Bitové operácie v jazyku C

Náhradná Prednáška 11, PROG-2, 2020
Pavol Zajac

Jazyk C je často využívaný na systémovej úrovni, pre písanie efektívnych algoritmov a knižníc, vrátane ovládačov hardvéru a programovania mikroprocesorov. Kým na štandardnom PC máme k dispozícii veľké množstvo pamäte, na mikroprocesoroch a obmedzených zariadeniach je potrebné pamäť šetriť a efektívne využívať každý bit. Preto sa dostávame k téme bitových operácií. Na štandardných PC majú bitové operácie tiež význam, vďaka nim sa dá realizovať veľké množstvo algoritmov efektívne: jedna bitová operácia na 64-bitovom procesore naraz spracuje 64 bitov, čo môže pri vhodnom algoritme priniesť 64-násobné zrýchlenie.

Ak preferujete video zdroje, opäť som vybral nejaké referencie:

Všeobecný prehľad bitových operácií (bez ohľadu na jazyk):
https://www.youtube.com/watch?v=NLKQEOgBAnw

Séria 4 prednášok zameraná na bitové operácie v jazyku C:
https://www.youtube.com/watch?v=jlQmeyce65Q
https://www.youtube.com/watch?v=8aFik6lPPaA
https://www.youtube.com/watch?v=GhhJP6vpEA8
https://www.youtube.com/watch?v=kYR5biY4OHw

Bitové operátory

Bitové operátory sa dajú aplikovať na celočíselný typ (so znamienkom alebo bez znamienka). Bitové operátory sa aplikujú vždy na celé slovo daného typu, čiže počet bitov, s ktorými sa pracuje závisí na zvolenom type (od 8 bitov pre `char` po 64 bitov pre `long long`).

Pozn.: Pripomeňme si, že v jazyku C neviem priamo zadávať binárne čísla ako konštanty. Na tento účel sa využíva zväčša hexadecimálna sústava: každá hexa číslica kóduje presne 4 bity. Napr. konštanta `0xdeadbeef` je v binárnom zápise `1101 1110 1010 1101 1011 1110 1110 1111`. POZOR, presné umiestnenie bitov a bajtov v pamäti závisí od architektúry počítača, v jazyku C pracujeme však vždy chápeme čísla ako reťazce bitov zapísané zľava doprava od najvýznamnejšieho bitu (MSB, most significant bit) po najmenej významný bit (LSB, least significant bit). Ak je číslo kratšie ako rozsah typu, dopĺňa sa zľava nulami /kladné čísla, alebo bezznamienkový typ/, resp. jednotkami /záporné čísla/.

V príkladoch budeme používať typ `typedef unsigned char byte;`

Unárny operátor, bitová negácia:

```
byte a = 0xc5;
printf("%02x", ~a);     //~ vymeni kazdy bit za opacny 0 <-> 1

> 3a

Bitovo: a = 1100 0101       ~a = 0011 1010
```

Binárne logické operátory, bitové OR, AND, XOR:

```
byte a = 0xc5, b = 0x0f;
printf( "%02x", a | b );    // |  bitove OR

> cf

Bitovo: a = 1100 0101
        b = 0000 1111
      a|b = 1100 1111
```

Bitové OR premennej s konštantou sa používa, keď chceme nastaviť nejaké bity na hodnotu 1, bez ohľadu na to, čo je v premennej.

```
printf( "%02x", a & b );    // &  bitove AND

> 05

Bitovo: a = 1100 0101
        b = 0000 1111
      a&b = 0000 0101
```

Bitové AND premennej s konštantou sa používa, keď chceme nastaviť nejaké bity na hodnotu 0, bez ohľadu na to, čo je v premennej.

```
printf( "%02x", a ^ b );    //  ^  bitove XOR

> ca

Bitovo: a = 1100 0101
        b = 0000 1111
      a&b = 1100 1010
```

Bitové XOR premennej s konštantou sa používa, keď chceme vymeniť vybrané bity v premennej (selektívna negácia).

Aritmetické a logické posuny vľavo a vpravo:

```
byte a = 0xc5;
char b = 0xc5;

printf( "%02x", a << 1 );    // posun dolava o 1 bit

> 8a

Bitovo: a = 1100 0101
     a<<1 = 1000 1010
```

```
printf( "%02x", a >> 1 );     // logicky posun doprava o 1 bit

> 62

Bitovo: a = 1100 0101
     a>>1 = 0110 0010
```

Vo všeobecnosti, posúvať sa dá o ľubovoľný počet bitov vľavo alebo vpravo. Reálne sú však definované posuny len o menej ako je rozsah čísla. Napr. `a >> 8` prekvapivo nemusí dať výsledok `00`, ale môže nechať pôvodné `a` (posun modulo veľkosť typu v bitoch), závisí na procesore.

Logický posun sa používa v prípade bezznamienkových čísel, aritmetický v prípade znamienkových čísel. Rozdiel je v tom, že pri logickom posune sa vždy doplní nula, pri aritmetickom posune vpravo sa doplní znamienkový bit.

```
printf( "%02x", b >> 1 );     // aritmeticky posun doprava o 1 bit

> e2

Bitovo: b = 1100 0101
     b>>1 = 1110 0010
```

Logické posuny sú užitočné, keď chcete dostať konkrétny bit na inú pozíciu. Aritmetické posuny zase realizujú rýchle násobenie a delenie mocninami dvojky.

Príklad, výpis všetkých bitov čísla od najvýznamnejšieho:

```
void binary(unsigned int number) {
     for (int k = 31; k >= 0; k--)
          printf("%i", (number >> k) & 1);
     // number sa posunie o k bitov, k-ty bit bude LSB,
     // operacia x&1 vyberie len LSB z x
}
```

Binárne bitové operácie sa dajú kombinovať s priradením, napr. `a &= b; b <<= 1;`

Bitové operácie, príznaky a enum

Bitové operácie sú užitočné vtedy, keď potrebujeme prístup k dátam na najnižšej bitovej úrovni, ale aj vtedy, keď pracujeme s logickými hodnotami, a nechceme plytvať miestom. Predstavme si napr. simuláciu šírenia infekcie, kde si chceme poznačiť o každom jedincovi, či je alebo nie je infikovaný, či sa správa zodpovedne, a či vôbec ešte žije:

```
struct _jedinec { int zije, zodpovedny, infekcia; };
```
`sizeof (struct _jedinec)` je 12

```
struct _jedinec { char zije, zodpovedny, infekcia; };
```
`sizeof (struct _jedinec)` je minimálne 3, ale môže byť až 12 kvôli zarovnaniu v pamäti

Jedna z možností je použiť bitové príznaky skombinované do jedného `int` slova.

```
#define ZIJE        0x1
#define INFEKCIA    0x2
#define ZODPOVEDNY  0x4

int jedinec1 = ZIJE | ZODPOVEDNY;
int jedinec2 = ZIJE | INFEKCIA;
```

Pomocou 2 premenných typu `int` sme definovali 2 rôznych jedincov, obaja sú nažive, jeden je zodpovedný a neinfikovaný, druhý nie je zodpovedný a je infikovaný. Konštanty (príznaky, v angličtine *flags*) sú definované tak, aby bol vždy len 1 bit nenulový (1,2,4,8,16,...). Pomocou logického operátora `|` ich vieme skombinovať.

Pomocou operátora `&` potom vieme príznaky otestovať:

```
if (jedinec & ZIJE)
     printf("Jedinec je nazive\n");

if (jedinec & (ZIJE | INFEKCIA))
     printf("Zivy jedinec je infikovany\n");
```

Pozn.: Pri testovaní je potrebné dávať pozor na prioritu (ak si nie ste 100% istý, radšej zátvorkujte!). Tiež je potrebné si dať pozor na rozdiel medzi bitovými a logickými operáciami:

```
if (jedinec && ZIJE)
     printf("Vypise sa vzdy, ak jedinec je nenulove cislo\n");
```

Pomocou operátora `&` a bitovej negácie ~ sa dajú príznaky aj vymazať:

```
jedinec &= ~ZIJE;    //jedinec umrel
```

Aj keď príznaky sa zväčša definujú formou makier (často v príslušných hlavičkových súboroch), môžeme na ich definíciu využiť enumeračný dátový typ, v jazyku C pomocou kľúčového slova `enum`.

Enumeračný dátový typ sa definuje vymenovaním konštánt. Ak sa konštante nepriradí hodnota, sú číslované zaradom od 0, inak sa použije definovaná hodnota.

Príklad:

```
enum _obdobia {JAR, LETO, JESEN, ZIMA};
enum _priznaky {ZIJE=0x1, INFEKCIA=0x2, ZODPOVEDNY = 0x4};
```

Symboly vnútri `enum` sa dajú používať ako pomenované (číselné) konštanty, a samotný `enum` typ môže byť použitý pri vytváraní premenných.

```
enum _obdobia obdobie = JAR;
```

Viac napr. tu:
https://www.programiz.com/c-programming/c-enumeration

Bitové polia, union

Niekedy chceme mať hodnoty v malom rozsahu, ale vyžadujúce viac ako 1 bit. Príklad boli 4 ročné odbobia (potrebujú 2 bity), dni v týždni (3 bity), dni v mesiaci (5 bitov). Viac takýchto údajov sa dá efektívne vložiť do a čítať z jediného celého čísla pomocou bitových operácií:

```
int hhmm = (12 << 8) | 53;     //12:53
printf("%i:%i", hhmm>>8, hhmm&0xf);
```

Elegantnejšie riešenie je použiť bitové polia (*bit fields):*
https://www.geeksforgeeks.org/bit-fields-c/

Bitové polia sa používajú podobne ako štruktúry, ale reálne sa realizujú cez bitové operácie. Preto nepodporujú niektoré pokročilejšie vlastnosti štruktúr (pointer do štruktúry, polia/pointery v štruktúre, a pod.).

Ak chcete používať tie isté dáta rôznym spôsobom (napr. raz ako int, raz ako bitové pole), alebo chcete používať pamäť, kde sa budú na to isté miesto ukladať rôzne typy údajov (nie v tom istom čase), môžete použiť typ `union`. Dobré vysvetlenie je napr. tu: https://www.learn-c.org/en/Unions

Čo ešte môže byť užitočné poznať

Počas semestra sme pokryli základné témy v oblasti programovania v jazyku C. Mohli ste si vyskúšať niektoré techniky, syntax a vybrané funkcie na zadaniach a príkladoch. Niektoré témy, ktoré sa oplatí poznať pri programovaní na nízkej úrovni vo väčšom rozsahu ako sme ich prebrali sú aj nasledujúce:

Pointer na funkciu: https://www.geeksforgeeks.org/function-pointer-in-c/

Štandardná knižnica: https://en.wikipedia.org/wiki/C_standard_library (používali sme viaceré funkcie deklarované v stdio.h, stdlib.h, string.h, math.h, ctype.h, time.h)

Vložený assembler:
https://en.wikibooks.org/wiki/Embedded_Systems/Mixed_C_and_Assembly_Programming