Mikroradiče

Jazyk C pre mikroradiče

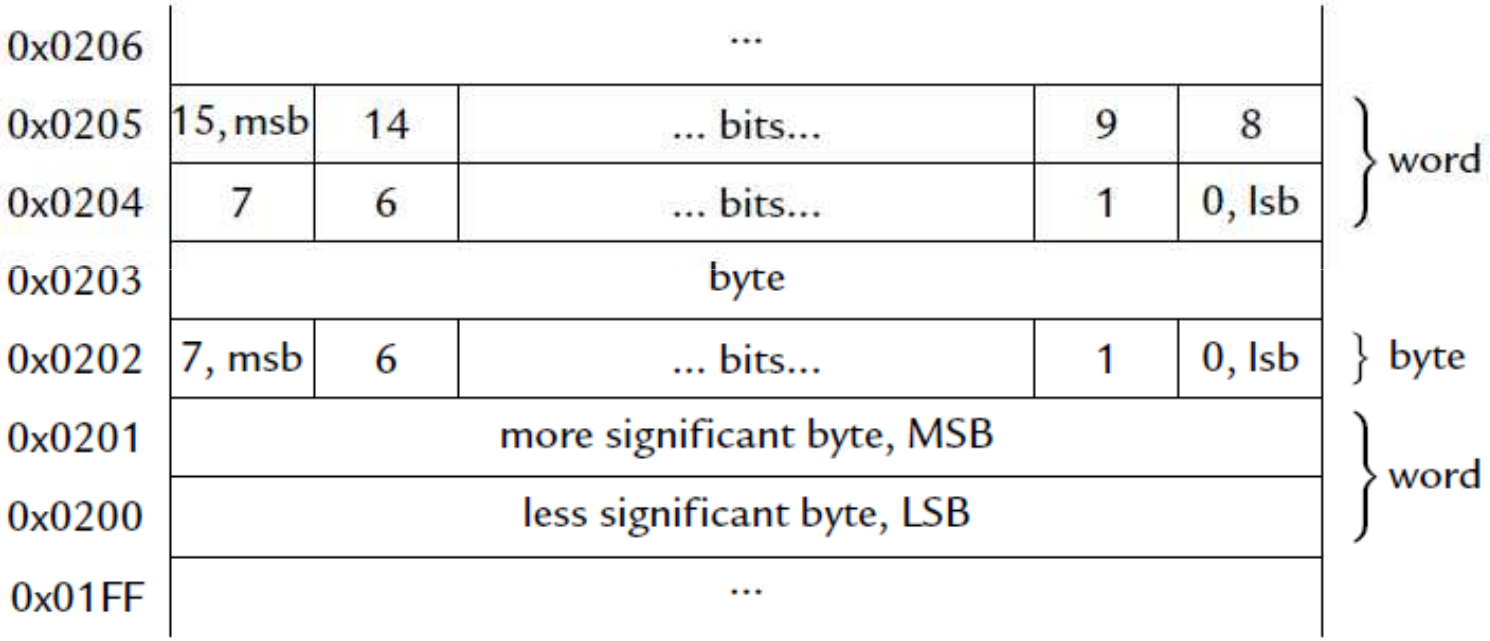
Operátory, typová konverzia, riadenie toku programu

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
ÚSTAV ELEKTRONIKY A FOTONIKY
Laboratórium DSP a mikroradičov

:: Usporiadanie dát v pamäti - little endian a big endian

- Little-endian
  Nižší byte je uložený na nižšej adrese a vyšší byte na vyššej adrese. Tento spôsob usporiadania dát v pamäti používa aj procesor MSP430 a je v procesorovej technike bežnejší.

- Big-endian
  Vyšší byte je uložený na nižšej adrese. Tento spôsob uspriadania dát používa napr. procesor Freescale HCS08.



:: Aritmetické operátory

operator	meaning	examples
+	addition	x=3+2; /*constants*/ y+z; /*variables*/ x+y+2; /*both*/
-	subtraction	3−2; /*constants*/ **int** x=y−z; /*variables*/ y−2−z; /*both*/
*	multiplication	**int** x=3*2; /*constants*/ **int** x=y*z; /*variables*/ x*y*2; /*both*/

:: Aritmetické operátory

operator	meaning	examples
/	division	**float** x=3/2; /∗produces x=1 (int /) ∗/ **float** x=3.0/2 /∗produces x=1.5 (float /) ∗/ **int** x=3.0/2; /∗produces x=1 (int conversion)∗/
%	modulus (remainder)	**int** x=3%2; /∗produces x=1∗/ **int** y=7;**int** x=y%4; /∗produces 3∗/ **int** y=7;**int** x=y%10; /∗produces 7∗/

:: Relačné operátory

Relačné operátory porovnávajú dva operandy. Výsledkom porovnania je hodnota boolean. V jazyku C je akákoľvek nenulová hodnota (v zmysle konvencie 1), považovaná za ’true’ a 0 je považovaná za ’false’.

operator	meaning	examples
>	greater than	3>2; /*evaluates to 1 */ 2.99>3 /*evaluates to 0 */
>=	greater than or equal to	3>=3; /*evaluates to 1 */ 2.99>=3 /*evaluates to 0 */
<	lesser than	3<3; /*evaluates to 0 */ 'A'<'B' /*evaluates to 1*/
<=	lesser than or equal to	3<=3; /*evaluates to 1 */ 3.99<3 /*evaluates to 0 */

:: Relačné operátory

- operátor "==" je odlišný od operátora priradenia "="
- je operátor "==" aplikovaný na premenné typu float spoľahlivý?

operator	meaning	examples
==	equal to	3==3; /*evaluates to 1 */ 'A'=='a' /*evaluates to 0 */
!=	not equal to	3!=3; /*evaluates to 0 */ 2.99!=3 /*evaluates to 1 */

:: Logické operátory

Vyhodnocovanie výrazov obsahujúcich logické operátory je v jazyku C ukončené v okamihu, kedy je výsledná logická hodnota zrejmá.
Pozor na možné nežiaduce efekty v zdrojovom kóde:

```
(3==3) || ((c=getchar())=='y') ;druhý výraz nebude vyhodnotený

(0) && ((x=x+1)>0)  ;druhý výraz nebude vyhodnotený
```

operator	meaning	examples
&&	AND	((9/3)==3) && (2*3==6); /*evaluates to 1 */ ('A'=='a') && (3==3) /*evaluates to 0 */
||	OR	2==3 || 'A'=='A'; /*evaluates to 1 */ 2.99>=3 || 0 /*evaluates to 0 */
!	NOT	!(3==3); /*evaluates to 0 */ !(2.99>=3) /*evaluates to 1 */

:: Inkrementačné a dekrementačné operátory

Patria k základným aritmetickým operátorom.

Postfix

x++	zodpovedá	x=x+1	
x−−	zodpovedá	x=x−1	
y=x++	zodpovedá	y=x; x=x+1	x je najskôr priradené y a potom inkrementované.
y=x−−	zodpovedá	y=x; x=x−1	x je najskôr priradené y a potom dekrementované.

Prefix

++x	zodpovedá	x=x+1	
−−x	zodpovedá	x=x−1	
y=++x	zodpovedá	x=x+1; y=x;	x je najskôr inkrementované a potom priradené y
y=−−x	zodpovedá	x=x−1; y=x;	x je najskôr dekrementované a potom priradené y

:: Bitové operátory

operator	meaning	examples
&	AND	0x77 & 0x3; /*evaluates to 0x3 */ 0x77 & 0x0; /*evaluates to 0 */
|	OR	0x700 | 0x33; /*evaluates to 0x733 */ 0x070 | 0 /*evaluates to 0x070 */
^	XOR	0x770 ^ 0x773; /*evaluates to 0x3 */ 0x33 ^ 0x33; /*evaluates to 0 */
«	left shift	0x01<<4; /*evaluates to 0x10 */ 1<<2; /*evaluates to 4 */
»	right shift	0x010>>4; /*evaluates to 0x01 */ 4>>1 /*evaluates to 2 */

:: Maskovanie pomocou bitových operátorov

Jednotlivé bity majú obvykle štandardnú definíciu uvedenú v hlavičkovom súbore daného procesora, napr. vo forme:

```
BIT3 = 00001000
```

Testovanie individuálneho bitu pomocou maskovania

```
if ((P1IN & BIT3) == 0) { // Test P1.3
        // akcia ak P1.3 == 0
} else {
        // akcia ak P1.3 != 0
}
```

Modifikácia individuálneho bitu pomocou maskovania

Nastavenie bitu vykonávame pomocou operátora OR:

```
P1OUT |= BIT3
```

Nulovanie bitu vykonávame pomocou operátora AND:

```
P1OUT &= BIT3
```

Zmena stavu bitu na opačný (toggling):

```
P1OUT ^= BIT3
```

:: Operátory priradenia

Pre časté výrazy pri programovaní v jazyku C typu

```
x=x+1
x=x*10
x=x/2
```

je v jazyku C zavedených niekoľko operátorov priradenia:

```
x+=1        /* zodpovedá x=x+1*/
x-=1        /* zodpovedá x=x-1*/
x*=10       /* zodpovedá x=x*10 */
x/=2        /* zodpovedá x=x/2 */
x%=2        /* zodpovedá x=x%2 */
```

:: Podmienené výrazy

Jednou z najbežnejších štruktúr jazyka C a väčšiny iných programovacích jazykov je podmienený výraz

```
if (cond)
        x = <expr_a>;
else
        x = <expr_b>;
```

V jazyku C túto štruktúru môžeme zjednodušiť použitím ternárneho operátora '?:'. Ako potom zapíšeme nasledujúcu štruktúru?

```
if ( x > 0 )
        sign = 1;
else
        sign = -1;
```

:: Typová konverzia

V prípade výrazov s rôznymi dátovými typmi vykonáva kompilátor jazyka C automaticky tzv. implicitnú konverziu:

```
int i;
float f;
f = i + 3.14159;  / * typ i bude zmenený na float,
                    f = (float) i + 3.14159 * /
```

Kompilátor tiež automaticky vykonáva konverziu ’char’ → ’int’.
Toto umožňuje porovnávanie a manipuláciu so znakovými premennými:

```
isupper = (c>=’A’ && c<=’Z’) ? 1 : 0;
/ * c a  znakové konštanty sú konvertované na int * /

if (! isupper )
      c = c – ’a’ + ’A’;
/ * odčítanie je možné vďaka konverzii na int * /
```

:: Typová konverzia

Výrazy, ktoré nedávajú zmysel (použitie float v úlohe indexu pola) nie sú povolené.

Výrazy, pri ktorých by mohlo dôjsť k strate informácie (priradenie väčšieho celočíselného typu menšiemu) môžu vyvolať varovné hlásenie kompilátoru, ale nie sú zakázané.

V akomkoľvek výraze môžeme vynútiť explicitnú konverziu pomocou unárneho operátora pretypovania:

```
(názov_typu) výraz
```

:: Blokový príkaz

Jednoduchý príkaz končí bodkočiarkou: `z = funkcia(x+y);`

Viaceré príkazy

```
temp = x+y;
z = funkcia(temp);
```

môžeme pomocou zložených zátvoriek kombinovať do zloženého príkazu, resp. bloku, pričom premenné môžeme deklarovať v rámci takéhoto bloku:

```
{
   int temp = x+y;
   z = funkcia(temp);
}
```

:: Blokový príkaz

Blokom môžeme nahradiť jednoduchý príkaz.

Blok môže byť prázdny: { }

Bloky môžeme vnárať:

```
{
   int temp = x+y;
   z = funkcia(temp);
   {
      float temp2 = x*y;
      z += funkcia2(temp2);
   }
}
```

Za blokom neuvádzame bodkočiarku.

:: Podmienené príkazy

- príkaz if
- príkaz switch

:: Príkaz **if**

```
if(x%2)
    y+=x/2;
```

- príkaz vyhodnotí podmienku `if(x%2==0)`
- ak je pravdivá, potom vyhodnotí vnútorný výraz `y+=x/2;`
- ak nie je pravdivá, nevykoná sa nič

:: Príkaz **if** – kľúčové slovo **else**

```
if (x % 2 == 0)
      y += x / 2;
else
      y += (x + 1) / 2;
```

- kľúčové slovo `else` je voliteľné
- určujeme ním príkaz, ktorý sa vykoná, ak podmienka nie je splnená: `y += (x+1)/2;`
- ktorýkoľvek z vnútorných príkazov môže byť blok

:: Príkaz **if** – kľúčové slovo **else if**

```
if (x % 2 == 0)
      y += x / 2;
else if (x % 4 == 1)
      y += 2 * ((x + 3) / 4);
else
      y += (x + 1) / 2;
```

- ďalšia voliteľná alternatíva riadenia toku príkazov
- podmienky sú vykonávané v poradí až kým jedna nie je splnená, potom sa vykoná jej vnútorný príkaz
- ak je splnených viacero podmienok, je vykonaná iba prvá v poradí
- spôsob zápisu je ekvivalentný vnorenému príkazu `if`

:: Vnáranie príkazov **if**

```
if (x % 4 == 0)
      if (x % 2 == 0)
            y = 2;
else
      y = 1;
```

Ktorému z dvoch príkazov `if` prináleží kľúčové slovo `else`?

Ak chceme priradiť `else` k vonkajšiemu príkazu `if`, musíme použiť zátvorky

```
if (x % 4 == 0){
      if (x % 2 == 0)
            y = 2;
} else
      y = 1;
```

:: Príkaz **switch**

- alternatívny podmienený príkaz
- vstupom môže byť premenná typu `int` alebo `char`
- podľa konkrétnej hodnoty testovanej premennej sa vykoná príslušný príkaz (blok)

```
switch (ch) {
      case 'Y' :  / * ch == 'Y' * /
            / * vykonaj nejaký príkaz * /
        break ;
      case 'N' :  / * ch == 'N' * /
            / * vykonaj iný príkaz * /
        break ;
      default :   / * keď ani jeden test
                      nie je pozitívny * /
            / * vykonaj príkaz * /
        break ;
}
```

:: Príkaz **switch** – špeciálne prípady

- pri porovnávaní zhody v jednotlivých prípadoch postupuje podľa poradia, v akom sú v štruktúre jednotlivé prípady uvedené
- v prípade nájdenia zhody začne vykonávať vnútorný kód prislúchajúci danému prípadu až kým nenarazí na príkaz `break;`
- ak príkaz `break;` nie je uvedený, pokračuje ďalej vo vykonávaní príkazov

```
switch (ch) {
      case 'Y' :
      case 'y' :
            / * vykonaj príkazy
           ak ch == 'Y' alebo
                ch == 'y' * /
        break ;
}
```

:: Príkaz **switch** – špeciálne prípady

- pri porovnávaní zhody v jednotlivých prípadoch postupuje podľa poradia, v akom sú v štruktúre jednotlivé prípady uvedené
- v prípade nájdenia zhody začne vykonávať vnútorný kód prislúchajúci danému prípadu až kým nenarazí na príkaz `break;`
- ak príkaz `break;` nie je uvedený, pokračuje ďalej vo vykonávaní príkazov

```
switch (ch) {
      case 'Y' :
            / * vykonaj nejaký príkaz ak
                ch == 'Y' * /
      case 'N' :
            / * vykonaj nejaký príkaz ak
                ch == 'Y' alebo
                ch == 'N' * /
        break ;
}
```

:: Príkazy pre tvorbu cyklov

- **`while`**
- **`for`**
- **`do - while`**
- kľúčové slová **`break`** a **`continue`**

:: Príkaz **while**

```
while (/ * podmienka * /)
        / * loop body * /
```

- najjednoduchšia realizácia cyklu – kým je podmienka splnená je opakovane vykonávané telo cyklu
- podmienka je vyhodnocovaná najskôr, teda k vykonaniu tela vôbec nemusí dôjsť

:: Príkaz **for**

```
int factorial (int n) {
      int i, j = 1;
      for (i = 1; i <= n; i++)
            j *= i;
      return j;
}
```

- realizácia cyklov s daným počtom opakovaní
- v zátvorkách uvádzame tri výrazy oddelené bodkočiarkou
  - inicializácia: i = 1
  - podmienka: i <= n
  - inkrementácia počítadla: i++
- výrazy môžu byť prázdne – prázdna podmienka je považovaná za vždy pravdivú

while ekvivalent cyklu **for**:

```
int factorial (int n) {
      int j = 1;
      int i = 1;                /* inicializácia */
      while (i <= n) {          /* podmienka */
            j *= i;
            i ++;               /* inkrementácia */
      }
      return j;
}
```

Viaceré výrazy v príkaze **for** oddeľujeme čiarkou:

```
int factorial (int n) {
      int i, j;
      for (i =1, j=1; i<= n; j *= i , i++)
            ;
      return j;
}
```

V tomto prípade predstavuje čiarka operátor s najnižšou prioritou, ktorý je vyhodnocovaný zľava doprava. Nezamieňame ho teda s čiarkou medzi argumentmi funkcie.

:: Príkaz **do – while**

```
char c;
do {
      /* telo cyklu */
      puts( „Pokracovat? (y/n) " );
      c = getchar();
      /* dalsie prikazy */
} while (c == 'y' && /* dalsie podmienky */ );
```

Od cyklu `while` sa líši tým, že podmienka je vyhodnocovaná až po každej iterácií, t.j. telo cyklu bude vykonané najmenej jeden krát.

Na konci musí byť bodkočiarka.

:: Kľúčové slovo **break**

- predčasné ukončenie cyklu
- `break;` ukončuje najvnútornejšiu slučku alebo príkaz `switch`
- modifikácia príkladu `do-while`:

```
char c;
do {
   /* telo cyklu */
   puts ( "Keep going? (y/n) " );
   c = getchar ( ) ;
   if (c != 'y')
     break ;
   /* dalsie prikazy */
} while (/* dalsie podmienky*/);
```

:: Kľúčové slovo **continue**

- používame ho pri vynechávaní iterácie
- `continue;` preskočí zvyšok najvnútornejšej slučky a ihneď vyvolá skok na podmienku cyklu

```
#define min(a,b) ((a) < (b) ? (a) : (b))

int gcd (int a, int b) {
      int i , ret = 1, minval = min(a,b);
      for (i = 2; i <= minval; i++) {
        if (a%i)
          continue ;
        if (b%i == 0)
          ret = i;
      }
return ret ;
}
```

:: Užitočné referencie

1. Kernighan B., Ritchie D.: Programovací jazyk C, Computer Press, 2006, ISBN: 80-251-0897-X

2. Herout, P.: Učebnice jazyka C, 4. vydání, Kopp, 2004,ISBN 80-7232-220-6

3. Texas Instruments, Inc.: MSP430 Optimizing C/C++ Compiler v.4.2 User's Guide (Rev. H)

Koniec prednášky

Jazyk C pre mikroradiče

Operátory, typová konverzia, riadenie toku programu