

## Preprocesorul C

### Funcții cu număr variabil de argumente

6 decembrie 2005

Programarea calculatoarelor 2. Curs 10

Marius Minea

Preprocesorul C. stdarg.h

## Preprocesorul C

2

– extensii (macro-uri) pentru scrierea mai concisă a programelor  
– preprocesorul efectuează transformarea într-un program C propriu-zis  
– directivele de preprocesare au caracterul # la început de linie

```
#include <numefisier> SAU #include "numefisier"
– include textual fișierul numit (în mod tipic definiții)
(a doua variantă: caută întâi în directorul curent apoi în cele standard)

#define LEN 20 /* substituție textuală simplă */
int tab[LEN]; /* pentru definirea de constante simbolice */
for (i = 0; i < LEN; i++) { ... } /* mai robust la modificări */
forma generală: #define nume(arg1,...,argn) substituție
#define max(A, B) ((A) > (B) ? (A) : (B))
#define swapint(a, b) { int tmp; tmp = a; a = b; b = tmp; }
Obs: substituția se face textual ⇒ pot apărea probleme subtile
– folosiți paranteze în jurul argumentelor (evită erori de precedență)
– argumentele: evaluate la fiecare apariție textuală (ex. de 2x în max)
⇒ rezultat incorect la evaluarea repetată a expresiilor cu efect lateral
```

Programarea calculatoarelor 2. Curs 10

Marius Minea

Preprocesorul C. stdarg.h

## Operatorii # și ##

3

Operatorul # aplicat unui argument de macro produce un șir de caractere care are ca și conținut argumentul macro-ului.

```
#define mkstring(x) # x
În acest caz, mkstring(ume) devine "ume"
```

Operatorul ## are ca efect concatenarea elementelor lexicale de dinainte și de după. Exemplu:

```
#define concat(x,y) x ## y
În acest caz, concat(unu, doi) devine unudoi
```

Obs.: în standardul C două *constante șir* consecutive sunt echivalente cu concatenarea lor: "unu" "doi" e echivalent cu "unudoi").

Observație: de regulă, *definiția* unui macro nu include un ; final; acesta e scris în program după *folosirea* macro-ului, obținând un aspect uniform la scriere (ca și cum ar fi un apel de funcție)

Programarea calculatoarelor 2. Curs 10

Marius Minea

Preprocesorul C. stdarg.h

4

## Compilarea condițională

pt. a compila selectiv porțiuni de cod din program în funcție de opțiuni (caracteristici arhitecturale; pt. depanare; funcționalitate în plus; etc)

Sintaxa: *grup-cod* ::= *test-if grup-cod grupuri-elif<sub>opt</sub> grup-else<sub>opt</sub> #endif*  
*test-if* ::= #if *expr-const* | #ifdef *identificator* | #ifndef *identificator*  
*grup-elif* ::= #elif *expr-const grup-cod* (toate # apar pe linie nouă)  
*grup-else* ::= #else *grup-cod*

*expr-const* ::= cele obișnuite | defined *identificator*

```
#define DEBUG /* dacă depanăm */ #if defined __GNUC__ /* compilator GNU */
/* cod obișnuit */ #if __GNUC__ == 2 /* versiunea 2 */
#define DEBUG /* cod specific pt. versiune */
printf("am ajuns aici, x = ..."); #else /* altă versiune */
#endif /* cod specific pt. altă versiune */
/* alt cod obișnuit */ #endif
```

Identificatori predefiniți: `__FILE__` `__LINE__` `__DATE__` `__TIME__`  
Ex: `fprintf(stderr, "eroare în %s linia %d\n", __FILE__, __LINE__);`

Programarea calculatoarelor 2. Curs 10

Marius Minea

Preprocesorul C. stdarg.h

5

## Funcții cu număr variabil de argumente

Funcțiile de tipul `printf/scanf` au număr variabil de argumente (...) Pentru a implementa o astfel de funcție, trebuie un mod de acces la argumentele cu număr variabil, pornind de la ultimul arg. numit.

⇒ limbajul C definește o serie de macro-uri în `stdarg.h`

– tipul `va_list` pentru a reține informații despre lista de argumente  
`void va_start(va_list ap, ultimarg);`  
– inițializează `ap` pornind de la adresa ultimului argument  
`tip va_arg(va_list ap, tip);`

– returnează următorul argument din listă, presupus a fi de tipul `tip` apelată repetat pentru fiecare argument; tipul argumentelor și numărul lor trebuie deduse din argumentele fixe (ex. formatul la `printf/scanf`)  
`void va_copy(va_list dest, va_list src);`

– copiază un `va_list`, inclusiv punctul curent de prelucrare atins  
`void va_end(va_list ap);`  
– apelat pentru încheierea corectă a prelucrării argumentelor

Programarea calculatoarelor 2. Curs 10

Marius Minea

Preprocesorul C. stdarg.h

6

## Funcții de intrare-ieșire formatată cu nr. variabil de argumente

– declarații similare, dar în loc de ... apare `va_list ap`:

```
int vprintf(const char *format, va_list ap);
int vscanf(const char *format, va_list ap);
```

similar: `vfprintf`, `vscanf`, `vsprintf`, `vsscanf`, `vsprintf`  
– se folosesc când dorim să facem întâi o prelucrare preliminară, și apoi să apelăm `printf/scanf` etc. cu argumentele rămase

```
void errprintf(const char *format, ...)
{
    va_list ap;
    va_start(ap, format);
    fprintf(stderr, "Error: %s", strerror(errno));
    vfprintf(stderr, format, ap);
    va_end(ap);
}
```

Programarea calculatoarelor 2. Curs 10

Marius Minea