

# Preprocesorul C

## Tech Talks

Răzvan Deaconescu  
razvan@rosedu.org

ROSEdu

10 octombrie 2009

- 1 Noțiuni generale
- 2 Macrodefiniții
- 3 Preprocesare condiționată
- 4 Includerea fișierelor header
- 5 Altele
- 6 Resurse utile

## 1 Noțiuni generale

## 2 Macrodefiniții

## 3 Preprocesare condiționată

## 4 Includerea fișierelor header

## 5 Altele

## 6 Resurse utile

# Rolul preprocesorului C

- Includerea fișierelor header
- Expandarea macro-urilor
- Compilare condiționată
- Diagnosticare

# Invocarea preprocesorului

- Intern de compilator

```
gcc -Wall file.c
```

- Intern de compilator doar pentru faza preprocesării

```
gcc -Wall -E file.c -o file.i
```

- Cu ajutorul comenzii cpp

```
cpp file.c file.i
```

# Exemplu de utilizare

```
1 #include "sample.h"
2
3 #define TEST_NUM 25
4 #define MAX(a, b) ((a) > (b) ? (a) :
(b))
5
6 int main(void)
7 {
8     test_fun(TEST_NUM);
9     test_max(MAX(5, 10));
10
11     return 0;
12 }
```

```
1 # 1 "sample.c"
2 # 1 "<built-in>"
3 # 1 "<command-line>"
4 # 1 "sample.c"
5 # 1 "sample.h" 1
6
7
8
9 int test_fun(int num);
10 int test_max(int num);
11 # 2 "sample.c" 2
12
13
14
15
16 int main(void)
17 {
18     test_fun(25);
19     test_max(((5) > (10) ? (5) :
(10)));
20
21     return 0;
22 }
```

1 Noțiuni generale

**2 Macrodefiniții**

3 Preprocesare condiționată

4 Includerea fișierelor header

5 Altele

6 Resurse utile

# Directivile `#define` și `#undef`

- **Directivile de preprocesare încep cu `#`**
- Definirea și anularea definirii unui macro
- `#define MY_VAR 50`
- `#undef MY_VAR`
- `a = a + MY_VAR` → `a = a + 50`

# Macro-uri simple (object-like macros)

- `#define NUME VALOARE`
- `NUME` respectă regulile de nume pentru o variabilă
- `VALOARE` poate să conțină aproape orice
- Se înlocuiește `NUME` cu `VALOARE` peste tot

# Stupid example

```
1 #define begin    {
2 #define end      }
3 #define stop     return 0
4
5 int main(void)
6 begin
7     printf("Hello,
World!\n");
8
9     stop;
10 end
```

```
1 # 1 "stupid_example.c"
2 # 1 "<built-in>"
3 # 1 "<command-line>"
4 # 1 "stupid_example.c"
5
6
7
8
9 int main(void)
10 {
11     printf("Hello, World!\n");
12
13     return 0;
14 }
```

# Folosirea parantezelor

- O greșeală frecventă
- Once upon a time during a PSO lab (2006-2007)

```
#define NR_syscalls MY_SYSCALL_NO+1
...
new_syscall_table =
    malloc(NR_syscalls * sizeof(void *));
```

- **Nu** faceți așa

```
#define MAX(a, b) (a > b ? a : b)
```

- Faceți așa

```
#define MAX(a, b) ((a) > (b) ? (a) : (b))
```

- E OK tot timpul?

# Macro-uri vs. const

- Avantaje macro-uri
  - mai rapide (faza de preprocesare)
  - interpretate la preprocesare
  - nu ocupă spațiu în memorie
- Avantaje const
  - type safety
  - sunt, de fapt, variabile (au o adresă)
  - în C++ se pot folosi pentru alocări statice

# Macro-uri vs. enum

- Avantaje macro-uri
  - flexibile (nu sunt limitate la valori întregi)
  - nu impun constrângeri de spațiu ocupat
- Avantaje enum
  - type safety
  - block scope

## Macro-uri predefinite

```
$ gcc -dM -E sample.c
#define __CHAR_BIT__ 8
#define __unix__ 1
#define __x86_64 1
#define __SHRT_MAX__ 32767
#define __linux 1
#define __gnu_linux__ 1
#define SAMPLE_H_ 1
#define TEST_NUM 25
#define MAX(a,b) ((a) > (b) ? (a) : (b))
#define __STDC__ 1
```

```
$ gcc -dM -E sample.c | wc -l
131
```

## Definirea de macro-uri în linia de comandă

```
$ gcc -DMY_MACRO=40 -Wall comm.c
$ ./a.out
MY_MACRO = 40
```

```
$ gcc -dM -E code/sample.c | grep __linux__
#define __linux__ 1
$ gcc -U__linux__ -dM -E code/sample.c | grep __linux__
$
```

## Macro-uri cu parametri (function-like macros)

```
#define echo() printf("Hello, World!\n")
```

```
#define max(a, b) ((a) > (b) ? (a) : (b))
```

```
#define do_error(msg) \  
    perror(msg); \  
    exit(EXIT_FAILURE)
```

- E ceva greșit?

```
if (error_condition) do_error();
```

## Macro-uri cu parametri (cont.)

```
#define do_error(msg) { \  
    perror(msg); \  
    exit(EXIT_FAILURE); \  
}
```

- Inconsistență - nu e nevoie de ; (punct și virgulă) la apel

```
#define do_error(msg) do { \  
    perror(msg); \  
    exit(EXIT_FAILURE); \  
} while (0)
```

- Perfect!

# Macro-uri vs. funcții inline

- Avantaje funcții inline
  - type safety
  - fără bătăi de cap
- Avantaje macro-uri
  - sunt portabile
  - unele lucruri se realizează mai greu cu funcții inline (argumente transmise prin adresă)
  - unele lucruri nu se pot face cu funcții inline

```
#define container_of(ptr, type, member) \
    (type *)((char *)ptr - offsetof(type, member))
#define offsetof(TYPE, MEMBER) \
    ((size_t) &((TYPE *)0)->MEMBER)
```

# Macro-uri cu număr variabil de argumente (variadic macros)

- Similar funcțiilor cu număr variabil de argumente
- În standardul C99

```
#define DEBUG(fmt, ...) \  
    fprintf(stderr, fmt, __VA_ARGS__)
```

- O problemă: apel `DEBUG("problema");`
- Soluție: token paste operator (`##`) (extensie GNU)

```
#define DEBUG(fmt, ...) \  
    fprintf(stderr, fmt, ##__VA_ARGS__)
```

# Probleme în folosirea macro-urilor

- Precedența operatorilor
  - soluție: folosirea parantezelor
- Omiterea operatorului ; (punct și virgulă)
  - soluție: folosirea `do { ... } while(0)`
- Linii multiple
  - soluție: folosirea operatorului `\` (backslash)

- 1 Noțiuni generale
- 2 Macrodefiniții
- 3 Preprocesare condiționată**
- 4 Includerea fișierelor header
- 5 Altele
- 6 Resurse utile

# Directive de preprocesare condiționată

- `#if expresie_conditionala`
- `#if defined macro / #ifdef macro`
- `#endif`
- `#else`
- `#elif`

# Asigurarea portabilității

```
1 #include <stdio.h>
2
3 int main (void)
4 {
5     #if defined (__linux__)
6         printf ("This is Linux.\n");
7     #elif defined (__win32__)
8         printf ("This is Win32.\n");
9     #else
10        printf ("This is Sparta.\n");
11    #endif
12
13    return 0;
14 }
```

## “Eliminarea” codului

- Util pentru “comentarii”
- Trebuie să conțină cod “preprocesabil”

```
#if 0
    aaaa
    /*
    ...
    */
    if ...
    for ...
    ...
#endif
```

# Activare/dezactivare mod debug

```
1 #ifndef DEBUG_H_
2 #define DEBUG_H_      1
3
4 #if defined DEBUG__
5 #define Dprintf(format, ...) \
6     fprintf(stderr, "[%s]:%s:%d: " format, __FILE__, \
7         __func__, __LINE__, ##__VA_ARGS__)
8 #else
9 #define Dprintf(format, ...) do { } while(0)
10 #endif
11
12 #endif
```

- Se poate defini un macro în același fișier header
- Se poate folosi linia de comandă `gcc -DDEBUG__ ...`
- Avansat, se pot stabili niveluri de jurnalizare (`DEBUG_LEVEL + Dprintf_info, Dprintf_warn` etc.)

# Combinare cod C și cod C++

- Compilatorul de C++ face “name mangling”
- Linker-ul nu recunoaște simbolurile

```
1 #ifndef MIX_HEADER_H_
2 #define MIX_HEADER_H_ 1
3
4 #ifdef __cplusplus
5 extern "C" {
6 #endif
7
8 void fun1(void);
9 void fun2(void);
10
11 #ifdef __cplusplus
12 }
13 #endif
14
15 #endif
```

- 1 Noțiuni generale
- 2 Macrodefiniții
- 3 Preprocesare condiționată
- 4 Includerea fișierelor header**
- 5 Altele
- 6 Resurse utile

# Directiva #include

- Permite inserarea unui fișier header
- Se copiază conținutul aceluși fișier
- Poate fi (și de obicei este) folosită pe mai multe niveluri (un fișier header include alte fișiere header)

## Exemplu de structură de fișier header

```
#ifndef MY_HEADER_H_
#define MY_HEADER_H_ 1
...
#endif
```

- Macro-uri de gardă/control
- Previn includerea de mai multe ori a aceluiași fișier
- Denumite, în general `_MY_HEADER_H`, `__MY_HEADER_H` (rezervate pentru biblioteca standard), `MY_HEADER_H_`, `MY_HEADER_H__`

# Ce conține un fișier header?

- Macro de gardă (`#ifndef ...`)
- Includerea altor fișiere header
  - se folosesc tipuri definite în alt header
  - se folosesc funcții declarate în alt header (vezi `Dprintf`)
- Macro-uri
- Structuri de date, enumerări și definiții de tipuri
- Declarații **externe** de variabile
- Declarații (antete) de funcții
- Definiții de funcții statice
- Definiții de funcții inline (de asemenea statice)
- Sfârșit macro de gardă (`#endif`)

## De ce nu se include un fișier C?

- Un fișier C conține funcții și variabile globale care pot fi non-stactice
- Dacă acel fișier e inclus de două fișiere diferite care sunt link-editate apare conflict de forma *already defined*
- Un fișier header **nu** trebuie să aibă forma unui fișier C
  - fără definiții de variabile globale non-stactice
  - fără definiții de funcții non-stactice

# #include <file.h> vs. #include "file.h"

- #include <file.h>
  - system headers
  - căutare într-un set de directoare predefinite
  - localizate preponderent în /usr/include și /usr/local/include
- #include "file.h"
  - local headers
  - căutare începând cu directorul local

## Extinderea căii de căutare a fișierelor header

- `CPPFLAGS = -I/usr/include/gdk/ -iquote../include`
- Forma `#include <file.h>` caută întâi în directoarele marcate cu `-I` și apoi în cele implicite
- Forma `#include "file.h"` caută în directoarele marcate cu `-iquote`, apoi în cele marcate cu `-I`, apoi în directorul local și apoi în cele implicite

- 1 Noțiuni generale
- 2 Macrodefiniții
- 3 Preprocesare condiționată
- 4 Includerea fișierelor header
- 5 Altele**
- 6 Resurse utile

# Stringification

- Operatorul # în fața unui argument al unui macro
- Construcția este un șir de caractere reprezentând *numele* argumentului

```
1 #include <stdio.h>
2
3 #define DBG(val, fmt) \
4     fprintf(stderr, #val " = " fmt "\n", val)
5
6 int main(void)
7 {
8     int a = 42;
9     char *s = "hax0r";
10
11     DBG(a, "%d");
12     DBG(s, "%s");
13
14     return 0;
15 }
```

# Concatenare

- Compilatorul concatenează șirurile adiacente (*string literal concatenation*)
  - "an" "a" " a" "re" → "ana are"
- Operatorul ## concatenează doi tokeni într-unul singur
- Wanna see something really scary?

```
1 #include <stdio.h>
2
3 #define SYSCALL(ret, name, ...) ret __sys_ ## name(__VA_ARGS__)
4
5 SYSCALL(int, open, const char *path, int oflag, int mode)
6 {
7     /* TODO */
8     return 0;
9 }
10
11 SYSCALL(int, read, int fildes, void *buf, size_t nbytes)
12 {
13     /* TODO */
14     return 0;
15 }
```

# Diagnosticare

- Directiva `#error` forțează eroare de preprocesare și încheierea acțiunii
- Directiva `#warning` permite continuarea procesării

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5 #ifdef __x86_64__
6 #warning "why?"
7 #endif
8
9 #if defined __linux__ && ! defined DEBUG__
10 #error "DEBUG__ macro must be defined on Linux"
11 #endif
12
13     printf("Hello, World!\n");
14
15     return 0;
16 }
```

- 1 Noțiuni generale
- 2 Macrodefiniții
- 3 Preprocesare condiționată
- 4 Includerea fișierelor header
- 5 Altele
- 6 Resurse utile**

# Link-uri

- [http://en.wikipedia.org/wiki/C\\_preprocessor](http://en.wikipedia.org/wiki/C_preprocessor)
- <http://c-faq.com/cpp/index.html>

# Altele

- `man cpp`
- `info cpp`
- `comp.lang.c` (Usenet)
- `##c` (IRC, Freenode)

# Întrebări