Bitové operace a bitové pole

Vše, co jsme doteď probírali má ekvivalenci snad ve všech vyšších programovacích jazycích. Jazyk C je však nízkoúrovňový programovací jazyk, tak nabízí programátorům i operátory, pomocí kterých mohou nepřímo pracovat i s jednotlivými bity.

1 Bitové operace

Práce s jednotlivými bity většinou vyžaduje hlubší znalost systému, například jakým stylem se ukládají čísla. Operace s jednotlivými bity však dokáže výrazně zrychlit program a některé operace ani pomocí dříve zmíněných konstrukcí není možné provést (pro ně byl nejmenší jednotkou informace byte).

V jazyce C máme k dispozici 6 operátorů:

- **bitový součin** = AND
- **bitový součet** = OR
- **bitový exkluzivní součet** = XOR
- **posun doleva**
- **posun doprava**
- **jedničkový doplněk** = NOT

Argumenty operací mohou být pouze celá čísla (signed i unsigned)

1.1 Bitový součin

`x & y`

`i`-tý bit výsledku bitového součinu `x & y` bude roven 1 pokud `i`-tý bit `x` i `y` budou rovny 1, jinak bude výsledný bit roven 0.

Výsledkem operace `100 & 50` bude 32 protože:

100	0 1 1 0 0 1 0 0
50	0 0 1 1 0 0 1 0
100 & 50	0 0 1 0 0 0 0 0

Bitový součin se často používá k výběru jen určitých bitů (nastavení některých bitů na 0). Například, pokud chceme zjistit, zda je číslo liché, stačí nám znát hodnotu nejméně významného bitu (0. bitu. V textu je nejméně významný bit vpravo).

```
#define liche(x) (1 & (x))
```

Uvědomte si rozdíl mezi bitovým a logickým součinem

```
int i = 1, j=2, k, l;

k = i && j;
l = i & j;
```

Výsledky se budou lišit. Zatímco `k` je rovno 1, protože obě `i` i `j` jsou nenulová, tak `l` bude rovno 0. A to proto, že

1	=	0 0 0 0 0 0 0 1
2	=	0 0 0 0 0 0 1 0

1.2 Bitový součet

`x | y`

`i`-tý bit výsledku bitového součtu `x | y` bude roven 1 pokud `i`-tý bit `x` nebo `y` bude roven 1. Budou-li oba nulové výsledný bit bude roven 0.

Často se používá k nastavení určitých bitů na 1 a ostatní nechá tak, jak jsou.

Použití například k převodu velkého písmena na malé.

```
#define na_mala(c) (c | 0x20)
```

`0x20` odpovídá binárně `0 0 1 0 0 0 0 0`.

1.3 Bitový exkluzivní součet

`x ^ y`

`i`-tý bit výsledku bitového exkluzivního součtu `x ^ y` bude roven 1 pokud `i`-tý bit `x` se nerovná `i`-tému bitu `y`. Budou-li oba shodné (oba rovny 0, nebo 1) výsledný bit bude roven 0.

Použití třeba při porovnání dvou čísel

```
if( x ^ y)
    /* cisla jsou rozdilna */
```

Pokud jsou `x` a `y` rozdílná, pak alespoň jeden bit bude roven 1, tedy výsledek bude nenulový.

1.4 Bitový posun doleva

`x << n`

Posune bity v `x` o `n` pozic doleva. Při tomto posunu se bity zleva ztrácí a vpravo jsou doplňovány 0.

Používá se k násobení mocninou 2.

`x = y << 1` vynásobí číslo 2

```
 1       =  0 0 0 0 0 0 0 1
 1 << 1  =  0 0 0 0 0 0 1 0
```

`x = y << 3` vynásobí číslo 3 (2^3)

Takovéto násobení je rychlejší než skutečné násobení. Než násobit číslo 80, je lepší ho vynásobit 64 a 16 a výsledky sečíst.

```
i = j * 80; //pomalejsi
```

```
i = (j << 6) + (j << 4); // rychlejsi
```

Priorita bitových operátorů je nízká, je potřeba používat závorky.

1.5 Bitový posun doprava

`x >> y`

Posune bity v `x` o `n` pozic doprava. Při tomto posunu se bity zprava ztrácí a vlevo jsou doplňovány 0. (To platí pro unsigned typy. Pro signed je výsledek implementačně závislý).

Má opačný význam než bitový posun doleva. Tedy používá se k celočíselnému dělení mocninou 2.

`x = y >> 1` vydělí číslo 2

`x = y >> 3` vydělí číslo 8

Stejně jako u bitového posunu doleva, takovéto dělení mocninou 2 je rychlejší, než klasické dělení. Priorita tohoto posunu je také nízká.

1.6 Jedničkový doplněk = negace bit po bitu

`~x`

Nulové bity nahradí 1 a 1 nulovými. Nulovány jsou i počáteční nuly, takže je výsledek závislý na typu čísla (kolik bitů obsahuje). Jiný výsledek bude pro char (8 bitů) a int (32 bitů - záleží na konkrétní implementaci).

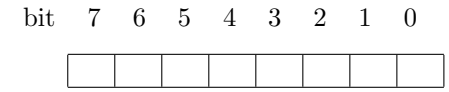
```
int main()
{
    int i = 10, i2;
    unsigned char j = 10, j2;

    i2 = ~i;
    j2 = ~j;

    printf("%i %i \n", sizeof(char), sizeof(int));
    printf("%u %i", i2, j2);
}
```

2 Práce se skupinou bitů

Často se bitové operace používají pro práci se skupinou bitů, které přestavuje například stavové slovo definované jako

`unsigned int status;`

Nejprve se definují konstanty, které určí pozice příznakových bitů ve stavovém slově. Například použijeme 3,4 a 5 bit pro příznaky číst, psát, vymazat.




```
#define READ 0x8
#define WRITE 0x10
#define DELETE 0x20
```

Nastavení příznaků:

- všechny příznaky na 1:

  `status |= READ | WRITE | DELETE;`

- příznaky pro čtení a zápis na 1:

  `status |= READ | WRITE;`

- všechny příznaky na 0:

  `status &=  (READ | WRITE | DELETE);`

- příznaky pro čtení na 0:

  `status &=  READ;`

Test, zda je příznak čtení nastaven na 0:

`if (! (status & READ ))`

3 Bitové pole

Bitové pole je struktura, která je omezená velikostí typu `int`. Nejmenší délka jedné položky v této struktuře je 1 bit. Definuje se obdobně, jako se definují struktury, pouze položky struktury nejsou určeny libovolným typem a jménem, ale jménem a délkou v bitech. Typy jednotlivých položek můžou být jen unsigned (neznaménkové) nebo signed (znaménkové) a za každým členem uvádíme za dvojtečkou počet bitů, které budou pro tento člen vyhrazeny.

Předchozí příklad řešený pomocí bitového pole:

```
typedef struct{
    unsigned zacatek : 3; // bity od 0 do 2
    unsigned read : 1; // bit 3
    unsigned write : 1; // bit 4
    unsigned delete : 1; // bit 5
} STAV;


STAV status;
```

S jednotlivými bity se pracuje pomocí tečkové notace.

- Nastavení příznaku pro četní na 1:

  `status.read = 1`

- Nastavení příznaku pro zápis na 0:

  `status.write = 0`

- Test, zda je příznak čtení nastaven na 1:

  `if(status.read)`

Položka `zacatek` je zde jen kvůli přeskočení prvních 3 bitů.

4 Cvičení

1. Z příkladu v sekci o bitovém součtu víte, že se malé písmeno a odpovídající velké liší v 5. bitu. Napište makro, které převede malé písmeno na velké.
2. Napište funkci, která pro zadané číslo `cislo` a číslo `n` zjistí hodnotu `n`-tého bitu čísla `cislo`. Například hodnota 1. bitu čísla 3 je 1, 2. bit má hodnotu 0. Bude se vám hodit operace bitového posunu.
3. Vyzkoušejte si práci se stavovým slovem (Sekce 2). Vytvořte funkci, která bude brát jako vstup celé číslo `n` a `stav` a vypíše všechna čísla od 0 do `n` a bude vynechávat násobky podle `stav`. `stav` bude uchovávat informaci, zda se mají vypisovat násobky 2, 3 a 5. Například ve `stav` je nastaven příznak, že se nemají vypisovat násobky 2, pak funkce vypíše jen lichá čísla do `n`.
4. Upravte předchozí kód tak, aby stav nastavoval uživatel po dotazu v konzoli.
5. Upravte předchozí příklad tak, aby `stav` byl definován jako bitové pole (viz Sekce 3).
6. Napište funkci, která způsobí rotaci (ne posun) čísla `x` o `n` bitů doprava. Pro číslo 3 (00000011) a `n=3` dostaneme 96 (01100000) (pracujeme-li nad char).
7. **POVINNÁ ÚLOHA - odevzdat do 8. 5.** Napište program, který bude využívat bitového pole pro úschovu času (hodiny, minuty, vteřiny). Vytvořte i funkce na převod času do bitového pole a funkci, která vypíše bitové pole jako čas.