9 Les structures et les classes

Peter Schlagheck

Université de Liège

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

9 Les structures et les classes

9.1 Les structures

Exemple: traiter des dates de naissance de 20 personnes

```
const int N = 20;
int jour[ N ];
int mois[ N ];
int an[ N ];

// lecture des dates
for ( int i = 0; i < N; i++ )
  lire( jour[ i ], mois[ i ], an[ i ] );

// ordonner les dates
tri( N, jour, mois, an );

// affichage les dates ordonnees
affichage( N, jour, mois, an );
```

```
void lire( int &j, int &m, int &a )
{
  cout << "rentrez votre ";
  cout << "date de naissance:" << endl;
  cout << "jour:" << endl;
  cin >> j;
  cout << "mois:" << endl;
  cin >> m;
  cout << "an:" << endl;
  cin >> a;
}
```

```
void affichage( int N, int *j, int *m, int *a )
{
  for ( int i = 0; i < N; i++ )
    cout << j[ i ] << "/" << m[ i ] << "/" << a[ i ] << endl;
}
```

$\longrightarrow$ gestion de 3 tableaux séparés nécessaire

$\rightarrow$ des erreurs sont possibles . . .

9.1 Les structures

Exemple: traiter des dates de naissance de 20 personnes

```
const int N = 20;
int jour[ N ];
int mois[ N ];
int an[ N ];

// lecture des dates
for ( int i = 0; i < N; i++ )
  lire( jour[ i ], mois[ i ], an[ i ] );

// ordonner les dates
tri( N, an, mois, jour );

// affichage les dates ordonnees
affichage( N, jour, mois, an );
```

```
void lire( int &j, int &m, int &a )
{
  cout << "rentrez votre ";
  cout << "date de naissance:" << endl;
  cout << "jour:" << endl;
  cin >> j;
  cout << "mois:" << endl;
  cin >> m;
  cout << "an:" << endl;
  cin >> a;
}
```

```
void affichage( int N, int *j, int *m, int *a )
{
  for ( int i = 0; i < N; i++ )
    cout << j[ i ] << "/" << m[ i ] << "/" << a[ i ] << endl;
}
```

$\longrightarrow$ gestion de 3 tableaux séparés nécessaire

$\rightarrow$ des erreurs sont possibles . . .

9.1 Les structures

Exemple: traiter des dates de naissance de 20 personnes

```
const int N = 20;
int jour[ N ];
int mois[ N ];
int an[ N ];

// lecture des dates
for ( int i = 0; i < N; i++ )
  lire( jour[ i ], mois[ i ], an[ i ] );

// ordonner les dates
tri( N, jour, mois, an );

// affichage les dates ordonnees
affichage( N, jour, mois, an );
```

```
void lire( int &j, int &m, int &a )
{
  cout << "rentrez votre ";
  cout << "date de naissance:" << endl;
  cout << "jour:" << endl;
  cin >> j;
  cout << "mois:" << endl;
  cin >> m;
  cout << "an:" << endl;
  cin >> a;
}
```

```
void affichage( int N, int *j, int *m, int *a )
{
  for ( int i = 0; i < N; i++ )
    cout << j[ i ] << "/" << m[ i ] << "/" << a[ i ] << endl;
}
```

$\longrightarrow$ gestion de 3 tableaux séparés nécessaire

$\longrightarrow$ utilisation d'une structure

9.1 Les structures

```
struct date
{
  int j;
  int m;
  int a;
};
```

$\longrightarrow$ ceci définit un nouveau type nommé `date`

qui est composé de trois variables `j`, `m`, `a` du type `int`

9.1 Les structures

```
struct date
{
  int j;
  int m;
  int a;
};

const int N = 20;
date naiss[ N ];

// lecture des dates
for ( int i = 0; i < N; i++ )
  lire( naiss[ i ] );

// ordonner les dates
tri( N, naiss );

// affichage les dates ordonnees
affichage( N, naiss );
```

9.1 Les structures

```
struct date
{
  int j;
  int m;
  int a;
};

void lire( date &d );
void tri( int N, date *d );
void affichage( int N, date *d );

int main()
{
  const int N = 20;
  date naiss[ N ];
  for ( int i = 0; i < N; i++ )
    lire( naiss[ i ] );
  tri( N, naiss );
  affichage( N, naiss );
}
```

```
void lire( date &d )
{
  cout << "rentrez votre ";
  cout << "date de naissance:" << endl;
  cout << "jour:" << endl;
  cin >> d.j;
  cout << "mois:" << endl;
  cin >> d.m;
  cout << "an:" << endl;
  cin >> d.a;
}

void affichage( int N, date *d )
{
  for ( int i = 0; i < N; i++ )
    cout << d[ i ].j << "/" << d[ i ].m
         << "/" << d[ i ].a << endl;
}
```

9.1 Les structures

```
#include "date.h"

int main()
{
  const int N = 20;
  date naiss[ N ];
  for ( int i = 0; i < N; i++ )
    lire( naiss[ i ] );
  tri( N, naiss );
  affichage( N, naiss );
}
```

Le contenu de `date.h`:

```
struct date
{
  int j;
  int m;
  int a;
};

void lire( date &d );
void tri( int N, date *d );
void affichage( int N, date *d );
```

Le contenu de `date.cpp`:

```
#include "date.h"

void lire( date &d )
{
  cout << "rentrez votre ";
  cout << "date de naissance:" << endl;
  cout << "jour:" << endl;
  cin >> d.j;
  cout << "mois:" << endl;
  cin >> d.m;
  cout << "an:" << endl;
  cin >> d.a;
}

void affichage( int N, date *d )
{
  for ( int i = 0; i < N; i++ )
    cout << d[ i ].j << "/" << d[ i ].m
         << "/" << d[ i ].a << endl;
}

void tri( int N, date *d )
{
  ...
}
```

9.1 Les structures

```
struct ⟨stype⟩
{
   ⟨type1⟩ ⟨nom1⟩;
   ⟨type2⟩ ⟨nom2⟩;
   ...
};
```

$\longrightarrow$ définition du type ⟨stype⟩: il est composé des variables ⟨nom1⟩ du type ⟨type1⟩, ⟨nom2⟩ du type ⟨type2⟩, ...

9.1 Les structures

```
struct ⟨stype⟩
{
   ⟨type1⟩ ⟨nom1⟩;
   ⟨type2⟩ ⟨nom2⟩;
   ...
};
```

Définition d'une variable nommée ⟨snom⟩ du type ⟨stype⟩:

```
⟨stype⟩ ⟨snom⟩;
```

Accès aux composantes de la variable ⟨snom⟩
($\rightarrow$ les "membres" de la structure ⟨stype⟩) :

```
⟨snom⟩.⟨nom1⟩
⟨snom⟩.⟨nom2⟩
...
```

9.1 Les structures

```
struct ⟨stype⟩
{
   ⟨type1⟩ ⟨nom1⟩;
   ⟨type2⟩ ⟨nom2⟩;
   ...
};
```

Définition d'une variable nommée ⟨snom⟩ du type ⟨stype⟩:

```
⟨stype⟩ ⟨snom⟩;
```

$\longrightarrow$ on peut utiliser ⟨snom⟩ . ⟨nom1⟩, ⟨snom⟩ . ⟨nom2⟩, ...
de la même façon que toutes les autres variables
des types ⟨type1⟩, ⟨type2⟩, ...

9.1 Les structures

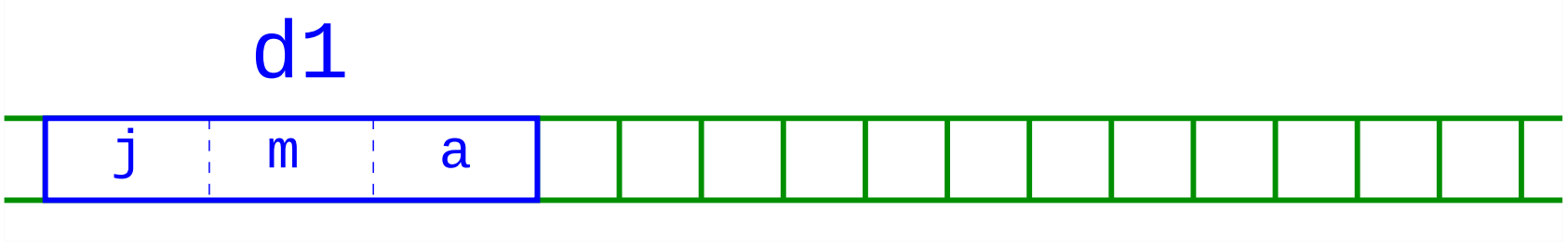
```
void lire( date &d )
{
  cout << "rentrez votre ";
  cout << "date de naissance:" << endl;
  cout << "jour:" << endl;
  cin >> d.j;
  cout << "mois:" << endl;
  cin >> d.m;
  cout << "an:" << endl;
  cin >> d.a;
  if ( ( d.j > 31 ) || ( d.m > 12 ) || ( d.a > 2018 ) )
  {
    cout << "rentrez svp une date valable!"<< endl;
    lire( d );
  }
}
```

9.1 Les structures

```
#include "date.h"

int main()
{
  date d1;
  date d2;
  lire( d1 );
  lire( d2 );
  int jour_diff = 365 * ( d2.a - d1.a )
                + 30 * ( d2.m - d1.m )
                + ( d2.j - d1.j );
  cout << "vos dates de naissance different environ ";
       << "par " << jour_diff << " jours" << endl;
}
```

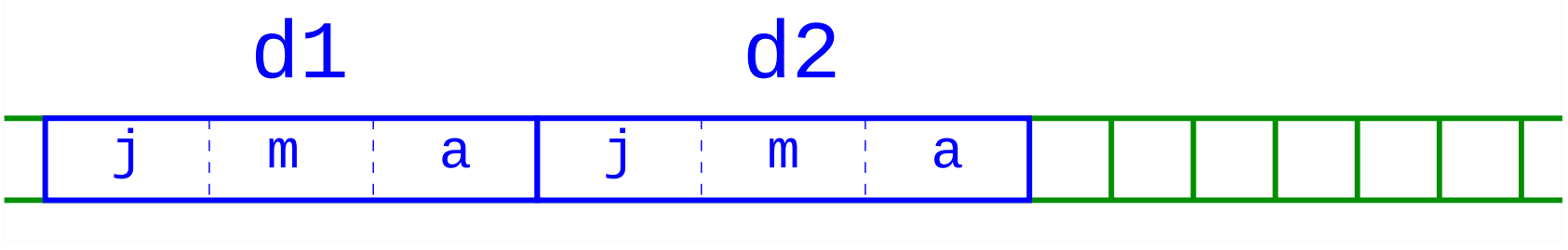
9.1 Les structures

```
#include "date.h"

int main()
{
  date d1;
  date d2;
  lire( d1 );
  lire( d2 );
  int jour_diff = 365 * ( d2.a - d1.a )
                + 30 * ( d2.m - d1.m )
                + ( d2.j - d1.j );
  cout << "vos dates de naissance different environ ";
       << "par " << jour_diff << " jours" << endl;
}
```



9.1 Les structures

```
#include "date.h"

int main()
{
  date d1;
  date d2;
  lire( d1 );
  lire( d2 );
  int jour_diff = 365 * ( d2.a - d1.a )
                + 30 * ( d2.m - d1.m )
                + ( d2.j - d1.j );
  cout << "vos dates de naissance different environ ";
       << "par " << jour_diff << " jours" << endl;
}
```



9.1 Les structures

On peut combiner des variables de types différents dans une structure ...

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  // date de naissance
  int jour;
  int mois;
  int an;
};
```

9.1 Les structures

On peut combiner des variables de types différents dans une structure . . . même des variables d'autres structures

```
struct date
{
  int j;
  int m;
  int a;
};

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  date naiss;
};
```

9.1 Les structures

On peut combiner des variables de types différents dans une structure . . . même des variables d'autres structures

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  date naiss;
  personne parent1;
  personne parent2;
};
```

⟶ erreur:

```
field 'parent1' has incomplete type
field 'parent2' has incomplete type
```

9.1 Les structures

On peut combiner des variables de types différents dans une structure . . . même des variables d'autres structures

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  date naiss;
  personne *parent1;
  personne *parent2;
};
```

9.1 Les structures

Les structures permettent

- de rassembler des variables qui logiquement tiennent ensemble

```
struct date
{
  int j;
  int m;
  int a;
};
```

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  date naiss;
};
```

9.1 Les structures

Les structures permettent

- de rassembler des variables qui logiquement tiennent ensemble
- de séparer les différentes hierarchies d'un programme
- de mieux gérer des programmes complexes

```
personne p[ 20 ];
p[ 1 ].naiss.m = 5;
```

...est mieux gérable que

```
char nom[ 20 ][ 50 ];
char prenom[ 20 ][ 50 ];
int jour[ 20 ];
int mois[ 20 ];
int an[ 20 ];
mois[ 1 ] = 5;
```

Les structures et les fonctions

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  date naiss;
};
```

```
void affichage( personne p )
{
  cout << p.prenom << " " << p.nom << endl;
}
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  lire( p.naiss );
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  affichage( p1 );
}
```

$\longrightarrow$ facile d'ajouter des éléments supplémentaires . . .

Les structures et les fonctions

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( personne p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  affichage( p1 );
}
```

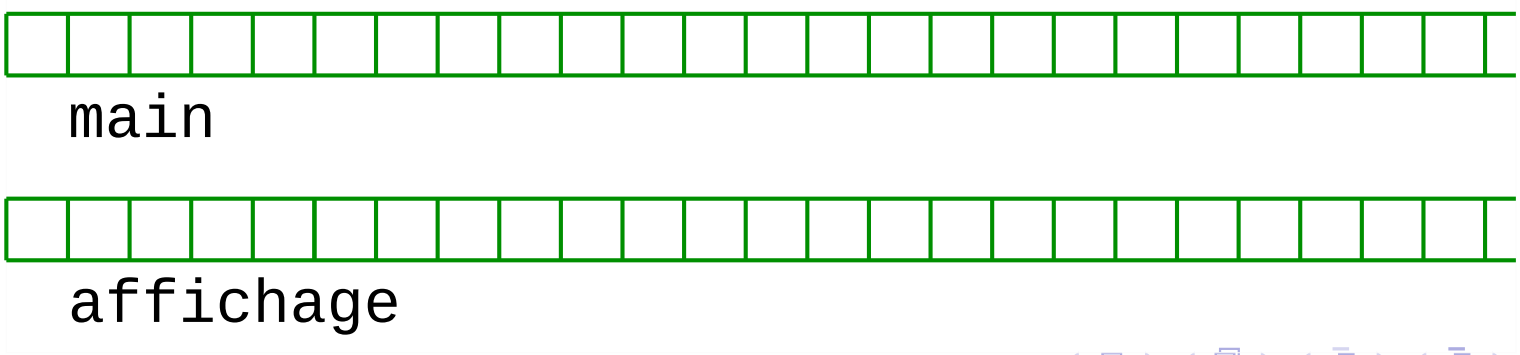
Les structures et les fonctions

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( personne p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  affichage( p1 );
}
```



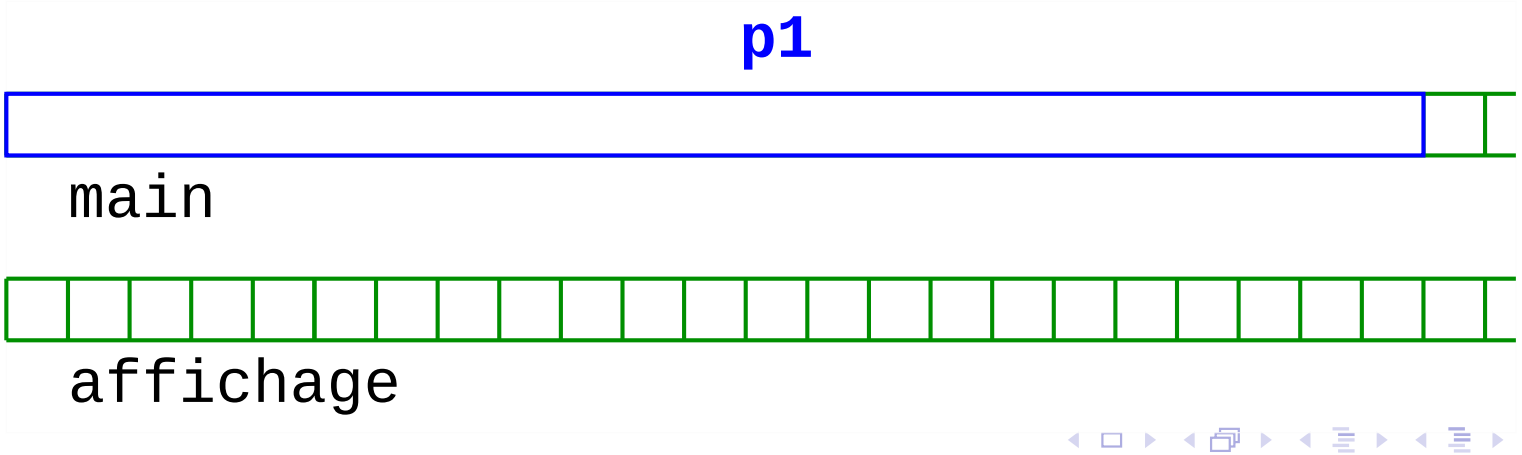
Les structures et les fonctions

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( personne p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  affichage( p1 );
}
```



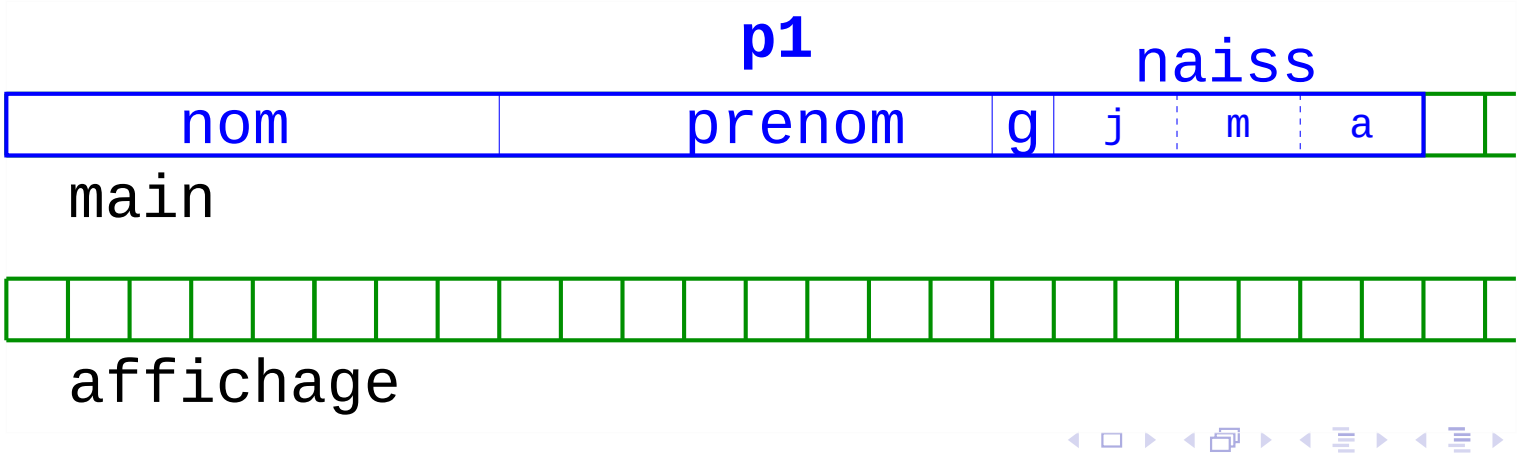
Les structures et les fonctions

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( personne p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  affichage( p1 );
}
```



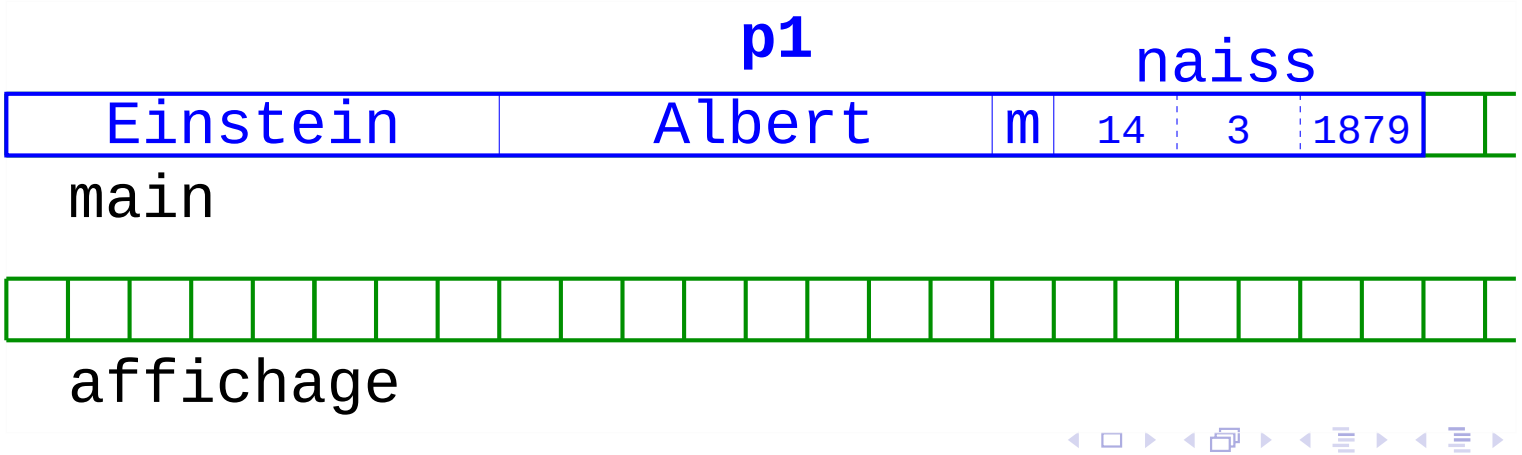
Les structures et les fonctions

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( personne p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  affichage( p1 );
}
```



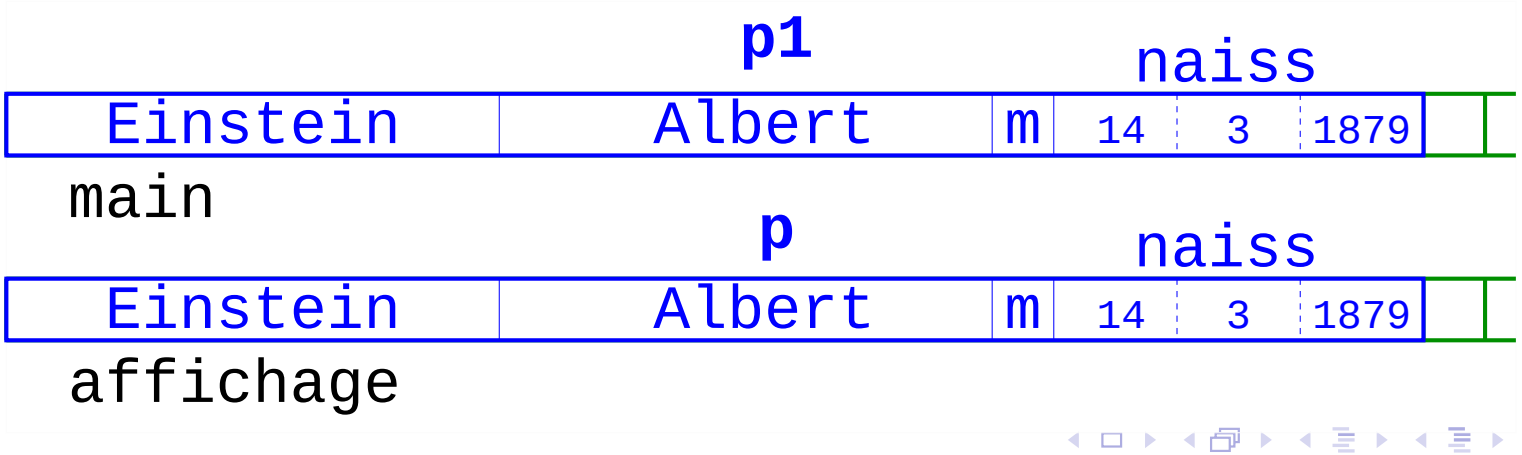
Les structures et les fonctions

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( personne p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  affichage( p1 );
}
```



Les structures et les fonctions

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( personne p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  affichage( p1 );
}
```

$\longrightarrow$ évitez de copier des grosses structures:

travailler plutôt avec l'original dans des fonctions

Les structures et les fonctions

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( personne &p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  affichage( p1 );
}
```

$\longrightarrow$ évitez de copier des grosses structures:

travailler plutôt avec l'original dans des fonctions

Mais attention: ne pas faire des manipulation involontaires!

Les structures et les fonctions

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( personne &p )
{
  if ( p.genre = 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  affichage( p1 );
}
```

$\longrightarrow$ évitez de copier des grosses structures:

travailler plutôt avec l'original dans des fonctions

Mais attention: ne pas faire des manipulation involontaires!

(Exécution: `Mme Albert Einstein`)

Les structures et les fonctions

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( const personne &p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  affichage( p1 );
}
```

$\longrightarrow$ évitez de copier des grosses structures:

travailler plutôt avec l'original dans des fonctions

$\longrightarrow$ utiliser une référence constante

Les structures et les fonctions

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( const personne &p )
{
  if ( p.genre = 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  affichage( p1 );
}
```

$\longrightarrow$ erreur: assignment of data-member 'personne::genre' in read-only structure

Les références constantes

⟨type⟩ ⟨nom⟩`( ..., const` ⟨type1⟩ `&`⟨nom1⟩`, ...);`

→ déclaration d'une fonction ⟨nom⟩ qui prend, comme un de ses arguments, une référence constante à une variable du type ⟨type1⟩

→ la fonction travaille avec l'original de cette variable, mais n'a pas de droit de la modifier quelque part

⟶ utilité pour des variables des types composés (structures, classes)

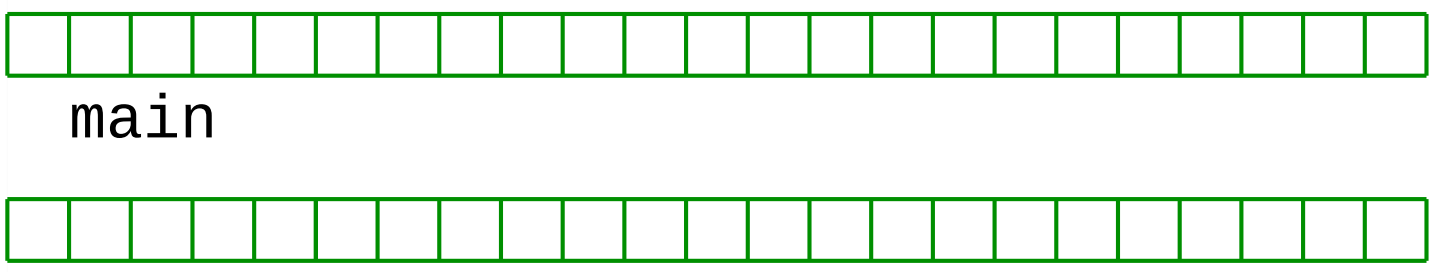
La copie des structures

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( const personne &p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  p2.genre = 'f';
}
```



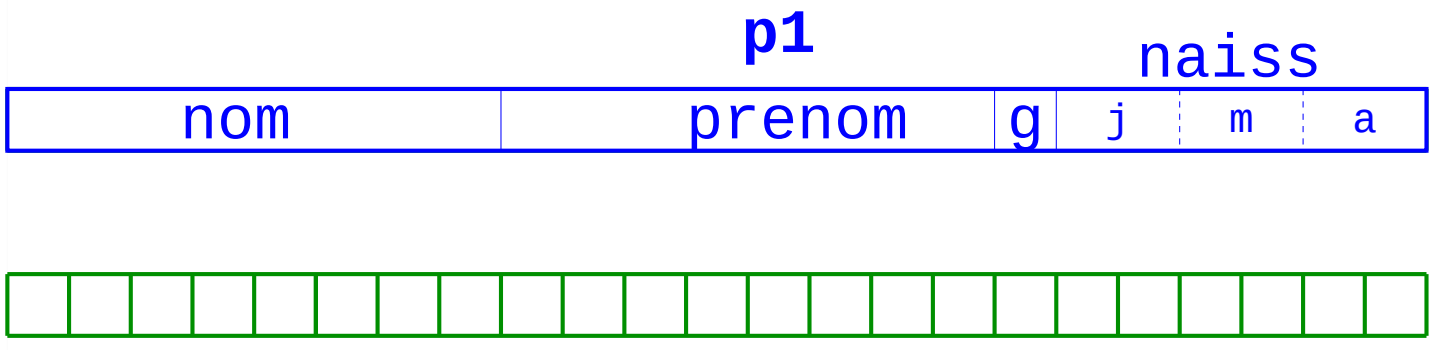
La copie des structures

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( const personne &p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  p2.genre = 'f';
}
```



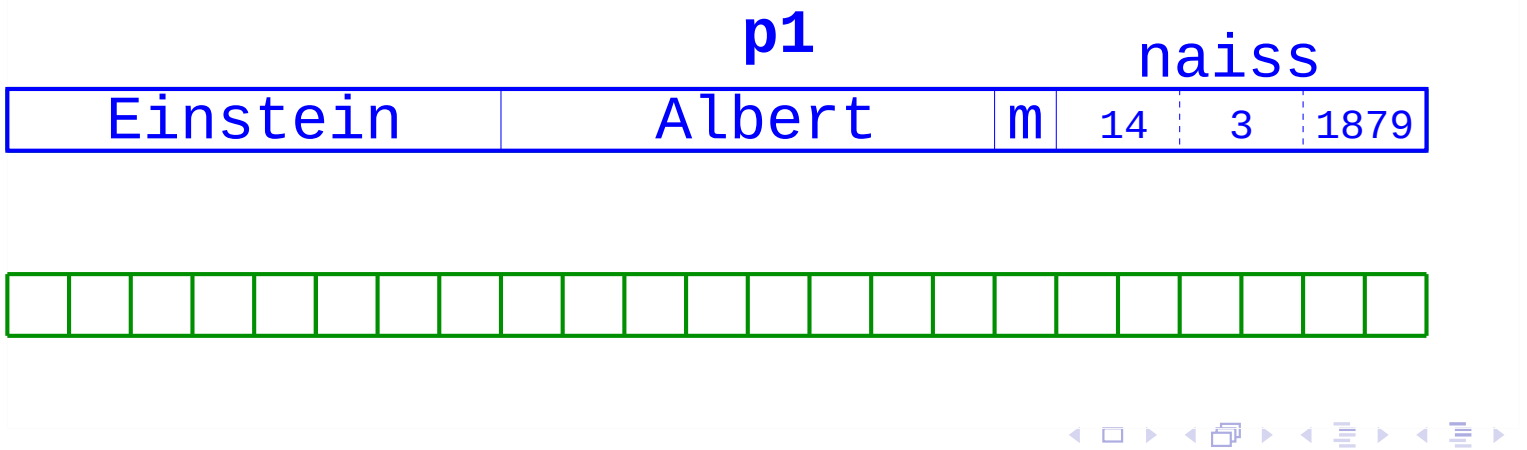
La copie des structures

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( const personne &p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  p2.genre = 'f';
}
```



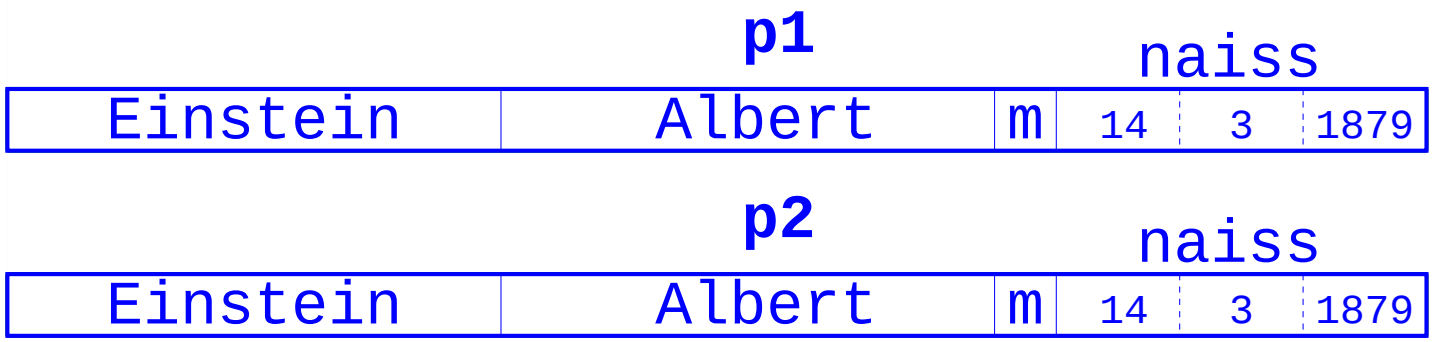
La copie des structures

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( const personne &p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  p2.genre = 'f';
}
```



La copie des structures

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( const personne &p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  p2.genre = 'f';
}
```

p1

			naiss		
Einstein	Albert	m	14	3	1879

p2

			naiss		
Einstein	Albert	f	14	3	1879

La copie des structures

```
#include "date.h"

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;
};
```

```
void registration( personne &p )
{
  cout << "nom:";
  cin >> p.nom;
  cout << "prenom:";
  cin >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}
```

```
void affichage( const personne &p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " " << p.nom << endl;
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  p2.genre = 'f';
}
```

L'opération de copie `p2 = p1` effectue une copie de chaque membre de la structure `personne`.

Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};
```

```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```

```
int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```

Exécution:

```
nom et prenom:
```

Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```

Exécution:

```
nom et prenom: Clinton
Bill
```

Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```

Exécution:

```
nom et prenom: Clinton
Bill
nombre d'enfants:
```

Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```

Exécution:

```
nom et prenom: Clinton
Bill
nombre d'enfants: 1
```

Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```

Exécution:

```
nom et prenom: Clinton
Bill
nombre d'enfants: 1
prenom de l'enfant 1:
```

Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```

Exécution:

```
nom et prenom: Clinton
Bill
nombre d'enfants: 1
prenom de l'enfant 1: Chelsea
```

Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```

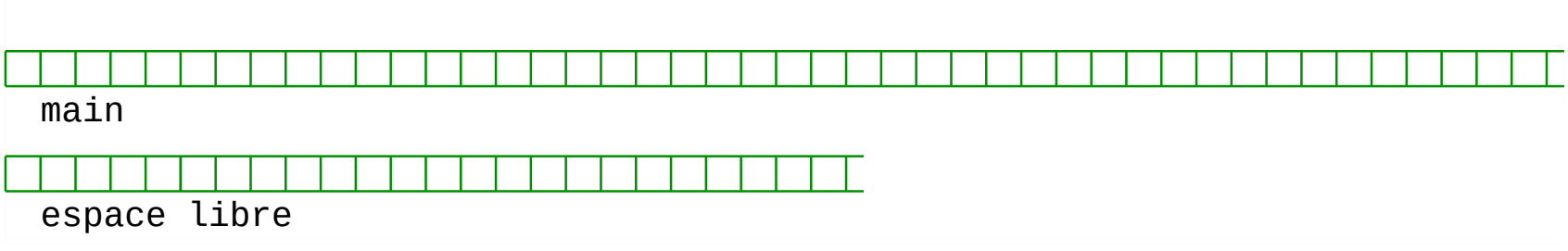
Exécution:

```
nom et prenom: Clinton
Bill
nombre d'enfants: 1
prenom de l'enfant 1: Chelsea
Claire
```

Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

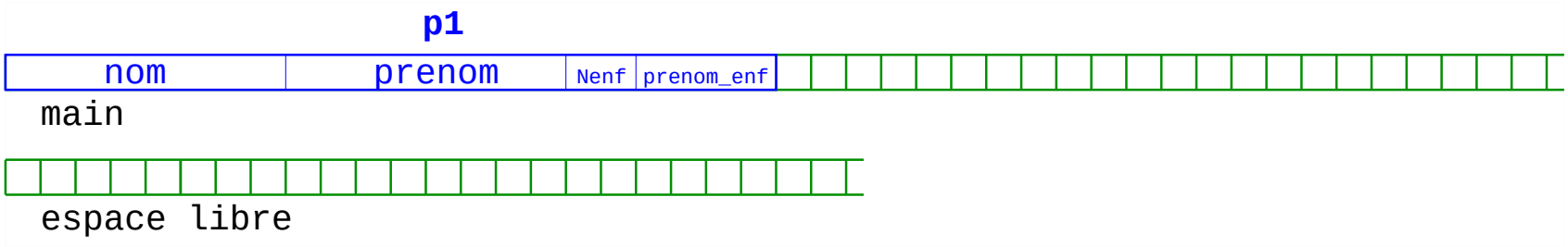
```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```



Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

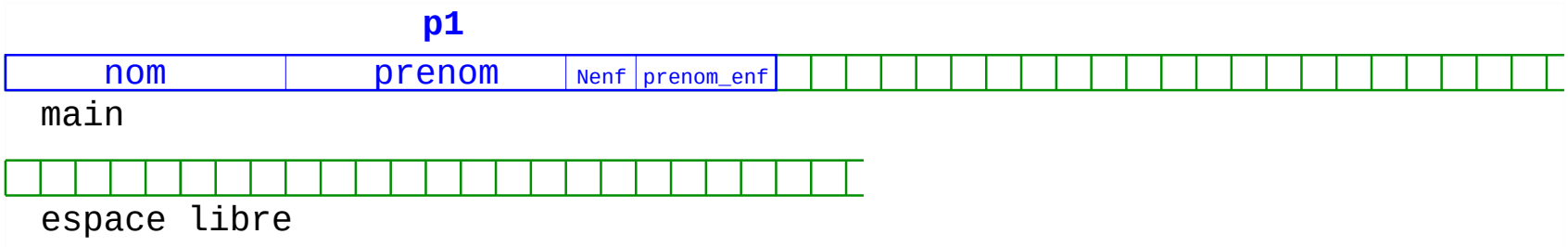
```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```



Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

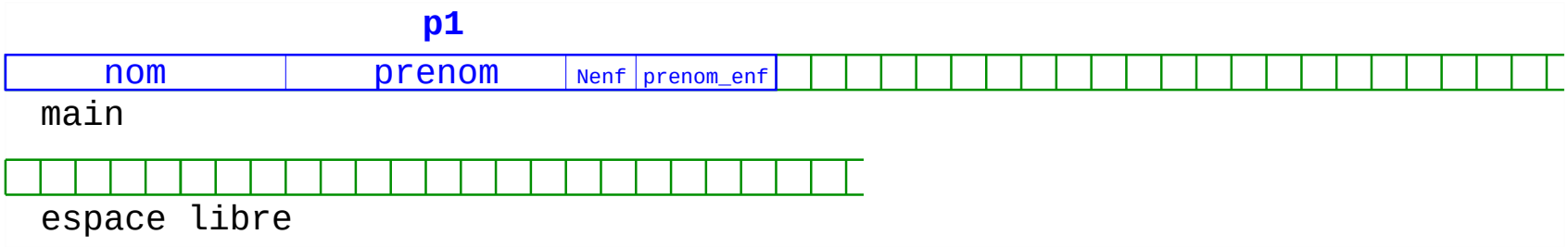
```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```



Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

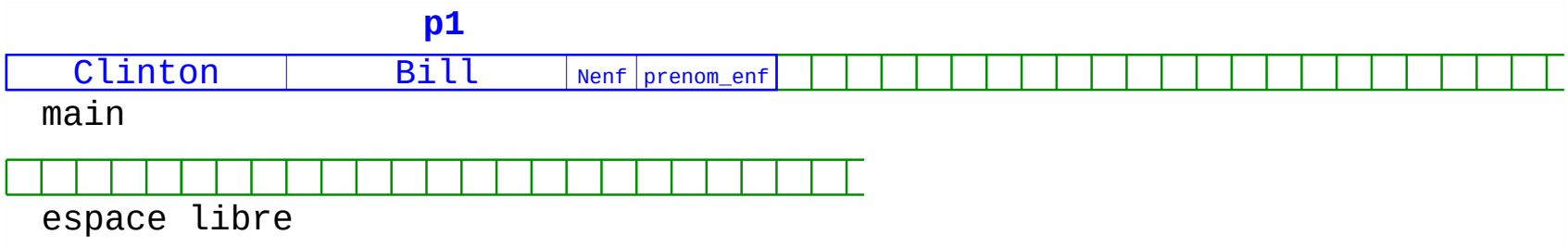
```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```



Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

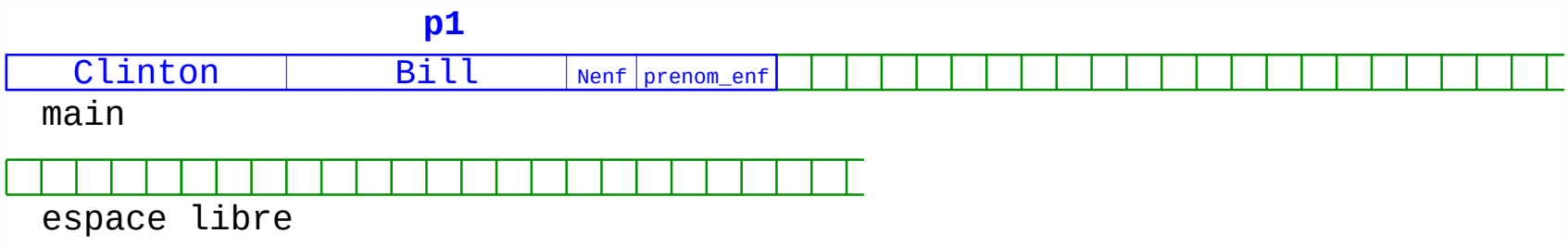
```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```



Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

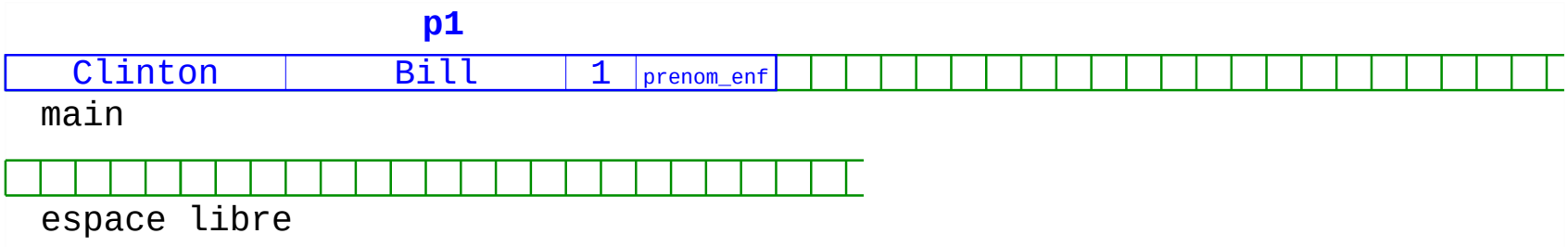
```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```



Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

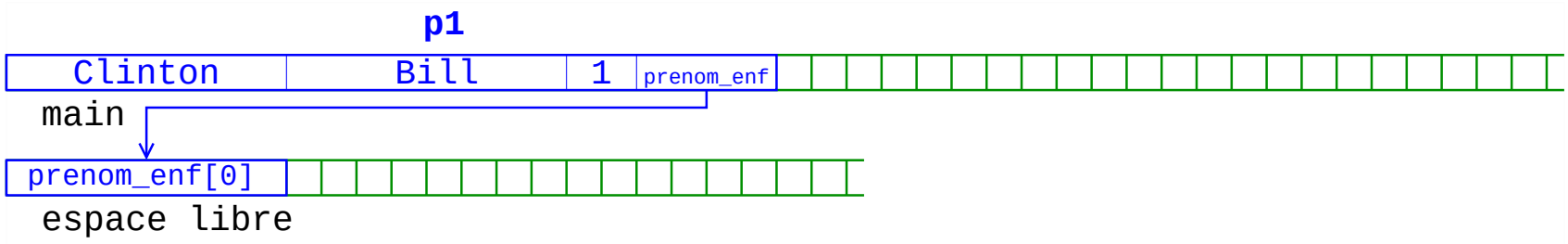
```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```



Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

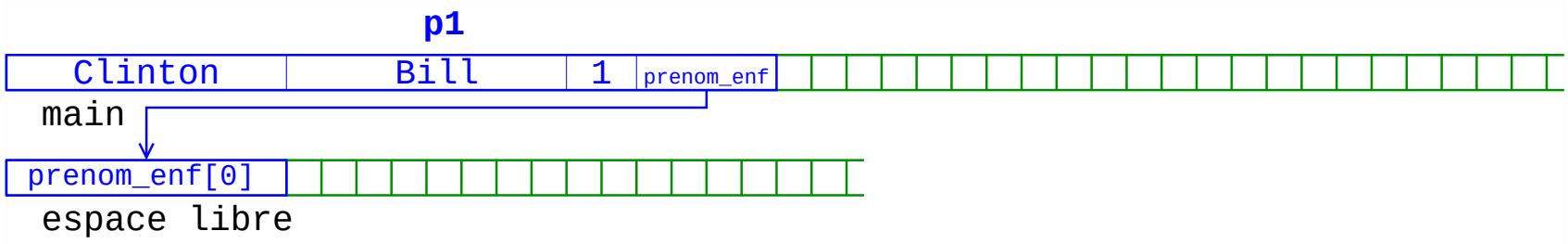
```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```



Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

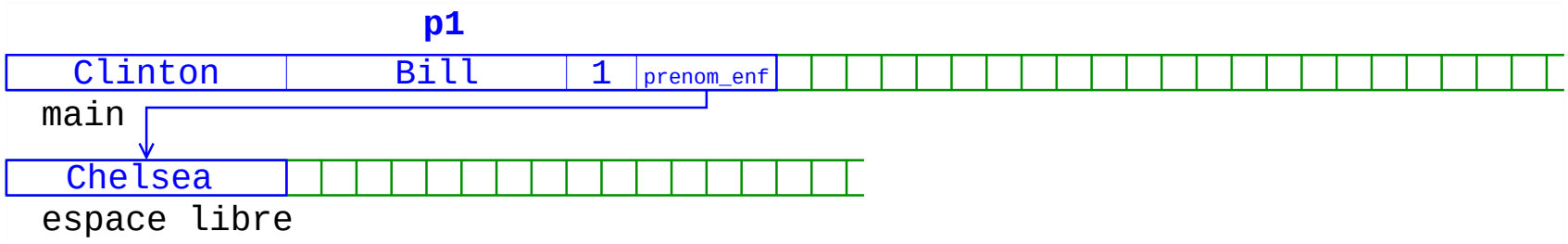
```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```



Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

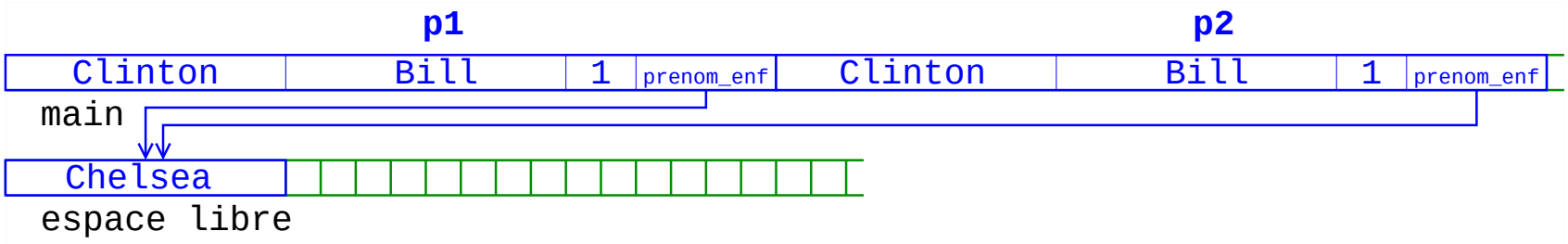
```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```



Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

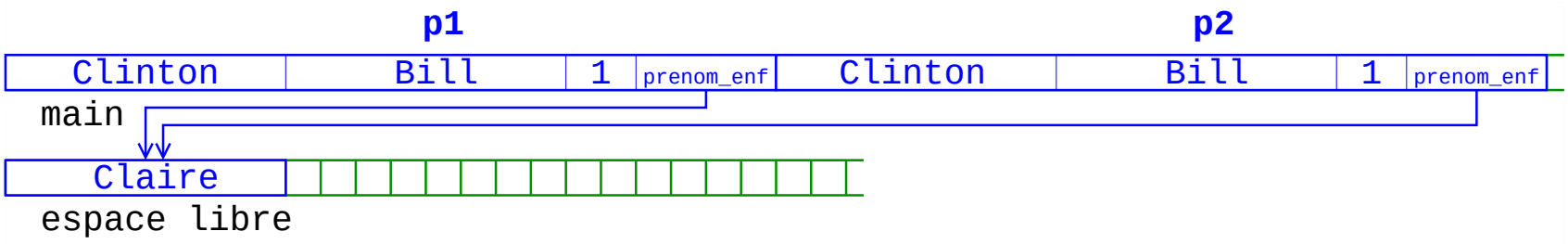
```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```



Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

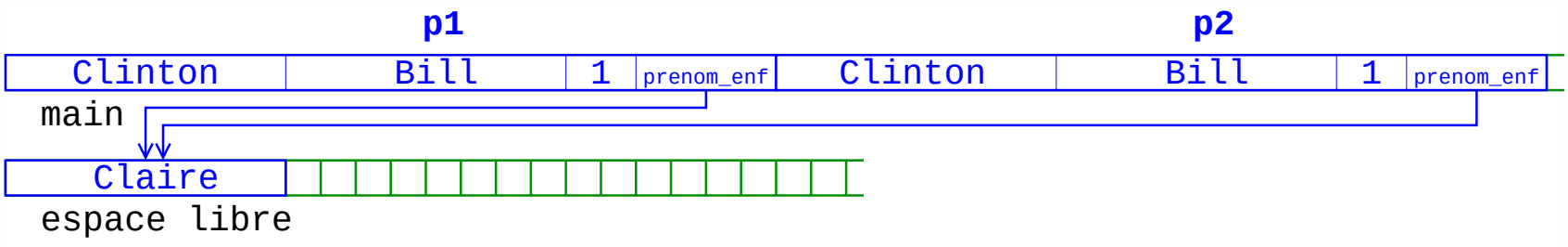
```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```



Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 0 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```



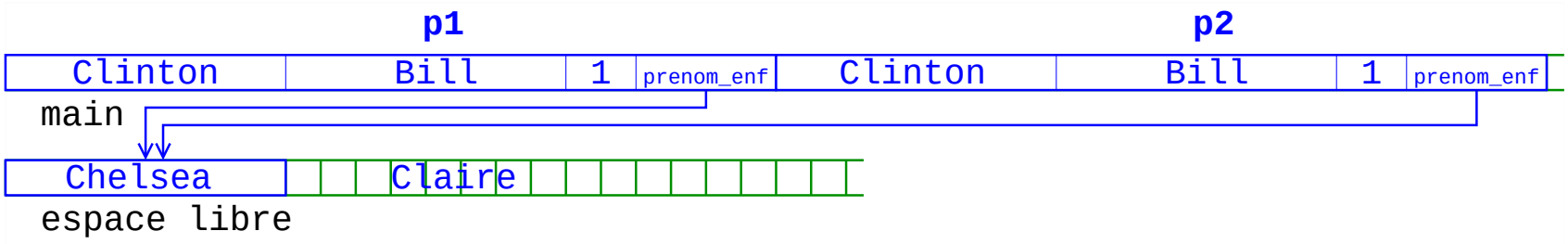
$\longrightarrow$ pas de copie du tableau dynamique dans l'espace libre

$\longrightarrow$ `p1` et `p2` réfèrent à des mêmes données

Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 1 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```



Les structures et les tableaux dynamiques

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  int Nenf;
  char ( *prenom_enf )[ 50 ];
};

int main()
{
  personne p1;
  registration( p1 );
  personne p2 = p1;
  strcpy( p2.prenom_enf[ 1 ], "Claire");
  cout << p1.prenom_enf[ 0 ] << endl;
}
```

```
void registration( personne &p )
{
  cout << "nom et prenom: ";
  cin >> p.nom >> p.prenom;
  cout << "nombre d'enfants: ";
  cin >> p.Nenf;
  p.prenom_enf = new char[ p.Nenf ][50];
  for ( int i = 0; i < p.Nenf; i++ )
  {
    cout << "prenom de l'enfant "
         << i + 1 << ": ";
    cin >> p.prenom_enf[ i ];
  }
}
```

Exécution:

```
nom et prenom: Clinton
Bill
nombre d'enfants: 1
prenom de l'enfant 1: Chelsea
Segmentation fault
```

Les fonctions de membre des structures

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;

  void registration()
  {
    cout << "nom et prenom:";
    cin >> nom >> prenom;
    cout << "genre(m/f/x):";
    cin >> genre;
    lire( naiss );
  }
};
```

```
void registration( personne &p )
{
  cout << "nom et prenom:";
  cin >> p.nom >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}

void affichage( const personne &p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " "
       << p.nom << endl;
}
```

Aussi des fonctions peuvent être des membres des structures.

Les fonctions de membre des structures

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;

  void registration()
  {
    cout << "nom et prenom:";
    cin >> nom >> prenom;
    cout << "genre(m/f/x):";
    cin >> genre;
    lire( naiss );
  }

  void affichage() const
  {
    if ( genre == 'f' )
      cout << "Mme ";
    if ( genre == 'm' )
      cout << "M. ";
    cout << prenom << " "
         << nom << endl;
  }
};
```

```
void registration( personne &p )
{
  cout << "nom et prenom:";
  cin >> p.nom >> p.prenom;
  cout << "genre(m/f/x):";
  cin >> p.genre;
  lire( p.naiss );
}

void affichage( const personne &p )
{
  if ( p.genre == 'f' )
    cout << "Mme ";
  if ( p.genre == 'm' )
    cout << "M. ";
  cout << p.prenom << " "
       << p.nom << endl;
}
```

Les fonctions de membre des structures

```
struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;

  void registration();

  void affichage() const;
};
```

```
void personne::registration()
{
  cout << "nom et prenom:";
  cin >> nom >> prenom;
  cout << "genre(m/f/x):";
  cin >> genre;
  lire( naiss );
}

void personne::affichage() const
{
  if ( genre == 'f' )
    cout << "Mme ";
  if ( genre == 'm' )
    cout << "M. ";
  cout << prenom << " "
       << nom << endl;
}
```

$\longrightarrow$ déclaration et définition des deux fonctions de membre `registration()` et `affichage()`

Les fonctions de membre des structures

Le contenu de `personne.h`

```
#include "date.h"
using namespace std;

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;

  void registration();

  void affichage() const;
};
```

Le contenu de `personne.cpp`

```
#include <iostream>
#include "personne.h"

void personne::registration()
{
  cout << "nom et prenom:";
  cin >> nom >> prenom;
  cout << "genre(m/f/x):";
  cin >> genre;
  lire( naiss );
}

void personne::affichage() const
{
  if ( genre == 'f' )
    cout << "Mme ";
  if ( genre == 'm' )
    cout << "M. ";
  cout << prenom << " "
       << nom << endl;
}
```

Les fonctions de membre des structures

Le contenu de `personne.h`

```
#include "date.h"
using namespace std;

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;

  void registration();

  void affichage() const;
};
```

Le contenu de `personne.cpp`

```
#include <iostream>
#include "personne.h"

void personne::registration()
{
  cout << "nom et prenom:";
  cin >> nom >> prenom;
  cout << "genre(m/f/x):";
  cin >> genre;
  naiss.lecture();
}

void personne::affichage() const
{
  if ( genre == 'f' )
    cout << "Mme ";
  if ( genre == 'm' )
    cout << "M. ";
  cout << prenom << " "
       << nom << endl;
}
```

Les fonctions de membre des structures

Le contenu de `date.h`

```
struct date
{
  int j;
  int m;
  int a;

  void lecture();
  void affichage() const;
};
```

Le contenu de `date.cpp`

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

void date::lecture()
{
  cout << "rentrez votre ";
  cout << "date de naissance:" << endl;
  cout << "jour:" << endl;
  cin >> j;
  cout << "mois:" << endl;
  cin >> m;
  cout << "an:" << endl;
  cin >> a;
}

void date::affichage() const
{
  cout << j << "/" << m
       << "/" << a << endl;
}
```

Les fonctions de membre des structures

Le contenu de `personne.h`

```
#include "date.h"
using namespace std;

struct personne
{
  char nom[ 50 ];
  char prenom[ 50 ];
  char genre;
  date naiss;

  void registration();

  void affichage() const;
};
```

Le contenu de `prog.cpp`

```
#include <iostream>
#include "personne.h"

int main()
{
  personne p1;
  p1.registration();
  personne p2 = p1;
  p2.genre = 'f';
  p2.affichage();
}
```

Les fonctions de membre des structures

```
struct ⟨stype⟩
{
   ⟨type1⟩ ⟨nom1⟩;
   ⟨type2⟩ ⟨nom2⟩;
   ...
   ⟨ftype⟩ ⟨fnom⟩( ⟨type_a⟩ ⟨nom_a⟩, ...);
};
```

$\longrightarrow$ déclaration de la fonction ⟨fnom⟩ comme membre de la structure ⟨stype⟩

Les fonctions de membre des structures

```
struct ⟨stype⟩
{
   ⟨type1⟩ ⟨nom1⟩;
   ⟨type2⟩ ⟨nom2⟩;
   ...
   ⟨ftype⟩ ⟨fnom⟩( ⟨type_a⟩ ⟨nom_a⟩, ...);
};

⟨ftype⟩ ⟨stype⟩::⟨fnom⟩( ⟨type_a⟩ ⟨nom_a⟩, ...)
{
   ...
}
```

$\longrightarrow$ définition “extérieure” de la fonction ⟨fnom⟩:

⟨stype⟩::⟨fnom⟩ signifie l'élément ⟨fnom⟩ de la structure ⟨stype⟩

Les fonctions de membre des structures

```
struct ⟨stype⟩
{
   ⟨type1⟩ ⟨nom1⟩;
   ⟨type2⟩ ⟨nom2⟩;
   ...
   ⟨ftype⟩ ⟨fnom⟩( ⟨type_a⟩ ⟨nom_a⟩, ...)
   {
      ...
   }
};
```

$\longrightarrow$ définition “intérieure” de la fonction ⟨fnom⟩

Les fonctions de membre des structures

```
struct ⟨stype⟩
{
   ⟨type1⟩ ⟨nom1⟩;
   ⟨type2⟩ ⟨nom2⟩;
   ...
   ⟨ftype⟩ ⟨fnom⟩( ⟨type_a⟩ ⟨nom_a⟩, ...);
};
```

Appel: ⟨snom⟩.⟨fnom⟩(⟨nom_a⟩, ...)
où ⟨snom⟩ est une variable du type ⟨stype⟩

C'est équivalent à l'appel ⟨fnom2⟩(⟨snom⟩, ⟨nom_a⟩, ...)
où ⟨fnom2⟩ est déclarée comme

⟨ftype⟩ ⟨fnom2⟩(⟨stype⟩ &⟨snom⟩, ⟨type_a⟩ ⟨nom_a⟩, ...);

(et, dans sa définition, ⟨nom1⟩ remplacé par ⟨snom⟩.⟨nom1⟩,...)

Les fonctions de membre des structures

```
struct ⟨stype⟩
{
   ⟨type1⟩ ⟨nom1⟩;
   ⟨type2⟩ ⟨nom2⟩;
   ...
   ⟨ftype⟩ ⟨fnom⟩( ⟨type_a⟩ ⟨nom_a⟩, ...) const;
};
```

$\longrightarrow$ la fonction ⟨fnom⟩ ne peut pas modifier
les variables ⟨nom1⟩, ⟨nom2⟩, ... de ⟨snom⟩

Les fonctions de membre des structures

```
struct ⟨stype⟩
{
   ⟨type1⟩ ⟨nom1⟩;
   ⟨type2⟩ ⟨nom2⟩;
   ...
   ⟨ftype⟩ ⟨fnom⟩( ⟨type_a⟩ ⟨nom_a⟩, ...) const;
};

⟨ftype⟩ ⟨stype⟩::⟨fnom⟩( ⟨type_a⟩ ⟨nom_a⟩, ...) const
{
   ...
}
```

$\longrightarrow$ ne pas oublier le `const` dans la définition !

9.2 Les classes

Les classes sont des structures avec des secteurs publics (`public`) et privés (`private`).

```
class date
{
public:

  int j;
  int m;
  int a;

  void lecture();
  void affichage() const;
};
```

est équivalent à

```
struct date
{
  int j;
  int m;
  int a;

  void lecture();
  void affichage() const;
};
```

$\longrightarrow$ tous les éléments de la classe `date` sont accessibles par l'extérieur

9.2 Les classes

Les classes sont des structures avec des secteurs publics (`public`) et privés (`private`).

```
class date
{
private:

  int j;
  int m;
  int a;

  void lecture();
  void affichage() const;
};
```

est équivalent à

```
class date
{
  int j;
  int m;
  int a;

  void lecture();
  void affichage() const;
};
```

$\longrightarrow$ aucun élément de la classe `date` n'est accessible par l'extérieur

(pas d'intérêt pratique . . .)

9.2 Les classes

Les classes sont des structures avec des secteurs publics (`public`) et privés (`private`).

```
class date
{
private:

  int j;
  int m;
  int a;

public:

  void lecture();
  void affichage() const;
};
```

est équivalent à

```
class date
{
  int j;
  int m;
  int a;

public:

  void lecture();
  void affichage() const;
};
```

⟶ `lecture` et `affichage` sont accessibles (publiques) et `j`, `m` et `a` sont inaccessibles (privés) par l'extérieur

9.2 Les classes

Les classes sont des structures avec des secteurs publics (`public`) et privés (`private`).

```
class date
{
private:

  int j;
  int m;
  int a;

public:

  void lecture();
  void affichage() const;
};
```

est équivalent à

```
class date
{
  int j;
  int m;
  int a;

public:

  void lecture();
  void affichage() const;
};
```

$\longrightarrow$ les variables `j`, `m` et `a` peuvent être accédées uniquement par les fonctions de membre `lecture` et `affichage`

9.2 Les classes

Les classes sont des structures avec des secteurs publics (`public`) et privés (`private`).

Typiquement, les variables de membre d'une classe sont déclarées privées et les fonctions de membres publiques.

Ceci permet de protéger les variables d'une classe contre une manipulation imprudente par l'utilisateur et de définir leur accès depuis l'extérieur par des fonctions de membres publiques.

Analogie avec un téléviseur:

$\rightarrow$ les boutons sur la télécommande correspondent aux fonctions puliques

$\rightarrow$ l'électronique à l'intérieur du téléviseur correspond à ses variables internes

Exemple: gestion sécurisée d'un tableau dynamique

```
class tableau
{
  int N;
  double *tab;

public:

  void construction( int N_ );

  void destruction();

  void set_element( int i, double x );

  double element( int i );
};

void tableau::construction( int N_ )
{
  N = N_;
  tab = new double[ N ];
}
```

Exemple: gestion sécurisée d'un tableau dynamique

```
class tableau
{
  int N;
  double *tab;

public:

  void construction( int N_ );

  void destruction();

  void set_element( int i, double x );

  double element( int i );
};

void tableau::destruction()
{
  delete[] tab;
}
```

Exemple: gestion sécurisée d'un tableau dynamique

```
class tableau
{
  int N;
  double *tab;

public:

  void construction( int N_ );

  void destruction();

  void set_element( int i, double x );

  double element( int i );
};

void tableau::set_element( int i, double x )
{
  if ( ( i >= 0 ) && ( i < N ) )
    tab[ i ] = x;
}
```

Exemple: gestion sécurisée d'un tableau dynamique

```
class tableau
{
  int N;
  double *tab;

public:

  void construction( int N_ );

  void destruction();

  void set_element( int i, double x );

  double element( int i );
};
```

```
double tableau::element( int i )
{
  if ( ( i >= 0 ) && ( i < N ) )
    return ( tab[ i ] );
  return 0;
}
```

Exemple: gestion sécurisée d'un tableau dynamique

```
class tableau
{
  int N;
  double *tab;

public:

  void construction( int N_ );

  void destruction();

  void set_element( int i, double x );

  double element( int i ) const;
};
```

```
double tableau::element( int i ) const
{
  if ( ( i >= 0 ) && ( i < N ) )
    return ( tab[ i ] );
  return 0;
}
```

Exemple: gestion sécurisée d'un tableau dynamique

```
class tableau
{
  int N;
  double *tab;

public:

  void construction( int N_ );

  void destruction();

  void set_element( int i, double x );

  double element( int i ) const;
};
```

$\longrightarrow$ tableau dynamique avec toutes fonctionnalitées "correctes"

$\longrightarrow$ pas de moyen de manipuler `tab` de manière pernicieuse depuis l'extérieur de la classe

9.2 Les classes

```
class tableau
{
  int N;
  double *tab;

public:

  void construction( int N_ )
  {
    N = N_;
    tab = new double[ N ];
  }

  void destruction()
  {
    delete[] tab;
  }

  void set_element( int i, double x )
  {
    if ( ( i >= 0 ) && ( i < N ) )
      tab[ i ] = x;
  }

  double element( int i ) const
  {
    if ( ( i >= 0 ) && ( i < N ) )
      return ( tab[ i ] );
    return 0;
  }
};
```

9.2 Les classes

```
class tableau
{
  int N;
  double *tab;

public:

  void construction( int N_ );

  void destruction();

  void set_element( int i, double x );

  double element( int i ) const;
};
```

```
int main()
{
  int N;
  cout << "taille du tableau: ";
  cin >> N;
  tableau t;
  t.construction( N );

  double x;
  cout << "les elements:" << endl;
  for ( int i = 0; i < N; i++ )
  {
    cin >> x;
    t.set_element( i, x );
  }
  cout << t.element( 2 ) << endl;
}
```

9.2 Les classes

```
class tableau
{
  int N;
  double *tab;

public:

  void construction( int N_ );

  void destruction();

  void set_element( int i, double x );

  double element( int i ) const;
};
```

```
int main()
{
  int N;
  cout << "taille du tableau: ";
  cin >> N;
  tableau t;
  t.construction( N );

  double x;
  cout << "les elements:" << endl;
  for ( int i = 0; i < N; i++ )
  {
    cin >> x;
    t.set_element( i, x );
  }
  t.tab++;
  cout << t.element( 2 ) << endl;
}
```

$\longrightarrow$ `erreur: 'double* tableau::tab' is private`

9.2 Les classes

```
class tableau
{
  int N;
  double *tab;

public:

  void construction( int N_ );

  void destruction();

  void set_element( int i, double x );

  double element( int i ) const;
};
```

```
int main()
{
  int N;
  cout << "taille du tableau: ";
  cin >> N;
  tableau t;

  double x;
  cout << "les elements:" << endl;
  for ( int i = 0; i < N; i++ )
  {
    cin >> x;
    t.set_element( i, x );
  }
  cout << t.element( 2 ) << endl;
}
```

Exécution:

```
taille du tableau: 2
les elements:
2
3
nan
```

9.2 Les classes

```
class tableau
{
  int N;
  double *tab;

public:

  void construction( int N_ );

  void destruction();

  void set_element( int i, double x );

  double element( int i ) const;
};
```

```
void routine()
{
  int N;
  cout << "taille du tableau: ";
  cin >> N;
  tableau t;
  t.construction( N );

  double x;
  cout << "les elements:" << endl;
  for ( int i = 0; i < N; i++ )
  {
    cin >> x;
    t.set_element( i, x );
  }
  cout << t.element( 2 ) << endl;
  t.destruction();
}
```

$\longrightarrow$ ne pas oublier la construction (au début de son utilisation) et la déstruction (à la fin) de l'objet `t` du type `tableau` !

Construction et déstruction des classes

```
class tableau
{
  int N;
  double *tab;

public:

  void construction( int N_ )
  {
    N = N_;
    tab = new double[ N ];
  }

  void destruction()
  {
    delete[] tab;
  }

  void set_element( int i, double x );

  double element( int i ) const;
};
```

Comment assurer que la routine `construction` soit appelée lors de chaque déclaration d'un objet du type `tableau` ?

(et `destruction` soit appelée à la fin de son utilisation)

Construction et déstruction des classes

```
class tableau
{
  int N;
  double *tab;

public:

  void construction( int N_ )
  {
    N = N_;
    tab = new double[ N ];
  }

  void destruction()
  {
    delete[] tab;
  }

  void set_element( int i, double x );

  double element( int i ) const;
};
```

Comment assurer que la routine `construction` soit appelée lors de chaque déclaration d'un objet du type `tableau` ?

$\longrightarrow$ utiliser des constructeurs et des déstructeurs

Construction et déstruction des classes

```
class tableau
{
  int N;
  double *tab;

public:

  tableau( int N_ )
  {
    N = N_;
    tab = new double[ N ];
  }

  ~tableau()
  {
    delete[] tab;
  }

  void set_element( int i, double x );

  double element( int i ) const;
};
```

Construction et déstruction des classes

```
class tableau
{
  int N;
  double *tab;

public:

  tableau( int N_ );

  ~tableau();

  void set_element( int i, double x );

  double element( int i ) const;
};

tableau::tableau( int N_ )
{
  N = N_;
  tab = new double[ N ];
}

tableau::~tableau()
{
  delete[] tab;
}
```

Construction et déstruction des classes

```
class tableau
{
  int N;
  double *tab;

public:

  tableau( int N_ )
  {
    N = N_;
    tab = new double[ N ];
  }

  ~tableau()
  {
    delete[] tab;
  }

  void set_element( int i, double x );

  double element( int i ) const;
};
```

```
int main()
{
  int N;
  cout << "taille du tableau: ";
  cin >> N;
  tableau t( N );

  double x;
  cout << "les elements:" << endl;
  for ( int i = 0; i < N; i++ )
  {
    cin >> x;
    t.set_element( i, x );
  }
  cout << t.element( 2 ) << endl;
}
```

$\longrightarrow$ `tableau t( N )` définit l'objet `t` et appelle la fonction de membre `tableau( int )` ($\rightarrow$ le "constructeur")

Construction et déstruction des classes

```
class tableau
{
  int N;
  double *tab;

public:

  tableau( int N_ )
  {
    N = N_;
    tab = new double[ N ];
  }

  ~tableau()
  {
    delete[] tab;
  }

  void set_element( int i, double x );

  double element( int i ) const;
};
```

```
int main()
{
  int N;
  cout << "taille du tableau: ";
  cin >> N;
  tableau t( N );

  double x;
  cout << "les elements:" << endl;
  for ( int i = 0; i < N; i++ )
  {
    cin >> x;
    t.set_element( i, x );
  }
  cout << t.element( 2 ) << endl;
}
```

$\longrightarrow$ la fonction `~tableau()` est automatiquement appelée à la fin du bloc où `t` a été défini ($\rightarrow$ le “déstructeur”)

Construction et déstruction des classes

```
class tableau
{
  int N;
  double *tab;

public:

  tableau( int N_ )
  {
    N = N_;
    tab = new double[ N ];
  }

  ~tableau()
  {
    delete[] tab;
  }

  void set_element( int i, double x );

  double element( int i ) const;
};
```

```
int main()
{
  int N;
  cout << "taille du tableau: ";
  cin >> N;
  tableau t( N );

  double x;
  cout << "les elements:" << endl;
  for ( int i = 0; i < N; i++ )
  {
    cin >> x;
    t.set_element( i, x );
  }
  cout << t.element( 2 ) << endl;
}
```

$\longrightarrow$ comme ça, il est exigé de spécifier la taille du tableau lorsque l'on définit un objet du type `tableau`

Construction et déstruction des classes

```
class tableau
{
  int N;
  double *tab;

public:

  tableau( int N_ )
  {
    N = N_;
    tab = new double[ N ];
  }

  ~tableau()
  {
    delete[] tab;
  }

  void set_element( int i, double x );

  double element( int i ) const;
};
```

```
int main()
{
  int N;
  cout << "taille du tableau: ";
  cin >> N;
  tableau t;

  double x;
  cout << "les elements:" << endl;
  for ( int i = 0; i < N; i++ )
  {
    cin >> x;
    t.set_element( i, x );
  }
  cout << t.element( 2 ) << endl;
}
```

$\longrightarrow$ erreur:

```
no matching function for call to 'tableau::tableau()'
```

Construction et déstruction des classes

```
class tableau
{
  int N;
  double *tab;

public:

  tableau( int N_ )
  {
    N = N_;
    tab = new double[ N ];
  }

  tableau()
  {
    N = 0;
    tab = new double[ N ];
  }

  ~tableau()
  {
    delete[] tab;
  }

  void set_element( int i, double x );

  double element( int i ) const;
};
```

```
int main()
{
  int N;
  cout << "taille du tableau: ";
  cin >> N;
  tableau t;

  double x;
  cout << "les elements:" << endl;
  for ( int i = 0; i < N; i++ )
  {
    cin >> x;
    t.set_element( i, x );
  }
  cout << t.element( 2 ) << endl;
}
```

$\longrightarrow$ `tableau t;` appelle le constructeur `tableau()`

Construction et déstruction des classes

```
class tableau
{
  int N;
  double *tab;

public:

  tableau( int N_ )
  {
    N = N_;
    tab = new double[ N ];
  }

  ~tableau()
  {
    delete[] tab;
  }

  void set_element( int i, double x );

  double element( int i ) const;
};
```

```
int main()
{
  int N;
  cout << "taille du tableau: ";
  cin >> N;
  tableau t( N );

  double x;
  cout << "les elements:" << endl;
  for ( int i = 0; i < N; i++ )
  {
    cin >> x;
    t.set_element( i, x );
  }
  cout << t.element( 2 ) << endl;

  tableau t2 = t;
  // pas de copie correcte...
  // t et t2 travaillent avec des donnees
  // identiques sur l'espace libre

  t2.set_element( 2, 0.3 );
  cout << t.element( 2 ) << endl;
}
```

Comment réaliser une copie correcte du tableau ?

$\longrightarrow$ définir un constructeur de copie :

```
tableau( const tableau& )
```

Construction et déstruction des classes

```
class tableau
{
  int N;
  double *tab;

public:

  tableau( int N_ )
  {
    N = N_;
    tab = new double[ N ];
  }

  tableau( const tableau &t )
  {
    N = t.N;
    tab = new double[ N ];
    for ( int i = 0; i < N; i++ )
      tab[ i ] = t.tab[ i ];
  }

  ~tableau()
  {
    delete[] tab;
  }

  void set_element( int i, double x );

  double element( int i ) const;
};
```

```
int main()
{
  int N;
  cout << "taille du tableau: ";
  cin >> N;
  tableau t( N );

  double x;
  cout << "les elements:" << endl;
  for ( int i = 0; i < N; i++ )
  {
    cin >> x;
    t.set_element( i, x );
  }
  cout << t.element( 2 ) << endl;

  tableau t2( t );
  // copie correcte!
  // t et t2 travaillent avec des donnees
  // non identiques sur l'espace libre

  t2.set_element( 2, 0.3 );
  cout << t.element( 2 ) << endl;
}
```

Construction et déstruction des classes

```
class tableau
{
  int N;
  double *tab;

public:

  tableau( int N_ )
  {
    N = N_;
    tab = new double[ N ];
  }

  tableau( const tableau &t )
  {
    N = t.N;
    tab = new double[ N ];
    for ( int i = 0; i < N; i++ )
      tab[ i ] = t.tab[ i ];
  }

  ~tableau()
  {
    delete[] tab;
  }

  void set_element( int i, double x );

  double element( int i ) const;
};
```

```
int main()
{
  int N;
  cout << "taille du tableau: ";
  cin >> N;
  tableau t( N );

  double x;
  cout << "les elements:" << endl;
  for ( int i = 0; i < N; i++ )
  {
    cin >> x;
    t.set_element( i, x );
  }
  cout << t.element( 2 ) << endl;

  tableau t2 = t;
  // copie correcte!
  // t et t2 travaillent avec des donnees
  // non identiques sur l'espace libre

  t2.set_element( 2, 0.3 );
  cout << t.element( 2 ) << endl;
}
```

9 Les structures et les classes