Algoritmizácia a programovanie
11. prednáška

Ján Grman

Obsah

1. typová konverzia
2. preprocesor

Typová konverzia

Typová konverzia

- prevod premennej určitého typu na iný typ
  - napr. **`int`** na **`float`**

- dva druhy konverzií:
  - implicitná - samovoľná, automatická
  - explicitná - vynútená, požadovaná

Pravidlá implicitnej typovej konverzie

1. pred vykonaním operácie sa samostatné operandy konvertujú:
   - kedykoľvek sa objaví typ **`char`** alebo **`short int`**, konvertuje sa na **`int`**
   - všetky operandy **`unsigned char`** a **`unsigned short`** sa konvertujú na **`int`** - ak **`int`** nepretečie, inak na **`unsigned int`**

Pravidlá implicitnej typovej konverzie

2. ak majú dva operandy jednej operácie rôzny typ, operand s nižšou prioritou je konvertovaný na typ s vyššou prioritou podľa hierarchie:

```
int             ⇒ unsigned int
unsigned int    ⇒ long
long            ⇒ unsigned long
unsigned long   ⇒ float
float           ⇒ double
double          ⇒ long double
```

- **`int`** má najnižšiu prioritu
- jeden operand typu **`float`** a druhý nižšiu prioritu, druhý ⇒ **`float`**

Pravidlá implicitnej typovej konverzie

3. v priraďovacích príkazoch je typ na pravej strane konvertovaný na typ z ľavej strany

   – napr.:

```
double x;

x = 5;
```

x bude mať hodnotu 5.0

```
int i;

i = 5.0;
```

i bude mať hodnotu 5

Príklady

```
char c;

c = 1;

c++;
c = c + '1';
```

1 je konvertované na **char**, **c** obsahuje znak "**ctrl A**"

c obsahuje znak "**ctrl B**"

ordinálne číslo znaku **'1'** je 49, teda **c** obsahuje znak s ordinálnym číslom 51, teda znak '**3**'

Príklady

```
int i;

i = 'A';
i = 'A' + 2;
i = 3.8;
```

i obsahuje ordinálne číslo znaku 'A', teda 65

c obsahuje znak číslo 67 (ordinálne číslo znaku **'C'**

i bude mať hodnotu 3 (0.8 sa odreže)

Príklady

```
double g;
int i;
char c;

g = 5;
i = g * c;
```

g obsahuje číslo 5.0

najprv sa **c** skonvertuje na **int**, potom sa **c** skonvertuje na **double**, výsledok **g * c** je **double**, ale podľa pravidla 3 sa skonvertuje na **int** a priradí **i**

Explicitná typová konverzia

- pretypovanie
  - jazyk C dovoľuje takmer ľubovoľnú konverziu → riziko, že to bude nevhodné
- syntax: **`(typ) výraz`**
- príklady:

```
(int) char_vyraz          - prevod znaku na ordinálne číslo
(char) int_vyraz          - prevod ordinálneho čísla na znak
(int) double_vyraz        - odrezanie desatinnej časti
(double) int_vyraz        - prevod celého čísla na reálne
(double) float_vyraz      - zväčšenie presnosti
```

Explicitná typová konverzia

- najčastejšie na pretypovanie ukazovateľov
- príklad použitia :

```
#include <math.h>
...
int i = 10;
double f;
f = sqrt((double) i);
```

`sqrt()` vráti druhú odmocninu

nie je to nutné, ale čitatelné (nutné to bolo v prvých verziách jazyka C)

Príklad 4: typová konverzia

program načíta tri celé čísla a vypíše
ich priemer na dve desatinné čísla

```
#include <stdio.h>

int main()
{
      int x, y, z;

      printf("Zadajte 3 cele cisla: ");
      scanf("%d %d %d", &x, &y, &z);

      printf("Priemer: %.2f\n", (x+y+z) / 3.0);
      return 0;
}
```

Preprocesor

Činnosť preprocesora

- spracováva zdrojový text **PRED** kompilátorom
- zamieňa text, napr. identifikátory konštánt za číselné hodnoty
- vypustí zo zdrojového textu všetky komentáre
- prevádza podmienený preklad
- nekontroluje syntakticú správnosť programu

- riadok, ktorý má spracovávať preprocesor sa začína znakom #

Konštrukcie pre preprocesor

- definovanie makra

```
#define meno_makra text
```

- zrušenie definície makra

```
#undef meno_makra
```

- podmienený preklad v závislosti na konštante **`konst`**

```
#if konst
#elif #else #endif
```

Konštrukcie pre preprocesor

- vloženie textu zo špecifikovaného súboru zo systémového adresára

```
#include <filename>
```

- vloženie textu zo špecifikovaného súboru v adresári používateľa

```
#include "filename"
```

- výpis chybových správ vo fáze predspracovania

```
#error text
```

Konštrukcie pre preprocesor

- podmienený preklad v závislosti od toho, či je makro definované, alebo nedefinované

```
#ifdef meno_makra
#elif #else #endif
```

- podmienený preklad v závislosti od toho, či je makro nedefinované, alebo definované

```
#ifndef meno_makra
#elif #else #endif
```

Konštanty - makrá bez parametrov

- symbolické konštanty
- používajú sa často (zbavujú program "magických čísel")
- väčšinou definované na začiatku modulu
- platnosť konštánt je do konca modulu
- náhrada konštanty hodnotou - rozvoj (expanzia) makra

Pravidlá pre písanie konštánt

- mená konštánt - veľkými písmenami
- meno konštanty je od hodnoty oddelené apsoň jednou medzerou
- za hodnotou by mal byť vysvetľujúci komentár
- nové konštanty môžu využívať skôr definované konštanty
- ak je hodnota konštanty dlhšia ako riadok, musí byť na konci riadku znak \ (nie je súčasťou makra)

Príklady defninovania konštánt

```
#define MAX 1000
#define PI 3.14
#define DVE_PI (2 * PI)
#define MOD %
#define AND &&
#define MENO_SUBORU "list.txt"
#define DLHA_KONSTANTA Toto je dlha konstanta, \
            ktora sa nezmesti do jednoho riadku.
```

- za hodnotou nie je ;
- medzi menom konštanty a jej hodnotou nie je =

Príklad požitia konštanty: výpočet obsahu kruhu

```
#include <stdio.h>
#define PI 3.14

int main(){
    double r;

    printf("Zadajte polomer: ");
    scanf("%lf" &r);
    printf("Obvod kruhu s polomerom %f je %f\n",
                r, 2 * r * PI);
   return 0;
}
```

Príklad použitia konštanty: malé písmená zmení na veľké

```
#include <stdio.h>
#define POSUN ('a' - 'A')
#define EOLN '\n'
#define PRED_MALE  '*'

int main() {
   int c;

   while((c = getchar()) ! = EOLN) {
      if (c >= 'a' && c <= 'z') {
         putchar(PRED_MALE);
         putchar(c - POSUN);
      }
      else
         putchar(c);
   }
   return 0;
}
```

ak je symbolickou konštantou výraz, vhodné je uzavrieť ho do zátvoriek

malé písmeno zmení na veľké a pred neho vypíše '*', inak vypíše načítaný znak

Kedy sa nerozvinie makro

- makro sa nerozvinie, ak je uzatvorené v úvodzovkách

```
#define MENO   "Katka"

...
printf("Volam sa MENO");

printf("Volam sa %s", MENO);
```

vypíše sa:
Volam sa MENO

vypíše sa:
Volam sa Katka

Prekrývanie definícií

- nová definícia prekrýva starú, pokiaľ je rovnaká (to ani nemá zmysel)
- ak nie je rovnaká:
  - zrušiť starú definíciu: `#undef meno_makra`
  - definovať meno_makra

```
#define POCET 10
#undef POCET
#define POCET 20
```

Makro ako skrytá časť programu

```
#define ERROR { printf("Chyba v datach.\n"); }
```

- pri použití nie je makro ukončené bodkočiarkou:

```
if (x == 0)
    ERROR
else
    y = y / x;
```

Makrá s parametrami

- krátka a často používaná funkcia vykonávajúca jednoduchý výpočet
  - problém s efektivitou (prenášanie parametrov a úschova návratovej hodnoty je časovo náročnejšia ako výpočet)
  - preto namiesto funkcie - makro (to sa pri preprocesingu rozvinie)

- je potrebné sa rozhodnúť medzi
  - funkcia: kratší ale pomalší program
  - makro: rýchlejší ale dlhší program

Makrá s parametrami

- nazývajú sa vkladané funkcie - rozvitie makra znamená, že sa meno makra nahradí jeho telom

definícia makra

```
#define je_velke(c) ((c) >= 'A' && (c) <= 'Z')
```

- zátvorka, v ktorej sú argumenty funkcie - hneď za názvom makra (bez medzery)

v zdrojovom súbore

```
ch = je_velke(ch) ? ch + ('a' - 'A') : ch;
```

rozvinie sa

```
ch = ((ch) >= 'A' && (ch) <= 'Z') ? ch + ('a'-'A') : ch;
```

Makrá s parametrami

- telo makra - uzavrieť do zátvoriek, inak môžu nastať chyby, napr.:

```
#define sqrt(x) x * x
...
 f + g * f + g;
```

po rozvinutí makra

- správne

```
 #define sqrt(x) ((x) * (x))
 ...
 ((f + g) * (f + g));
```

po rozvinutí makra

Preddefinované makrá

- **`getchar()`** a **`putchar()`** (v **`stdio.h`**)

```
#define getchar() getc(stdin)
#define putchar(c) putc(c, stdout)
```

- makrá v **`ctype.h`** - makrá na určenie typu znaku
  - **`isalnum`** - vráti 1, ak je znak číslica alebo malé písmeno
  - **`isalpha`** - vráti 1, ak je znak malé alebo veľké písmeno
  - **`isascii`** - vráti 1, ak je znak ASCII znak (0 až 127)
  - **`iscntrl`** - vráti 1, ak je znak Ctrl znak (1 až 26)
  - ...
  - viac v Herout: Učebnice jazyka C

Preddefinované makrá

- makrá v **`ctype.h`** - makrá na konverziu znaku
  - **`tolower`** - konverzia na malé písmeno
  - **`toupper`** - konverzia na veľké písmeno
  - **`toascii`** - prevod na ASCII - len najnižších 7 bitov je významných

Vkladanie súborov

- vkladanie systémových súborov **< >**
- vkladanie súborov v aktuálnom adresári **" "**

```
#include <stdio.h>
#include <ctype.h>
#include "KONSTANTY.H"
```

Podmienený preklad

- u väčších programov
  - ladiace časti - napr. pomocné výpisy
- program
  - trvalá časť
  - voliteľná časť (napr. pri ladení, alebo ak je argumentom programu nejaký prepínač)

Riadenie prekladu hodnotou konštantného výrazu

```
#if konstantny_vyraz
    cast_1
#else
    cast_2
#endif
```

ak je hodnota konštantného výrazu nenulová, vykoná sa časť 1, inak časť 2

```
#if 0
    cast programu, co
    ma byt vynechana
#endif
```

ak pri testovaní nechcete prekladať časť programu, namiesto `/* */` (problém by robili vhniezdené komentáre)

Riadenie prekladu hodnotou konštantného makra

```
#define PCAT 1

#if PCAT
    #include <conio.h>
#else
    #include <stdio.h>
#endif
```

- ak je program závislý na konkrétnom počítači
- ak na PC/AT - definujeme PCAT na 1, inak na 0

Riadenie prekladu definíciou makra

```
#define PCAT

#ifdef PCAT
    #include <conio.h>
#else
    #include <stdio.h>
#endif
```

- ak je program závislý na konkrétnom počítači
- ak na PC/AT - definujeme PCAT (bez hodnoty),
- stačí, že je konštanta definovaná

```
#ifndef PCAT
```

- ak nie je definovaná konštanta

```
#undef PCAT
```

- zrušenie definície makra

Operátory defined, #elif a #error

- **`#ifdef`**, alebo **`#ifndef`** zisťujú existenciu len jednoho symbolu, čo neumožňuje kombinovať viaceré

- ak treba kombinovať viaceré podmienky:

```
#if defined TEST
```

```
#if !defined TEST
```

- **`#elif`** - má význam else-if

- **`#error`** - umožňuje výpis chybových správ (v priebehu preprocesingu - nespustí sa kompilácia)

Operátory defined, #elif a #error - príklad

```
#if defined(ZAKLADNY)    &&    defined(DEBUG)
    #define VERZIA_LADENIA     1
#elif defined(STREDNY)    &&    defined(DEBUG)
    #define VERZIA_LADENIA 2
#elif !define(DEBUG)
    #error Ladiacu verziu nie je mozne pripravit!
#else
    #define VERZIA_LADENIA     3
#endif
```

Oddelený preklad

- program sa delí na menšie časti - moduly
  - logicky sa program delí na časti
  - je veľký
  - pracuje na ňom viac programátorov
  - aby bol prehľadný
- moduly
  - oddelené - zvlášť súbory
  - obsahujú premenné a funkcie, ktoré môžu povoliť alebo zakázať používať inými modulmi

najprv musíme prebrať funkcie...

Príklady

Príklad 1

program vypíše súčet prvých **N** čísel,
kde **N** je symbolická konštanta

```
#include <stdio.h>
#define N 5

int main() {
    int i, suma = 0;

    for (i = 1; i <= N; i++)
        sum += i;

    printf("Sucet prvych %d cisel je %d\n", N, suma);
    return 0;
}
```

Príklad 2

program použije makro **`na_tretiu(x)`**, ktorá bude počítať tretiu mocninu a použije ho v rôznych výrazoch

```
#include <stdio.h>

#define na_tretiu(x)  ((x) * (x) * (x))

int main(void) {
  int i = 2,
      j = 3;

  printf("%d^3 = %d", 3, na_tretiu(3));
  printf("%d^3 = %d", i, na_tretiu(i));
  printf("%d^3 = %d", 2+3, na_tretiu(2+3));
  printf("%d^3 = %d", i*j+1, na_tretiu(i*j+1));
  return 0;
}
```

```
2+3*2+3*2+3 =
(nie 12
```

```
2*3+1*2*3+
3+1 = 19,
```

```
3^3 = 9
```

```
5^3 =
```

Príklad 3

program zistí, či bola načítaná nula - pomocou makra **`citaj_int(x)`**

```
#include <stdio.h>

#define citaj_int(i) (scanf("%d", &i), i)

int main() {
   int j, k;

   printf("Zadajte cele cislo: ");
   if((j = citaj_int(k)) == 0)
      printf("Bola nacitana nula.\n");
   printf("Bolo nacitane cislo %d", k);
   return 0;
}
```

Príklad 4

program vynásobí dve čísla len pomocou sčitovania, v cykle použijeme ladiace výpisy

```
#include <stdio.h>
#define LADENIE
int main() {
   int x, y, nasobok = 0;

   printf("Zadajte dve cisla: ");
   scanf("%d %d", &x, &y);

   printf("%d * %d = ", x, y);
   for(; y>0; y--) {
      nasobok += x;

#ifdef LADENIE
            printf("\n(y: %d, nasobok: %d)\n", y, nasobok);
#endif

   }
   printf("%d\n", nasobok);
   return 0;
}
```