Éléments d'informatique – Cours 4. Compilation. Instruction de contrôle `for`.

Pierre Boudes

7 octobre 2009

Liens utiles

- ma page : `http://www-lipn.univ-paris13.fr/~boudes/`
- `http://www.siteduzero.com/` (chercher langage C)
- `http://www.developpez.com/` (chercher langage C)
- le cours de Anne Canteaut : `http://www-roc.inria.fr/secret/Anne.Canteaut/COURS_C/`
- le cours de Bernard Cassagne : `http://clips.imag.fr/commun/bernard.cassagne/Introduction_ANSI_C.html`
- le cours de Henri Garreta : `http://www.dil.univ-mrs.fr/~garreta/generique/index.html`
- codeblocks : `http://www.codeblocks.org/`
- ubuntu : `http://www.ubuntu-fr.org/`

Compilation

Les cinq grandes étapes de la compilation :

1. Analyse lexicale
2. Analyse syntaxique
3. Analyse sémantique
4. Génération du code
5. Édition de liens

Analyse lexicale

Analyse lexicale

Identifie les *lexèmes* (unités lexicales du langage). Les espaces sont inutiles (`3*x+1` ou `3 * x + 1`), sauf comme séparateurs (`int x`, `intx`).

Analyse lexicale

Analyse lexicale

Identifie les *lexèmes* (unités lexicales du langage). Les espaces sont inutiles (`3*x+1` ou `3 * x + 1`), sauf comme séparateurs (`int x`, `intx`).

Erreur lexicale :

code source `int x = @;`

compilation `error : stray '@' in program`

Analyse lexicale

Analyse lexicale

Identifie les *lexèmes* (unités lexicales du langage). Les espaces sont inutiles (`3*x+1` ou `3 * x + 1`), sauf comme séparateurs (`int x`, `intx`).

Erreur lexicale :

code source `int x = @ ;`

compilation `error : stray ‘@’ in program`

Erreur détectée uniquement au moment de l'analyse sémantique :

code source `intx = 0 ;`

compilation
```
error : ‘intx’ undeclared (first use in
        this function)
```

Analyse syntaxique

Analyse syntaxique

trouve la structure syntaxique, (arbre syntaxique), et teste l'appartenance au langage.

Analyse syntaxique

Analyse syntaxique

trouve la structure syntaxique, (arbre syntaxique), et teste l'appartenance au langage.

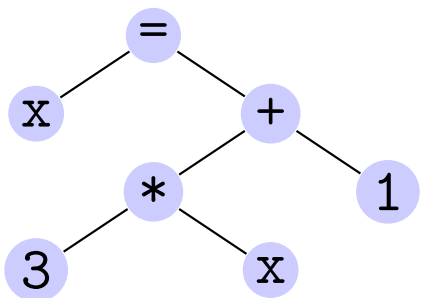
Exemple : dans l'expression `x = 3 * x + 1`, est-ce que la sous-suite `x + 1` correspond à une structure syntaxique ?

Analyse syntaxique

Analyse syntaxique

trouve la structure syntaxique, (arbre syntaxique), et teste l'appartenance au langage.

Exemple : dans l'expression `x = 3 * x + 1`, est-ce que la sous-suite `x + 1` correspond à une structure syntaxique ?

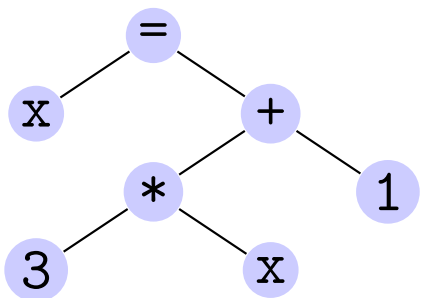


Analyse syntaxique

Analyse syntaxique

trouve la structure syntaxique, (arbre syntaxique), et teste l'appartenance au langage.

Exemple : dans l'expression `x = 3 * x + 1`, est-ce que la sous-suite `x + 1` correspond à une structure syntaxique ?



code source Un else sans if le précédant immédiatement (point-virgule mal placé ?)

compilation `error : expected expression before ‘else’`

Analyse sémantique

Analyse sémantique

trouver le sens des différentes actions voulues par le programmeur. Quelles sont les objets manipulés par le programme, quelles sont les propriétés de ces objets, quelles sont les actions du programme sur ces objets.

Analyse sémantique

Analyse sémantique

trouver le sens des différentes actions voulues par le programmeur. Quelles sont les objets manipulés par le programme, quelles sont les propriétés de ces objets, quelles sont les actions du programme sur ces objets.

Beaucoup d'erreurs peuvent apparaître durant cette phase : identificateur utilisé mais non déclaré (la réciproque génère un *warning* avec l'option `-Wall`), opération n'ayant aucun sens, etc.

code source variable `x` utilisée mais non déclarée

compilation
```
error : ‘x’ undeclared (first use in this
function)
```

Génération du code

Génération du code

encodage en assembleur, optimisations et allocations des registres, traduction en code objet.

Édition de liens

Édition de liens

le code objet des fonctions externes (bibliothèques) est ajouté à l'exécutable. Le point d'entrée dans le programme est choisi (`main`). Insertion de données de débogage (option `-g`).

Édition de liens

Édition de liens

le code objet des fonctions externes (bibliothèques) est ajouté à l'exécutable. Le point d'entrée dans le programme est choisi (`main`). Insertion de données de débogage (option `-g`).

code source Oublie de `stdio.h` et utilisation de `printf`.

compilation
```
warning : incompatible implicit declaration
of built-in function ‘printf’
```

Édition de liens

Édition de liens

le code objet des fonctions externes (bibliothèques) est ajouté à l'exécutable. Le point d'entrée dans le programme est choisi (`main`). Insertion de données de débogage (option `-g`).

code source Oublie de `stdio.h` et utilisation de `printf`.

compilation
```
warning : incompatible implicit declaration
of built-in function ‘printf’
```

code source Pas de fonction principale (`main`)

compilation
```
Undefined symbols : "_main", ...
```

Édition de liens

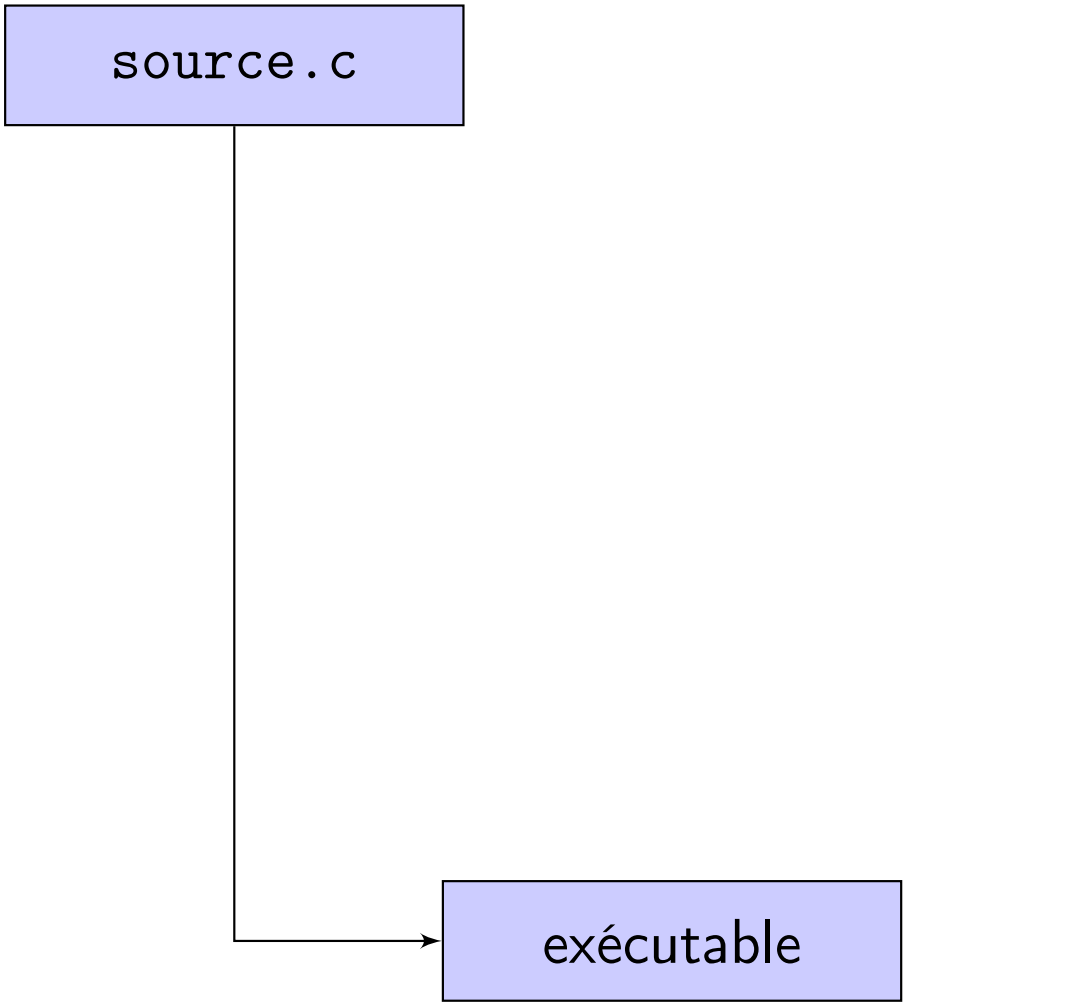
Édition de liens

le code objet des fonctions externes (bibliothèques) est ajouté à l'exécutable. Le point d'entrée dans le programme est choisi (`main`). Insertion de données de débogage (option `-g`).

code source	Oublie de `stdio.h` et utilisation de `printf`.
compilation	`warning : incompatible implicit declaration of built-in function ‘printf’`
code source	Pas de fonction principale (`main`)
compilation	`Undefined symbols : "_main", ...`

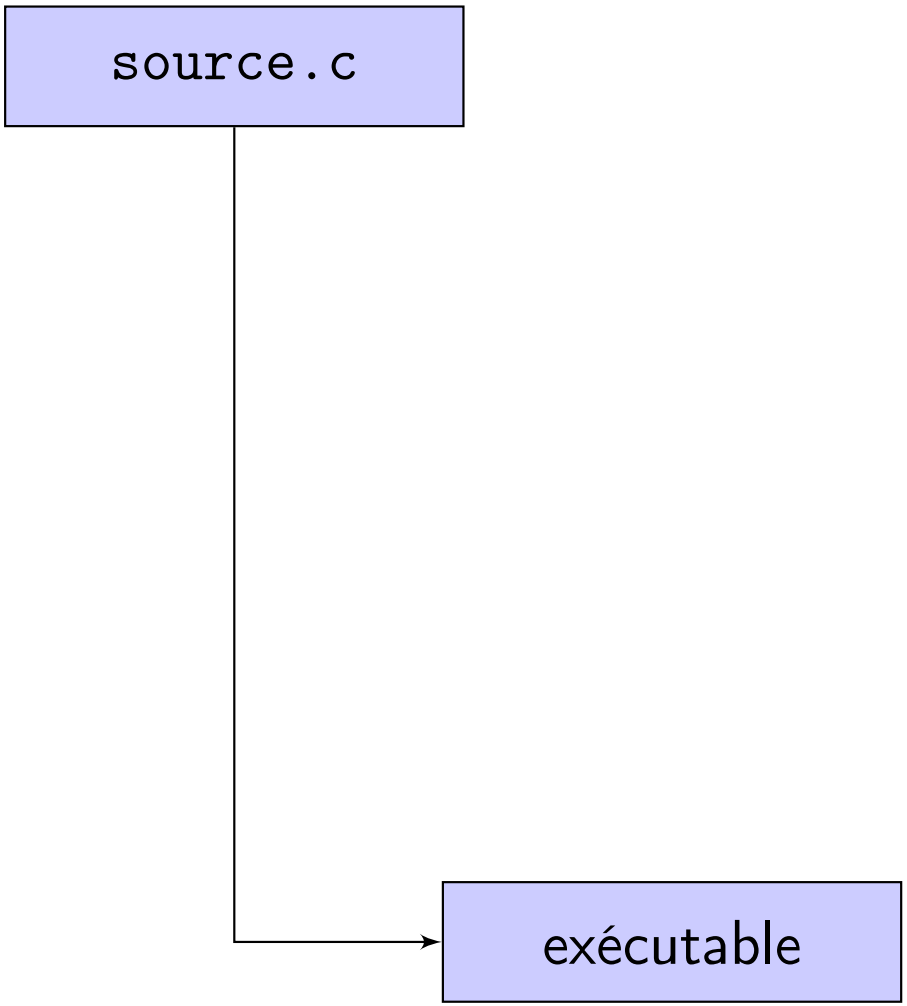
À votre avis : `Undefined symbols : "_prinft"` ?

La compilation en pratique



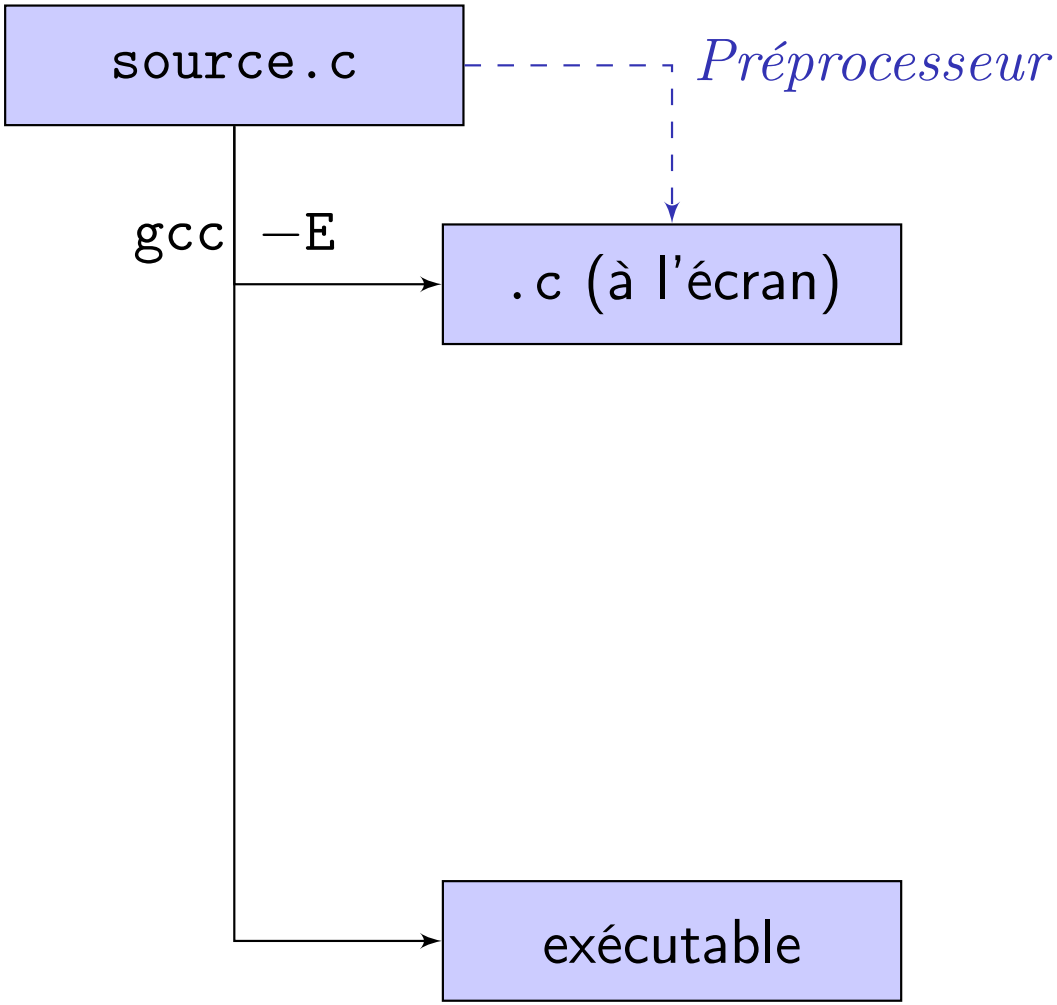

```
gcc        source.c -o nom
```

La compilation en pratique




```
gcc -Wall source.c -o nom
```

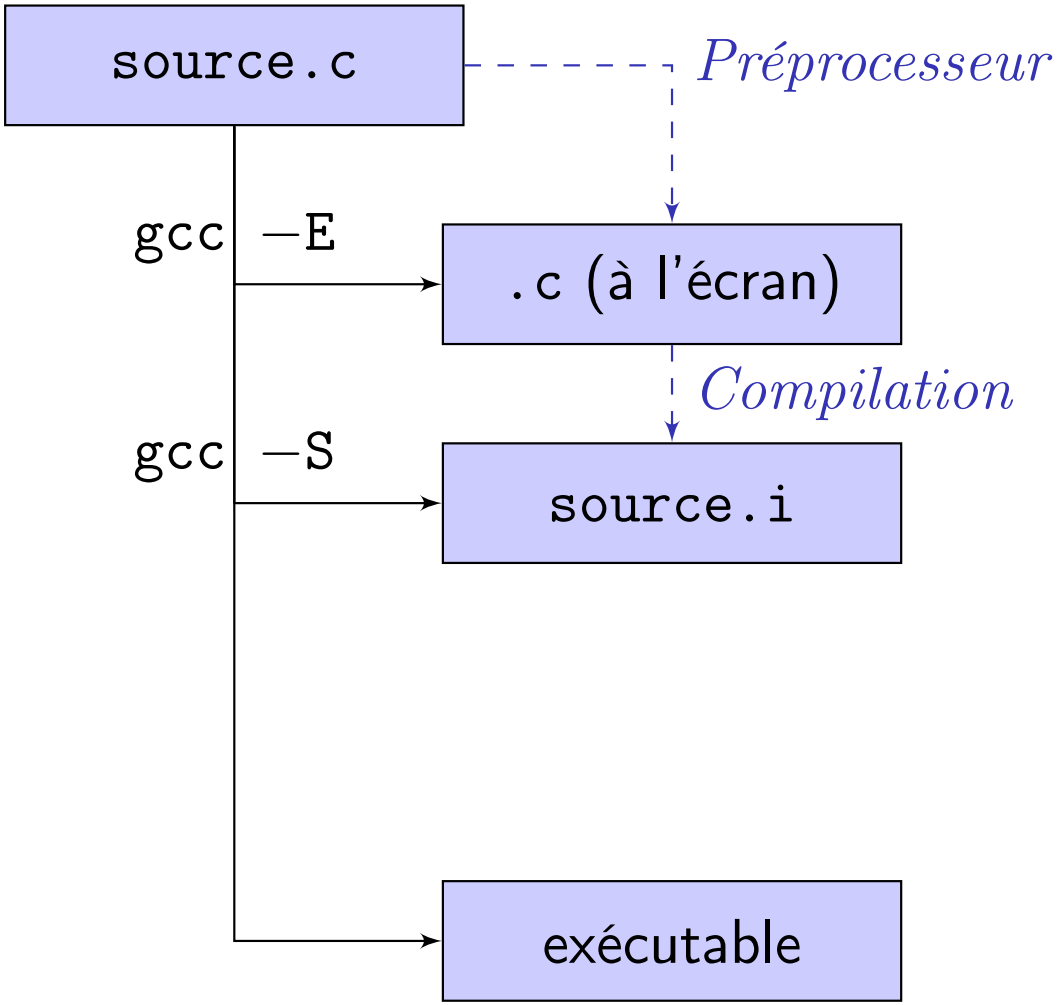
La compilation en pratique



Le préprocesseur enlève les commentaires et exécute les instructions commençant par un dièse : `#define` (rechercher-remplacer) et `#include` (insertion de fichier).

```
gcc −Wall source.c −o nom
```

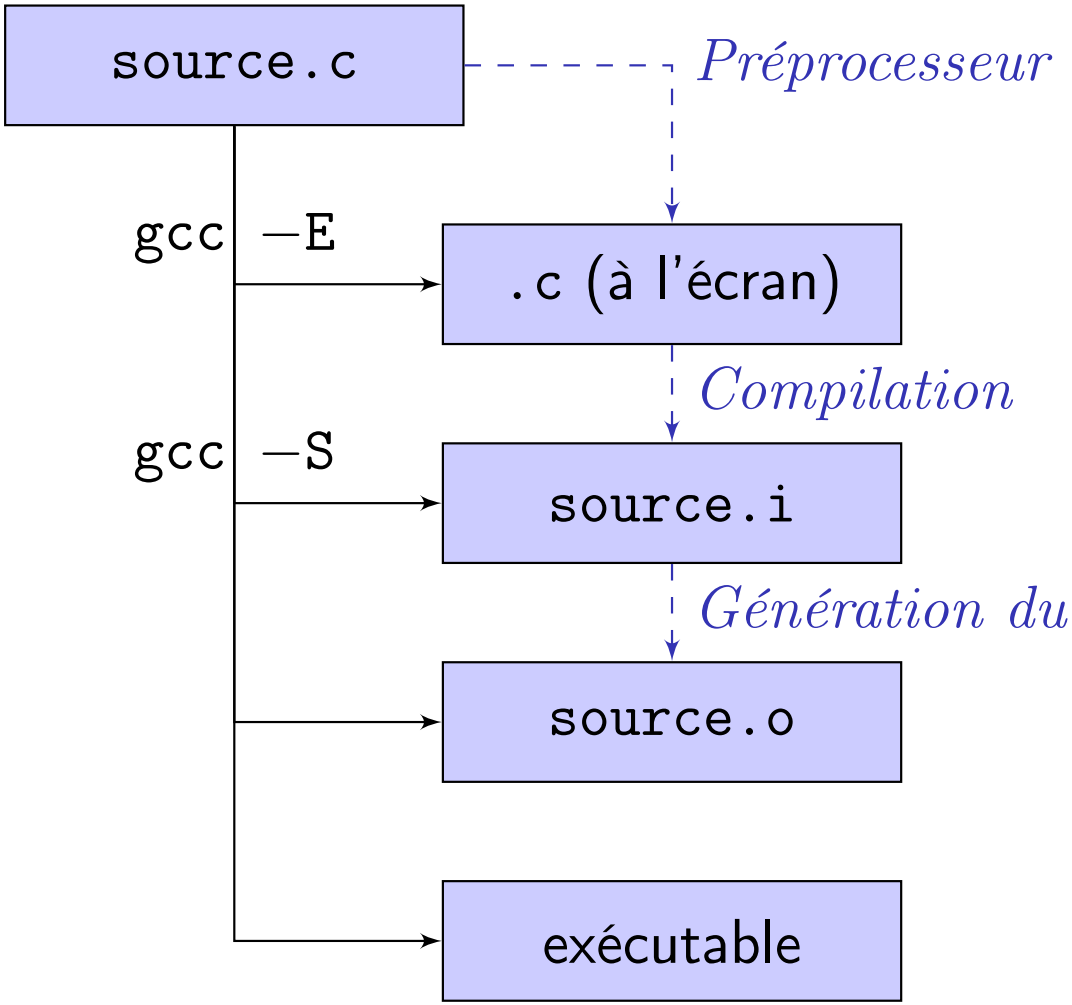
La compilation en pratique



Le préprocesseur enlève les commentaires et exécute les instructions commençant par un dièse : `#define` (rechercher-remplacer) et `#include` (insertion de fichier).

```
gcc −Wall source.c −o nom
```

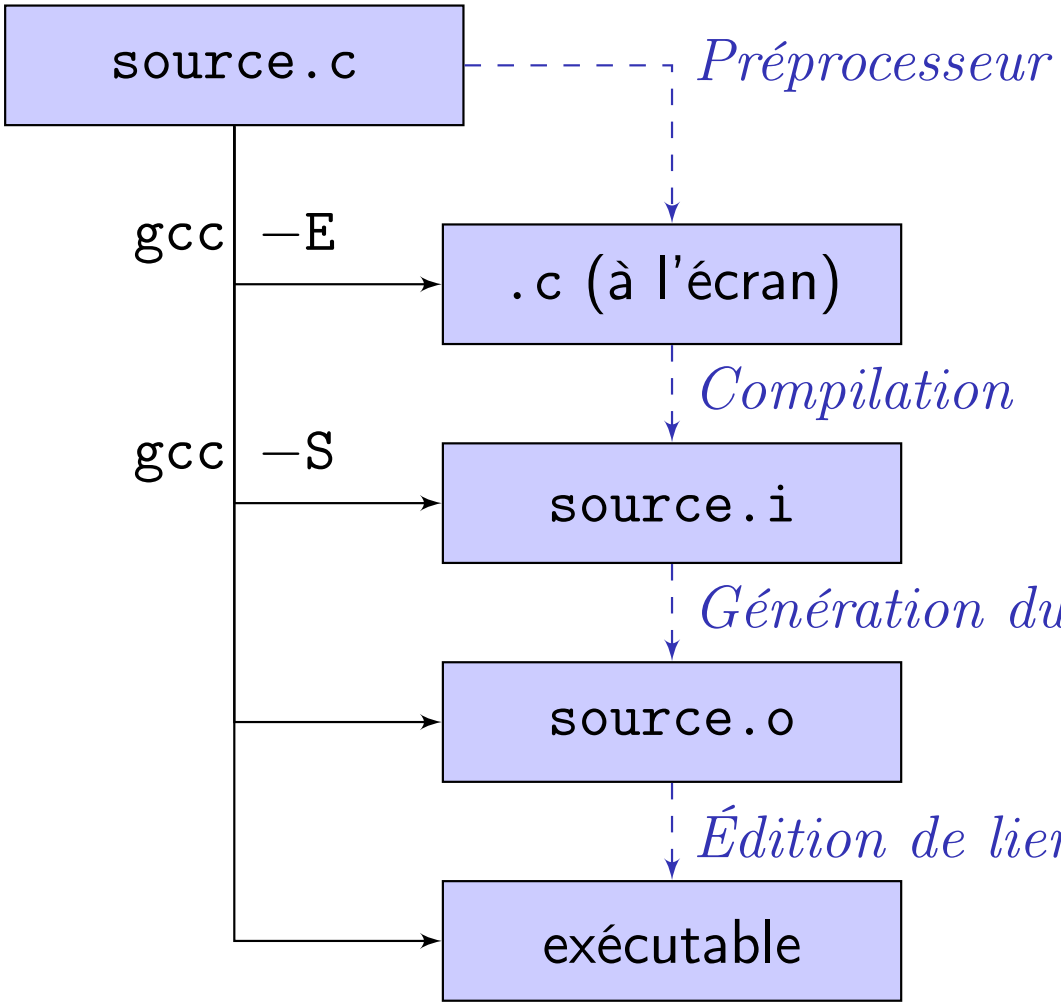
La compilation en pratique



Le préprocesseur enlève les commentaires et exécute les instructions commençant par un dièse : `#define` (rechercher-remplacer) et `#include` (insertion de fichier).

```
gcc −Wall source.c −o nom
```

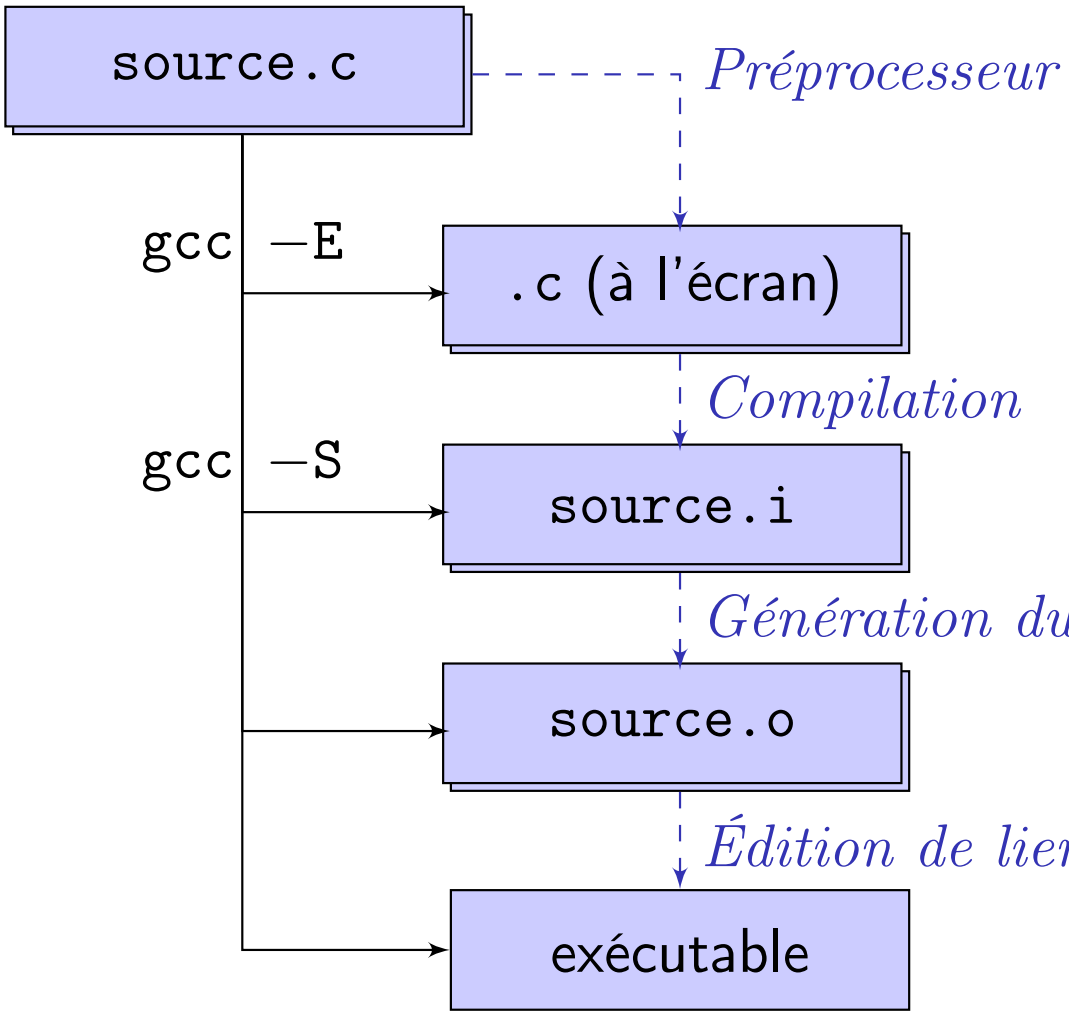
La compilation en pratique



Le préprocesseur enlève les commentaires et exécute les instructions commençant par un dièse : `#define` (rechercher-remplacer) et `#include` (insertion de fichier).

```
gcc −Wall source.c −o nom
```

La compilation en pratique



Le préprocesseur enlève les commentaires et exécute les instructions commençant par un dièse : `#define` (rechercher-remplacer) et `#include` (insertion de fichier).

```
gcc −Wall source.c −o nom
```

Programmation structurée : rappel

Programmer par blocs en les combinant de trois manières :

1. exécuter les blocs les uns à la suite des autres (séquence)
2. si une certaine condition est vraie, exécuter un bloc sinon en exécuter un autre (sélection)
3. recommencer l'exécution d'un bloc tant qu'une certaine condition est vraie (répétition).

Un bloc peut lui-même contenir une combinaison de blocs.

Programmation structurée : rappel

Programmer par blocs en les combinant de trois manières :

1. exécuter les blocs les uns à la suite des autres (séquence)
2. si une certaine condition est vraie, exécuter un bloc sinon en exécuter un autre (sélection)
3. recommencer l'exécution d'un bloc tant qu'une certaine condition est vraie (répétition).

Un bloc peut lui-même contenir une combinaison de blocs.
Aujourd'hui nous allons voir une première forme de répétition en C le `for`. Avant cela nous revenons sur la sélection (le `if` ou `if else`).

Rappels sur l'instruction de contrôle if

Syntaxe : `if (condition) { bloc1 } else { bloc2 }.`

Rappels sur l'instruction de contrôle if

Syntaxe : `if (condition) { bloc1 } else { bloc2 }.`

Code source

```
/* avant */
if (age < 18)
{
  permis = 0;
}
else
{
  permis = 1;
}
/* après */
```

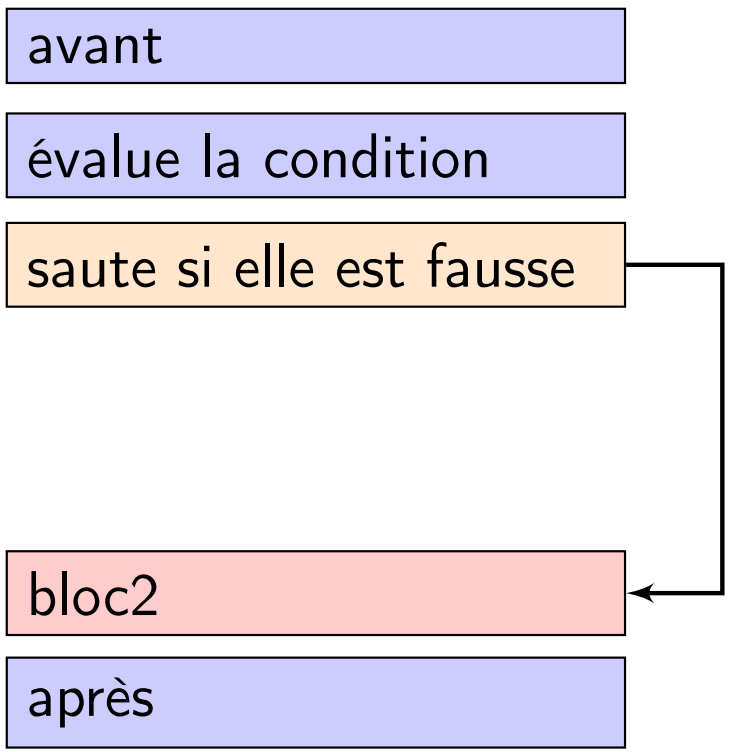
Rappels sur l'instruction de contrôle if

Syntaxe : `if (condition) { bloc1 } else { bloc2 }.`

Code source

```
/* avant */
if (age < 18)
{
  permis = 0;
}
else
{
  permis = 1;
}
/* après */
```

Schéma de traduction



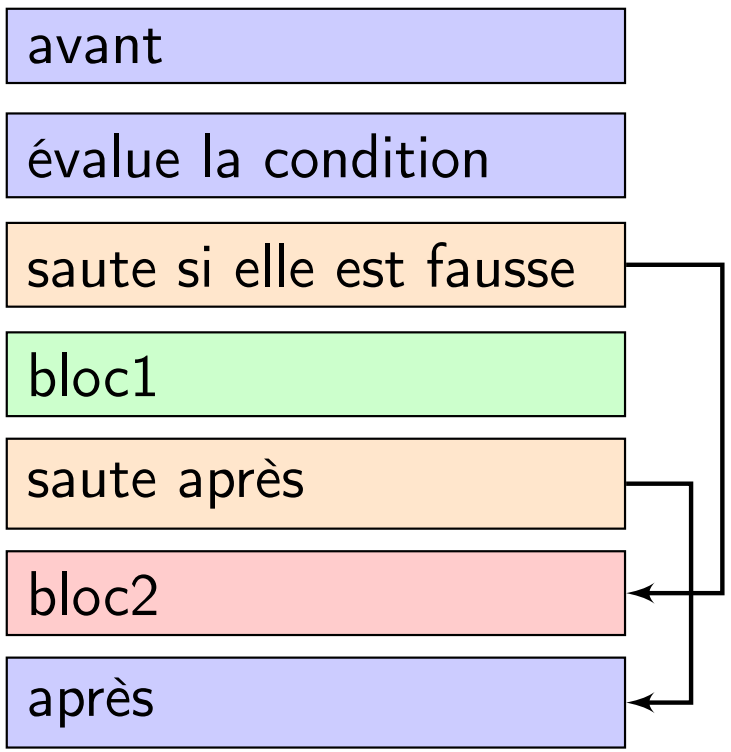
Rappels sur l'instruction de contrôle if

Syntaxe : `if (condition) { bloc1 } else { bloc2 }.`

Code source

```
/* avant */
if (age < 18)
{
  permis = 0;
}
else
{
  permis = 1;
}
/* après */
```

Schéma de traduction



L'instruction de contrôle for

Syntaxe :

```
for (instruct1 ; condition ; instruct2 ) { bloc }.
```

L'instruction de contrôle for

Syntaxe :

`for (`*instruct1*`;` *condition*`;` *instruct2*`) {` *bloc* `}`.

Code source

```
/* avant */
for (i = 0; i < 5; i = i + 1)
{
  printf("%d\n", i);
  ...
}
/* après */
```

La variable `i` est appelée variable de boucle, elle doit être préalablement déclarée comme toute autre variable.

L'instruction de contrôle for

Syntaxe :

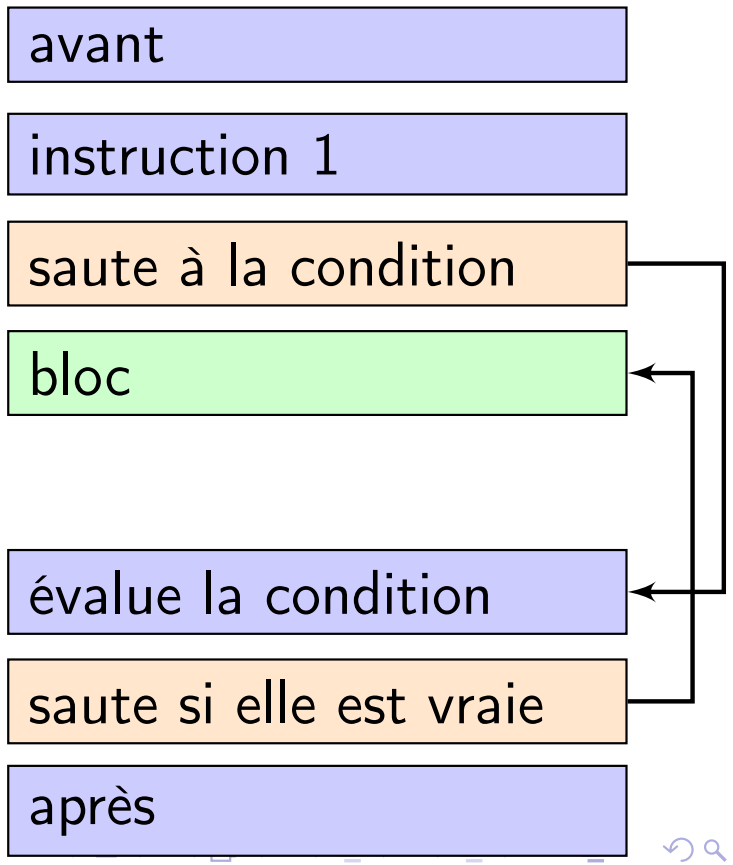
`for (`*instruct1*`;` *condition*`;` *instruct2*`) {` *bloc* `}`.

Code source

```
/* avant */
for (i = 0; i < 5; i = i + 1)
{
  printf("%d\n", i);
  ...
}
/* après */
```

La variable `i` est appelée variable de boucle, elle doit être préalablement déclarée comme toute autre variable.

Schéma de traduction



L'instruction de contrôle for

Syntaxe :

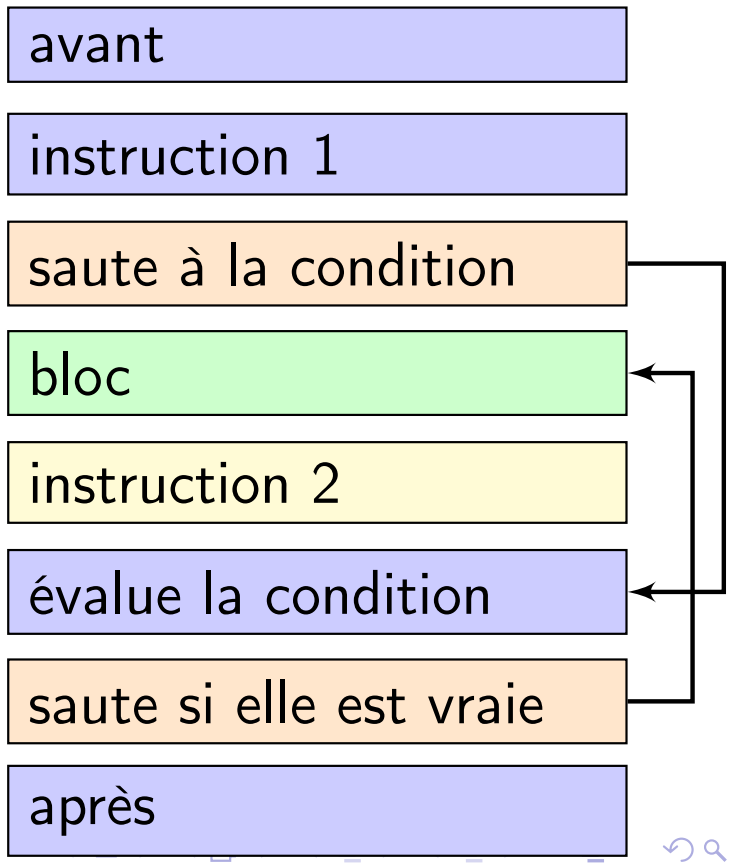
`for (`*instruct1*`;` *condition*`;` *instruct2*`) {` *bloc* `}`.

Code source

```
/* avant */
for (i = 0; i < 5; i = i + 1)
{
  printf("%d\n", i);
  ...
}
/* après */
```

La variable `i` est appelée variable de boucle, elle doit être préalablement déclarée comme toute autre variable.

Schéma de traduction



Démos