LE PRÉPROCESSEUR

2012-10-01

© Emmanuel Chieze
Département d'informatique, UQAM
INF3135

Plan

- La compilation en C
- La bibliothèque standard
- Les symboles et macros
- Autres directives de précompilation

Du source à l'exécutable

1. Préprocesseur : source => source expansé
   - traite les directives
     - inclusion d'en-têtes
     - remplacement de symboles
     - suppression de certaines zones de programme
   - suppression des commentaires
2. Compilateur : source expansé => langage machine
   - traduit le source expansé en langage machine
3. Assembleur : langage machine => fichier objet
   - traduit le langage machine en code machine
4. Éditeur de liens : fichier objet => exécutable
   - combine plusieurs fichiers objets

Le projet GNU

- Projet GNU
  - fournit des outils de qualité
  - disponibles avec les sources
  - diffusables librement
  - repose sur les contributions volontaires
  - démarche liée à l'environnement UNIX / LINUX
  - GNU est l'acronyme récursif de “GNU's Not UNIX” (http://www.gnu.org/)

gcc

5

- Compilateurs C :
  - cc : man cc pour plus d'info (sur Arabica)
  - gcc : compilateur C de GNU
    - implémente la norme ANSI/ISO du C
    - considéré comme d'excellente qualité
      - rapidité
      - qualité et optimisation du code généré
      - notification des erreurs
    - man gcc pour infos sur les options disponibles (sur Amalthee)

Plan

6

- La compilation en C
- La bibliothèque standard
- Les symboles et macros
- Autres directives de précompilation

La bibliothèque standard en C

7

- Le langage C inclut :
  - des types de données de base
  - des opérateurs sur les types de base
  - des constructeurs de types
  - des structures de contrôle
  - la définition et l'utilisation de fonctions
- La bibliothèque standard inclut des fonctions complémentaires
  - fait partie du standard ISO/ANSI
  - environ 150 fonctions

La bibliothèque standard en C

8

- Organisation de la bibliothèque standard en modules
- Chaque module est compilé séparément
- Lien entre un module et un programme y faisant appel
  - se fait lors de l'édition de liens
  - inclure la définition des prototypes des fonctions de la bibliothèque dans les programmes pour le compilateur
    - vérification des types et conversion éventuelle

```
#include <stdio.h>
```

La bibliothèque standard en C

9

- Une en-tête de module comprend :
  - la définition des prototypes des fonctions du module
  - le définition de types
  - la déclaration de variables externes
  - des symboles et macro-fonctions
- En-têtes de la bibliothèque standard placées dans /usr/include sous UNIX

 2012-10-01

La bibliothèque standard en C

10

- <assert.h> : gestion d'assertions
- <ctype.h> : gestion de caractères
- <errno.h> : notification d'erreurs
- <float.h> : limites de taille liées au codage des flottants
- <limits.h> : limites de taille liées au codage des entiers
- <locale.h> : paramètres d'internationalisation (caractères accentués, tri alphabétique, dates, point décimal …)
- <math.h> : fonctions mathématiques
- <setjmp.h> : sauvegarde/restauration du contexte d'exécution d'un programme

 2012-10-01

La bibliothèque standard en C

11

- <signal.h> : envoi/réception de signaux
- <stdarg.h> : gestion de fonctions avec un nombre variable d'arguments
- <stddef.h> : NULL, …
- <stdio.h> : fonctions d'entrée/sortie
- <stdlib.h> : fonctionnalités générales (allocation dynamique, nombres aléatoires …)
- <string.h> : manipulation de chaînes de caractères
- <time.h> : fonctions relatives au temps

La bibliothèque standard en C

12

- L'édition de liens est automatique pour certains modules.
- Sinon, la spécifier au compilateur avec -l
- Exemple : édition de liens avec la bibliothèque mathématique
  - -lx cherche des fichiers libx.a (ou libx.so) dans les répertoires désignés par LD_LIBRARY_PATH

gcc -lm -o toto toto.c

La bibliothèque standard en C

13

- Exemple (sans #include : À NE PAS FAIRE)

```
#include <stdio.h>

main ()
{
        printf("racine carree de %d : %f\n",5, sqrt(5));
}
```

- Donne

```
% gcc -lm -o toto toto.c
toto.c: In function `main':
toto.c:5: warning: type mismatch in implicit declaration
  for built-in function `sqrt'
% toto
racine carree de 5 : 2.236068
```

- Interprète parfois à tort des nombres flottants comme entiers

La bibliothèque standard en C

14

- Exemple correct

```
#include <stdio.h>
#include <math.h>

main ()
{
        printf("racine carree de %d : %f\n",5,
  sqrt(5));
}
```

- Donne

```
% gcc -lm -o toto toto.c
% toto
racine carree de 5 : 2.236068
```

La bibliothèque standard en C

15

- Voir la description complète, incluant de nombreux exemples, dans

Braquelaire, Jean-Pierre. 1998. "Méthodologie de la programmation en C", Ed. Masson, Paris.

  - Inclut également une description complète de la norme POSIX
    - interfaçage des programmes avec les systèmes d'exploitation (gestion des fichiers, des processus, des signaux, des pipes, E/S de bas niveau)
    - complémentaire à C ISO
- Autres sources d'information :
  - sous UNIX : man <bibliothèque>, man <fonction>
  - livre : *Kernighan, Brian W. et Ritchie, Dennis, M. "The C Programming Language", 2nd edition, Prentice-Hall, Upper Saddle River, NJ.*
  - sur le Web : site de GNU http://www.gnu.org/software/libc/manual/html_node/index.html

Plan

16

- La compilation en C
- La bibliothèque standard
- Les symboles et macros
- Autres directives de précompilation

Précompilation

17

- Pour voir le résultat de la précompilation (supprime les étapes ultérieures) :
  - gcc -E toto.c
  - gcc -E -C toto.c : ne supprime pas les commentaires
  - gcc -E -P toto.c : n'insère pas d'instructions #line

Symboles

18

- #define *identificateur valeur*
  - Le préprocesseur remplace toutes les occurrences de *identificateur* (comme mot) par *valeur*
  - Valeur : caractères jusqu'à la fin de la ligne, ou jusqu'à la fin de la ligne suivant si \ en fin de ligne (pas de ; à la fin d'une directive)
  - L'identificateur existe de sa définition jusqu'à la fin du fichier
  - Sauf si commande #undef identificateur

Symboles

19

```
/* Exemple de definition de symboles (a ne pas faire) */
#define i toto
#define begin {
#define end }
main ()
{
        int i = 6, j;
        if (i)
        begin
#undef i
                j = i * 2;
        end
}
```

% gcc -E toto.c

Instruction pour le compilateur
pour générer des messages d'erreur appropriés
(identification du fichier source et du numéro de ligne)

```
# 1 "toto.c"

main ()
{
        int toto  = 6, j;
        if (toto )
        {
                j = i  * 2;
        }
}
```


Symboles

20

- Pour éviter toute substitution inattendue
  - définir les symboles en majuscules exclusivement
  - définir les variables et noms de fonctions en minuscules
- Réserver l'usage des symboles à la définition de valeurs constantes
- Pour accroître la lisibilité, la modularité et la portabilité des programmes

Symboles

21

- Définition de symboles à la compilation
  - gcc -DLANGUE toto.c
  - équivaut à mettre la directive dans toto.c

  #define LANGUE
  - gcc -DLANGUE=fr toto.c
  - équivaut à mettre la directive dans toto.c

  #define LANGUE fr

Symboles

22

- Symboles prédéfinis
  - `__FILE__` : nom du source courant
  - `__LINE__` : numéro de ligne courante
  - `__DATE__` : date de compilation (Mmm jj aaaa)
  - `__TIME__` : heure de compilation (hh:mm:ss)
  - `__STDC__` : 1 lorsque le compilateur est conforme à la norme ISO

Symboles

23

- Les symboles doivent être utilisés lorsque les variables constantes sont interdites
  - déclaration d'un tableau :

```
const int taille = 6;
int v[taille] = {1, 2, 3, 4, 5, 6};
```

  - Message d'erreur :

$ gcc test.c
test.c: In function `main':
test.c:5: variable-sized object may not be initialized

- Du point de vue du compilateur, les variables constantes sont des variables, et non des valeurs constantes.
- La dimension d'un tableau doit être une expression constante.

Macro-fonctions

24

- Macro-fonction (ou macro) : symbole paramétrable
- #define identificateur(parametre[, …) corps
- Le remplacement ne concerne que les occurrences de identificateur(…)

Macro-fonctions

25

```
#define ABS(x) ((x) > 0 ? (x) : - (x))
main ()
{
        int i = -6, j, k;
        j = ABS(i);
        k = ABS;
        printf("ABS(%d) = %d\n",i,j);
}
```

% gcc -E -P toto.c

```
main ()
{
        int i = -6, j, k;
        j = (( i ) > 0 ? ( i ) : - ( i )) ;
        k = ABS;
        printf("ABS(%d) = %d\n",i,j);
}
```


Macro-fonctions

26

- Dangers associés aux macros
  - mauvaises substitutions si corps et/ou paramètres non-parenthésés
  - évaluation multiple des paramètres de la macro
    - erreurs dans le cas de paramètres à effet de bord
    - et inefficacité sinon
- Ne pas définir de macros
- Définir des fonctions à la place (même si définition sur une seule ligne)

Macro-fonctions

27

```
#define CARRE(x) ((x) * (x))
#define CARRE1(x) x * x
#define CARRE2(x) (x * x)
main ()
{
        int x = 6, j, k, l, m;
        j = -CARRE1(x+1);
        k = -CARRE2(x+1);
        l = -CARRE(x+1);
        m = -CARRE(x++);
}
```

Après prétraitement (gcc -E -P toto.c)

```
main ()
{
        int x = 6, j, k, l, m;
        j = - x+1  *  x+1  ;
        k = -( x+1  *  x+1 ) ;
        l = -(( x+1 ) * ( x+1 )) ;
        m = -(( x++ ) * ( x++ )) ;
}
```

Équivaut à (~x)+(1*x)+1
Équivaut à ~(x+(1*x)+1)
OK
Effets de bord

Plan

28

- La compilation en C
- La bibliothèque standard
- Les symboles et macros
- Autres directives de précompilation

#ifdef

29

```
#ifdef identificateur
  partie-si
[#else
  partie-else]
#endif
```

- Si identificateur défini, le préprocesseur recopie partie-si
- Sinon il recopie partie-else
- #ifndef : comportement opposé
- Imbrication possible des #ifdef

#ifdef

30

- Utilisations principales :
  - éviter les inclusions multiples des en-têtes sources (voir en-têtes standard)
    - on associe à chaque fichier nom.h un symbole NOM_H
    - dont on teste l'existence en début de fichier

```
#ifndef NOM_H
#define NOM_H
```

en-tête proprement dite

```
#endif /* NOM_H */
```

#ifdef

31

- Utilisations principales :
  - gestion du paramétrage de différentes versions du même programme
    - on associe à chaque version un symbole spécifique
    - dont on teste l'existence en début de fichier

```
#ifdef LINUX
# include "linux.h"
#endif /* LINUX */
#ifdef BSD4.x
# include "bsd4.x.h"
#endif /* BSD4.x */
```


#if

32

```
#if expression
  partie1-si
[#elif expression
  partie2-si]
…
[#else
  partie-sinon]
#endif
```

- Imbrication possible des #if
- Expression : basée sur des constantes, les opérateurs du C (plus l'opérateur unaire defined) et les parenthèses

exemple3.1.c


```
$ gcc -DLANG=EN -lm -o exemple3.1.en exemple3.1.c
$ gcc -DLANG=FR -lm -o exemple3.1.fr exemple3.1.c
$ gcc -DLANG=SP -lm -o exemple3.1.sp exemple3.1.c
exemple3.1.c: In function `main':
exemple3.1.c:21: `MESSAGE_ERR1' undeclared (first use in this function)
exemple3.1.c:21: (Each undeclared identifier is reported only once
exemple3.1.c:21: for each function it appears in.)
exemple3.1.c:25: `MESSAGE_ERR2' undeclared (first use in this function)
exemple3.1.c:27: `MESSAGE_RES' undeclared (first use in this function)

$ exemple3.1.en
Usage : toto n
$ exemple3.1.en -4
Provide a positive number
$ exemple3.1.en 4
The square root of 4.000000 is 2.000000

$ exemple3.1.fr
Usage : toto n
$ exemple3.1.fr -4
Fournir un nombre positif
$ exemple3.1.fr 4
La racine carree de 4.000000 est 2.000000
```