir

- des déclarations des fonctions
- des déclarations des constantes
- des nouveaux types / des définitions des classes

En principe, chaque fonction pourrait avoir son propre header et son propre fichier d'implémentation . . .

. . . mais on fait bien structurer son programme en mettant ensemble dans un seul fichier ce qui va bien ensemble

7.5 Les directives du compilateur

`#include "`⟨nom⟩`"`

inclut le fichier ⟨nom⟩ dans le programme.

Le compilateur cherche ce fichier dans le répertoire actuel.

7.5 Les directives du compilateur

`#include "`⟨nom⟩`"`

`#include "`⟨rep⟩`/`⟨nom⟩`"`

inclut le fichier ⟨nom⟩ dans le programme.

Le compilateur cherche ce fichier dans le sous-répertoire ⟨rep⟩ du répertoire actuel.

7.5 Les directives du compilateur

`#include "`⟨nom⟩`"`

`#include "`⟨rep⟩`/`⟨nom⟩`"`

`#include <`⟨nom⟩`>`

inclut le fichier ⟨nom⟩ dans le programme.

Le compilateur cherche ce fichier dans des répertoires standards `/usr/include`, `/usr/include/c++`, ...
(sous Linux / MAC OS)

... et aussi dans le sous-répertoire ⟨rep⟩ du répertoire actuel s'il est lancé avec l'option `-I`:

`c++ -c` ⟨fichier.cpp⟩ `-I`⟨rep⟩

7.5 Les directives du compilateur

- `#include ...`

  représente une directive pour le compilateur
  qui lui demande à inclure le fichier en question
  dans le programme actuel

  Cette directive est traitée par un préprocesseur
  qui est (automatiquement) lancé avant le “vrai” compilateur

7.5 Les directives du compilateur

le fichier `prog3.cpp`:

```
#include <iostream>
#include "fact.h"
#include "sort.h"
using namespace std;


int main()
{
  int n, m;
  cin >> n >> m;
  sort( n, m );
  int n_m = n_sur_m( n, m );
  cout << n_m << endl;
}
```

. . . “vu” par le compilateur après le préprocesseur:

```
...
// le contenu de <iostream>
...

int factoriel( int n );
int n_sur_m( int n, int m );

void echange( int &a, int &b );
void sort( int &a, int &b );
void sort( int &a, int &b, int &c );

using namespace std;

int main()
{
  int n, m;
  cin >> n >> m;
  sort( n, m );
  int n_m = n_sur_m( n, m );
  cout << n_m << endl;
}
```

7.5 Les directives du compilateur

- `#include ...`

- `#define ...`

  `#define` ⟨nom⟩ ⟨expression⟩ permet de définir des constantes et des macros

  ```
  #define pi 3.14159265359
  ```

  ⟶ définition du macro `pi` qui sera remplacé par `3.14159265359` lors de la compilation

  ```
  #define carre(x) (x)*(x)
  ```

  ⟶ définition du macro `carre` qui fait que des expressions `carre(`⟨arg⟩`)` soient remplacées par `(`⟨arg⟩`)*(`⟨arg⟩`)` lors de la compilation

7.5 Les directives du compilateur

le fichier `fermat2.cpp`:

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

#define carre(x) (x)*(x)

int main()
{
  int Nmax = 10;
  for ( int k = 1; k <= Nmax; k++ )
  for ( int l = 1; l <= Nmax; l++ )
  for ( int m = 1; m <= Nmax; m++ )
    if ( carre( k ) + carre( l )
         == carre( m ) )
      cout << k << "^2+" << l << "^2="
           << m << "^2" << endl;
}
```

. . . "vu" par le compilateur après le préprocesseur:

```
...
// le contenu de <iostream>
...
using namespace std;


int main()
{
  int Nmax = 10;
  for ( int k = 1; k <= Nmax; k++ )
  for ( int l = 1; l <= Nmax; l++ )
  for ( int m = 1; m <= Nmax; m++ )
    if ( ( k ) * ( k ) + ( l ) * ( l )
         == ( m ) * ( m ) )
      cout << k << "^2+" << l << "^2="
           << m << "^2" << endl;
}
```

$\longrightarrow$ n'utilisez pas de tels macros trop excessivement !

en général, il est préférable de les remplacer par des "vraies" fonctions

7.5 Les directives du compilateur

le fichier `fermat2.cpp`:

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

int carre( int x )
{
  return ( x * x );
}

int main()
{
  int Nmax = 10;
  for ( int k = 1; k <= Nmax; k++ )
  for ( int l = 1; l <= Nmax; l++ )
  for ( int m = 1; m <= Nmax; m++ )
    if ( carre( k ) + carre( l )
         == carre( m ) )
      cout << k << "^2+" << l << "^2="
           << m << "^2" << endl;
}
```

7.5 Les directives du compilateur

- `#include ...`
- `#define ...`
- `#if ...`, `#ifdef ...`, `#ifndef ...`, `#else`, `#endif`

  $\longrightarrow$ définition des branchements conditionnels pour le compilateur

7.5 Les directives du compilateur

Exemple:

calcul de la fonction Bessel $J_0(x) = \frac{1}{\pi} \int_0^{\pi} \cos(\varphi - x \sin \varphi) d\varphi$

- avec la fonction `nag_bessel_j0(double)`
  de la bibliothèque commerciale NAG
  (`www.nag.co.uk`)
- ...ou, si la dernière n'est pas disponible,
  avec la routine `gsl_sf_bessel_J0(double)`
  de la bibliothèque gratuite GSL
  (*GNU Scientific Library*)

$\longrightarrow$ interface (dans `bessel.h`):

```
double bessel( double x );
```

7.5 Les directives du compilateur

⟶ implémentation (dans `bessel.cpp`):

```
#ifdef _NAG_
#include <nag.h>
#else
#include <gsl_sf_bessel.h>
#endif

double bessel( double x )
{
#ifdef _NAG_
  return ( nag_bessel_j0( x ) );
#else
  return ( gsl_sf_bessel_J0( x ) );
#endif
}
```

code “vu” par le compilateur:

```
#include <nag.h>

double bessel( double x )
{
  return ( nag_bessel_j0( x ) );
}
```

Compilation pour l’utilisation de la routine de NAG:

```
c++ -c bessel.cpp -D_NAG_
```

⟶ l’option `-D_NAG_` définit l’expression `_NAG_`

`#ifdef _NAG_` sera donc vrai

7.5 Les directives du compilateur

$\longrightarrow$ implémentation (dans `bessel.cpp`):

```
#ifdef _NAG_
#include <nag.h>
#else
#include <gsl_sf_bessel.h>
#endif

double bessel( double x )
{
#ifdef _NAG_
  return ( nag_bessel_j0( x ) );
#else
  return ( gsl_sf_bessel_J0( x ) );
#endif
}
```

code “vu” par le compilateur:

```
#include <gsl_sf_bessel.h>

double bessel( double x )
{
  return ( gsl_sf_bessel_J0( x ) );
}
```

Compilation pour l’utilisation de la routine de GSL:

```
c++ -c bessel.cpp
```

$\longrightarrow$ `_NAG_` n’est pas défini

`#ifdef _NAG_` sera donc faux

7.5 Les directives du compilateur

$\longrightarrow$ implémentation (dans `bessel.cpp`):

```
#ifdef _NAG_
#include <nag.h>
#else
#include <gsl_sf_bessel.h>
#endif

double bessel( double x )
{
#ifdef _NAG_
  return ( nag_bessel_j0( x ) );
#else
  return ( gsl_sf_bessel_J0( x ) );
#endif
}
```

Compilation et linking avec la bibliothèque de NAG:

```
c++ -c bessel.cpp -D_NAG_
c++ prog.cpp bessel.o -o prog -lnag
```

$\longrightarrow$ l'option `-lnag` crée un lien avec le fichier `libnag.a` qui se trouve dans un des répertoires standards (e.g. `/usr/lib`) pour des bibliothèques

7.5 Les directives du compilateur

$\longrightarrow$ implémentation (dans `bessel.cpp`):

```
#ifdef _NAG_
#include <nag.h>
#else
#include <gsl_sf_bessel.h>
#endif

double bessel( double x )
{
#ifdef _NAG_
  return ( nag_bessel_j0( x ) );
#else
  return ( gsl_sf_bessel_J0( x ) );
#endif
}
```

Compilation et linking avec la bibliothèque de GSL:

```
c++ -c bessel.cpp
c++ prog.cpp bessel.o -o prog -lgsl
```