Uvod u programiranje

- predavanja -

prosinac 2019.

9. Makro

Makro (macro)

- Pravilo koje određuje kako se zadani niz znakova ulaznog teksta transformira u zadani niz znakova izlaznog teksta
- U programskom jeziku C
  - makro se definira direktivom pretprocesoru `#define`
  - transformaciju ulaznog teksta (izvornog koda) u izlazni tekst (pretprocesirani izvorni kod) obavlja pretprocesor
    - u jednom od niza koraka prevođenja
    - ili zasebno: `gcc -E prog.c`
- Makro se pojavljuje u dva oblika
  - jednostavni makro (*simple macro*, *object-like macro*)
  - makro s parametrima (*parameterized macro*, *function-like macro*)

Jednostavni makro

- Opći oblik direktive pretprocesoru

  `#define` *`identifikator lista_za_zamjenu`*

- *`lista_za_zamjenu`* može sadržavati:
  - identifikatore
  - ključne riječi
  - konstante
  - operatore
  - znakove interpunkcija
- pretprocesor će svaki *identifikator* u ulaznom tekstu (izvornom kodu) zamijeniti sadržajem *liste za zamjenu*

- prema konvenciji, identifikator za makro piše se velikim slovima
- uočiti: direktiva pretprocesoru ne smije se terminirati znakom **;**

Jednostavni makro

- prvenstveno je namijenjen imenovanju numeričkih i znakovnih konstanti te konstantnih znakovnih nizova. Imenovanjem konstanti postiže se:
  - veća čitljivost programa
    - `PLANCK_KONST` u programu je jasnije od `6.62607e-34`
  - olakšava se izmjena programa
    - promijeniti definiciju za npr. `MAX_DIM` na jednom mjestu lakše je nego po cijelom programu tražiti (i mijenjati) gdje se ta konstanta koristi
  - izbjegavaju se nekonzistentnosti
    - neće se dogoditi da se na jednom mjestu u programu napiše `3.14159`, a na drugom mjestu `3.1415926`
- ponekad se koristi za preimenovanje tipova
  - za ovo se, ipak, preporuča koristiti prikladniji mehanizam - typedef

Primjer

prog.c

```
#define C1      100
#define GRESKA  "Dogodila se greska u programu\n"

int main(void) {
   int i = C11;
   int m = C1 * 2;
   ...
   printf(GRESKA);
   printf("A GRESKA nije pozeljna");
```

```
gcc -E prog.c > prog.i
```

prog.i

```
int main(void) {
   int i = C11;                          uočiti: tekst C1 nije zamijenjen
   int m = 100 * 2;
   ...
   printf("Dogodila se greska u programu\n");
   printf("A GRESKA nije pozeljna");     uočiti: tekst GRESKA nije zamijenjen
```

Makro s parametrima

- opći oblik direktive pretprocesoru

  `#define` *identifikator*$(p_1, p_2, \ldots, p_n)$ *lista_za_zamjenu*

- *lista_za_zamjenu* može sadržavati:
  - elemente *liste za zamjenu* koji su navedeni za jednostavni makro
  - dodatno: parametre koji su navedeni u zagradama uz *identifikator*
- pretprocesor će svaku pojavu *identifikator*$(a_1, a_2, \ldots, a_n)$ u ulaznom tekstu (izvornom kodu) zamijeniti sadržajem *liste za zamjenu*, pri čemu će se svaki parametar p_i u listi za zamjenu zamijeniti argumentom a_i
- oprez: između *identifikator* i `(` ne smije biti praznina - to bi bio jednostavni makro
- iza `(` praznine su dozvoljene

Primjer

prog.c

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

int main(void) {
   printf("%d\n", MAX(2 * 3, 3));
   printf("%lf\n", MAX(3.5, 2.5 - 1.0));
```

```
gcc -E prog.c > prog.i
```

prog.i

```
int main(void) {
   printf("%d\n", ((2 * 3) > (3) ? (2 * 3) : (3)));
   printf("%lf\n", ((3.5) > (2.5 - 1.0) ? (3.5) : (2.5 - 1.0)));
```

Važna pravila

1. Direktiva pretprocesoru mora biti napisana u jednom retku ili, ako je napisana u više redaka, svaki redak osim posljednjeg treba završavati znakom \

```
#define PORUKA(jezik) ((jezik) == 'e' ?
"In the middle of difficulty lies opportunity." :
"Usred teskoca skriva se prilika.")
```
neispravno

```
#define PORUKA(jezik) ((jezik) == 'e' ? \
"In the middle of difficulty lies opportunity." : \
"Usred teskoca skriva se prilika.")
```
ispravno

2. Svaki parametar koji se koristi u *listi za zamjenu* treba zasebno uokviriti okruglim zagradama

```
#define PRODUKT(a, b) a * b
```
neispravno

```
#define PRODUKT(a, b) (a) * (b)
```
ispravnije (ali još uvijek ne potpuno - vidjeti sljedeće pravilo)

Važna pravila

3. Ako se u *listi za zamjenu* u definiciji koristi operator, cijelu *listu za zamjenu* treba uokviriti okruglim zagradama

```
#define PRODUKT(a, b) (a) * (b)          još uvijek neispravno
#define PRODUKT(a, b) ((a) * (b))        ispravno
```

4. Makro s parametrima ne smije imati prazninu između *identifikatora* i zagrade kojom počinje lista parametara

```
#define NEGATIVAN (a) (-(a))             neispravno
#define NEGATIVAN(a) (-(a))              ispravno
```

Primjer

- Što se loše može dogoditi ako cijela lista za zamjenu nije uokvirena zagradama?

```
#define PI_NA_KVADRAT 3.14159 * 3.14159
#define MAX(a, b) (a) > (b) ? (a) : (b)

int main(void) {
   double recipr = 1. / PI_NA_KVADRAT;
   double z;
   z = 2 * MAX(2, 3);
```

```
gcc -E prog.c > prog.i
```

```
int main(void) {
   double recipr = 1. / 3.14159 * 3.14159;
   double z;
   z = 2 * (2) > (3) ? (2) : (3);
```

Primjer (nastavak)

- Ako se definicije napišu ispravno:

```
#define PI_NA_KVADRAT (3.14159 * 3.14159)
#define MAX(a, b) ((a) > (b) ? (a) : (b))

int main(void) {
   double recipr = 1. / PI_NA_KVADRAT;
   double z;
   z = 2 * MAX(2, 3);
```

```
gcc -E prog.c > prog.i
```

```
int main(void) {
   double recipr = 1. / (3.14159 * 3.14159);
   double z;
   z = 2 * ((2) > (3) ? (2) : (3));
```

Primjer

- Što se loše može dogoditi ako svaki parametar u listi za zamjenu nije zasebno uokviren zagradama?

```
#define PROD(a, b) (a * b)

int main(void) {
   double x = 2.;
   double y = PROD(x + 1., x + 2.);
```

```
gcc -E prog.c > prog.i
```

```
int main(void) {
   double x = 2.;
   double y = (x + 1. * x + 2.);
```

Primjer (nastavak)

- Ako se definicije napišu ispravno:

```
#define PROD(a, b) ((a) * (b))

int main(void) {
   double x = 2.;
   double y = PROD(x + 1., x + 2.);
```

```
gcc -E prog.c > prog.i
```

```
int main(void) {
   double x = 2.;
   double y = ((x + 1.) * (x + 2.));
```

Makro s parametrima kao zamjena za funkciju?

- ZA:
  - za razliku od funkcije, ima generičku definiciju u odnosu na tip parametra (ne provjerava tipove, ne obavlja konverzije)
    - npr. makro MAX se može koristiti i za `int` i za `double` argumente
  - programi se mogu (neznatno) ubrzati
    - jer za poziv funkcije se uvijek troši dodatno vrijeme na kopiranje argumenata, upis povratne adrese na stog i vraćanje rezultata

- PROTIV:
  - izvršni kod će biti (neznatno) veći
  - makro ne provjerava tipove, ne obavlja konverzije, npr.
    - rezultat `MAX(3, 3.5)` je tipa `double`
    - rezultat `MAX(3, 2.5)` je tipa `int`

Makro s parametrima kao zamjena za funkciju?

- PROTIV (nastavak):
  - ako je parametar u *listi za zamjenu* naveden više puta i argument sadrži neki popratni efekt (*side effect*), rezultat će biti nepredvidiv ili neispravan.

```
...
#define kvadrat(x) ((x) * (x))

int main(void) {
   int a = 4;
   printf("%d", kvadrat(++a));

...
```

```
gcc -E prog.c > prog.i
```

```
   ...
   int a = 4;
   printf("%d", ((++a) * (++a)));
```

Neispravan i nepredvidiv rezultat
36