OPERATORI PENTRU LUCRU LA NIVEL DE BIT

Limbajul **C** oferă un număr de operatori pentru lucrul la nivel de bit. Aceşti operatori se aplică numai variabilelor de tip **int** şi **char** şi conduc la expresii ale căror rezultate depind de reprezentarea internă a întregilor şi au caracteristici care depind de implementarea tipurilor cu semn.

Operatorii binari de lucru la nivel de bit sunt:

- **<<** deplasare stânga cu un anumit număr de biţi
- **>>** deplasare dreapta cu un anumit număr de biţi
- **&** **ŞI (AND)** la nivel de bit;
- **^** **SAU EXCLUSIV (XOR)** la nivel de bit;
- **|** **SAU (OR)** la nivel de bit.

În afară de aceşti operatori binari, limbajul ne mai pune la dispoziţie un operator unar care permite determinarea complemetului faţă de 1 al operandului. Acest operator este **~** şi are aceeaşi prioritate ca şi ceilalţi operatori unari existenţi în limbaj.

Operatorii **&**, **^** şi **|** sunt folosiţi, de obicei, pentru a realiza operaţiile dorite asupra primului operand în conformitate cu o configuraţie de biţi indicată de cel de-al doilea operand. Din acest motiv, de foarte multe ori, cel de al doilea operand mai este numit şi „mască” pentru că nu face decât să „mascheze” anumiţi biţi din primul operand. Aceşti operatori se aplică fiecărui bit din reprezentarea internă (binară, în virgulă fixă) a numărului.

Operatorii de deplasare << şi >>

<< Deplasează la stânga numărul din stânga operatorului cu un număr de poziţii egal cu numărul existent în dreapta operatorului.

Exemplu: rezultatul expresiei

```
n << 3
```

este valoarea numărului **n** deplasat cu 3 (trei) poziţii la stânga.

Deplasarea la stânga completează poziţiile rămase libere cu zerouri. Deplasarea cu o poziţie a numărului **n** este echivalentă cu o înmulţire cu 2 a valorii iniţiale a numărului **n** şi în exemplu dat (n<<3) expresia este echivalentă cu n * 2 * 2 * 2.

>> Deplasează la dreapta numărul din stânga operatorului cu un număr de poziţii egal cu numărul existent în dreapta operatorului.

Exemplu: rezultatul expresiei

n >> 3

este valoarea numărului **n** deplasat cu 3 (trei) poziţii la dreapta.

Deplasarea spre dreapta completează poziţiile rămase libere cu o extensie a semnului (dacă numărul este pozitiv, completarea se face cu zero; dacă numărul este negativ completarea se face cu 1). Deplasarea la dreapta cu un bit (o poziţie) este echivalentă cu o împărţire la 2 a numărului iniţial. Astfel în exemplul considerat expresia n>>3 este echivalentă cu expresia n/(2*2*2).

În cazul în care considerăm **n** fiind un întreg reprezentat pe 2 octeţi (de tip short int) şi a cărui valoare este 38, reprezentarea sa internă este următoarea:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0

După deplasarea cu trei poziţii la stânga, conţinutul celor doi octeţi asociaţi variabilei **n** are următoarea reprezentare binară:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0

Rezultatul deplasării spre dreapta cu trei poziţii a valorii iniţiale a variabilei **n** are următoarea reprezentare binară:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Operatorul & (ŞI)

Lucrează după următorul tabel de adevăr:

&	**0**	**1**
0	0	0
1	0	1

şi este folosit pentru a pune pe zero (a reseta) anumiţi biţi din variabila luată în considerare (biţii respectivi „se şterg”). Datorită acestui mod de lucru, folosirea operatorului **&** permite scoaterea în evidenţă a biţilor care ne interesează.

Exemple:

a). c = n & 0100;

pune pe zero toţi biţii lui **n** mai puţin bitul 6 (numerotarea începe de la 0 (zero) şi din dreapta) care rămâne la valoarea iniţială.

$n = 67_{10} = 2^6 + 2^1 + 2^0 = 01000011_2 = 0103_8$

$c = n \,\&\, 0100_8 = 0103_8 \,\&\, 0100_8 = 0100_8$

b). $c = n \,\&\, 0177400_8 = n \,\&\, \underbrace{11111111}_{\text{octet superior}}\ \underbrace{00000000}_{\text{octet inferior}}{}_2$;

face octetul inferior egal cu zero, cel superior rămâne neschimbat;

c). $c = n \,\&\, 0377_8 = n \,\&\, \underbrace{00000000}_{\text{octet superior}}\underbrace{11111111}_{\text{octet inferior}}{}_2$;

face octetul superior egal cu zero, cel inferior rămânând la fel ca în **n**.

Operatorul | (SAU)

Lucrează după următorul tabel de adevăr:

|	**0**	**1**
0	0	1
1	1	1

şi este folosit pentru a pune pe 1 (unu) (a seta) toţi biţii care au valoarea 1 în "mască".

Exemplu:

z = x | MASK;

$x = 66_{10}$; $MASK = 011_8$;

$z = 66_{10} \mid 011_8 = 0102_8 \mid 011_8 = 0113_8 = 75_{10}$

$x = 66_{10}$; $MASK = 077_8$;

$z = 66_{10} \mid 077_8 = 0102_8 \mid 077_8 = 0177_8 = 127_{10}$

$x = 66_{10}$; $MASK = 68_{10}$;

$z = 66_{10} \mid 68_{10} = 0102_8 \mid 0104_8 = 01000010_2 \mid 01000100_2 = 01000110_2 = 0106_8 = 70_{10}$

Operatorii pe biţi **&** şi **|** sunt diferiţi de operatorii logici **&&** şi **||**. Operatorii logici implică o evaluare de la stânga la dreapta a valorii de adevăr a expresiilor implicate, pe când operatorii care lucrează pe bit execută operaţiile ca în algebra Booleană.

Exemplu: x = 1; y = 2 → x&y = 0, pe când x&&y = 1.

Operatorul ^ (SAU EXCLUSIV – XOR)

Lucrează în conformitate cu tabelul de adevăr următor:

^	**0**	**1**
0	0	1
1	1	0

Operatorul unar ~

Aplicarea acestui operator are ca rezultat complementul faţă de 1 al operandului (biţii egali cu zero devin unu şi cei egali cu unu devin zero).

Exemplu, în instrucţiunea:

x = x & ~077;

este folosit pentru complementarea valorii octale 077 şi astfel putem realiza resetarea (punerea pe zero) a ultimilor 6 biţi ai lui x.

Exemple:

a). $x = 66_{10}$ şi $\sim 077 = 1111111111000000 = 177700_8$, deci

$x = 66_{10} \& \sim 077 = 0102_8 \& 177700_8 = 0100_8$

b) $\sim 1234 = \sim 0000010011010010_2 = 1111101100101101_2 = 0175455_8$

$\sim -1234 = \sim 1111101100101110_2 = 0000010011010001_2 = 02321_8$

$-\sim 1234 = -1111101100101101_2 = 0000010011010011 = 02323_8$

TEMA

În cele ce urmează se vor folosi numai operatori de lucru pe bit pentru toate operaţiile cerute.

Se consideră că bitul cel mai semnificativ al unui număr este bitul din stânga, iar cel mai puţin semnificativ este cel din dreapta.

TEMA 1

Problema nr. 1.1

Să se scrie un program care foloseşte numai operatorii de lucru la nivel de bit pentru a realiza „împachetarea” unei date calendaristice într-un întreg de doi octeţi (astfel biţii 0-4 reprezintă ziua, biţii 5-8 reprezintă luna, biţii 9-15 reprezintă anul numărat de la 1900) şi afişarea reprezentării binare a rezultatului.

ATENŢIE: NU se folosesc vectori.

Se va defini tipul de dată WORD ca sinonim pentru întreg scurt fără semn.

Data se citeşte sub forma: **zz/ll/aaaa**.

Se vor scrie obligatoriu funcţii pentru: împachetare dată, afişare binară.

Problema nr. 1.2

Să se scrie o funcţie care, folosind numai operatori de lucru pe bit, verifică dacă un număr natural este o putere a lui 2 (un număr natural este o putere a lui 2 dacă, în reprezentare binară, numărul are numai un bit egal cu 1). Funcţia primeşte ca parametru numărul de analizat şi returnează 1 sau 0 după cum numărul este sau nu putere a lui 2. (NU se vor folosi operatorii aritmetici "/" sau "%").

Să se scrie o funcţie care, folosind numai operatori de lucru pe bit, inversează cei doi octeţi ai unui număr întreg reprezentat pe doi octeţi. Funcţia primeşte ca parametru un număr întreg şi returnează numărul cu octeţii inversaţi. (NU se vor folosi operatorii aritmetici "/" sau "%").

Să se scrie o funcţie care primeşte ca parametri un vector de întregi şi dimensiunea acestui vector (vectorul poate avea maximum 32 de elemente). Funcţia asociază fiecărui element din vector un bit dintr-un întreg şi pune bitul respectiv pe 1 dacă elementul are o valoare pară şi pe 0 dacă elementul respectiv are o valoare impară şi returnează numărul astfel construit. (NU se vor folosi operatorii aritmetici "+", "-", "*", "/" sau "%").

Să se scrie un program care face una din prelucrările de mai sus, în funcţie de alegerea utilizatorului.

Pentru afișarea pe biți a numerelor returnate de cea de a doua și a treia funcție se va scrie o funcție de afișare a unui număr întreg bit cu bit similară cu funcția afisareBinară prezentată la curs.

Problema nr. 1.3

Să se scrie un program care realizează rotirea la dreapta sau la stânga (în funcţie de opţiunea utilizatorului) cu un anumit număr de biţi a unui număr întreg lung.

Se vor scrie funcţii de rotire dreapta, rotire stânga, şi afişare bit cu bit (afişarea se va face astfel încât bitul cel mai semnificativ să apară în partea stângă a reprezentării).

Problema nr. 1.4

Cea mai simplă verificare a corectitudinii transmiterii unui şir de biţi este să ne asigurăm că numărul biţilor egali cu 1 din şirul de biţi este par sau impar. Pentru a realiza acest lucru se ataşează respectivului şir de biţi un bit numit bit de paritate. Folosirea biţilor de paritate este cea mai simplă formă de "cod de detectare a erorilor" folosit pentru verificarea corectitudinii transmiterii unui mesaj în formă binară.

Sunt două variante de biţi de paritate: bit de paritate pară şi bit de paritate impară. Când se utilizează paritatea pară, bitul de paritate este setat pe 1 dacă numărul de biţi egali cu 1 dintr-un şir de biţi (neincluzând bitul de paritate) este impar astfel încât numărul total de biţi (incluzând bitul de paritate) să fie par. Când se utilizează paritatea impară, bitul de paritate este setat pe 1 dacă numărul de biţi egali cu 1 dintr-un şir de biţi (neincluzând bitul de paritate) este par astfel încât numărul total de biţi (incluzând bitul de paritate) să fie impar.

Paritatea pară este un caz particular de cod CRC (Cyclic Redundacy Check) care este un cod de detectare a erorilor.

Să se scrie un program care determină bitul de paritate pară asociat unui număr întreg lung citit de la tastatură şi îl înscrie în bitul cel mai semnificativ (bitul de paritate va fi 1 sau 0 după cum numărul biţilor egali cu 1 din ceilalţi 31 de biţi este impar sau par). De asemenea, operatorii de lucru pe bit se vor folosi pentru a determina dacă un număr este par sau nu (cum?). Se va afişa numărul iniţial şi reprezentarea binară a numărului cu paritatea setată.

TEMA 2

Problema nr. 2.1

Să se scrie o funcţie care are ca parametri un număr întreg şi un număr real şi returnează un număr real şi care realizează ridicarea unui număr oarecare la o putere întreagă, folosind următorul algoritm cu aplicarea operatorilor de lucru pe bit:

Observaţie: orice număr întreg se poate exprima în baza 2.

De exemplu $23_{10} = 10111_2 = 2^4 + 2^2 + 2^1 + 2^0$.

Pentru ridicarea unui număr **a** la puterea 23 se poate face următorul calcul:

$$a^{23} = a^{(16 + 4 + 2 + 1)} = a^{16} \cdot a^4 \cdot a^2 \cdot a^1$$

Ca urmare, pentru ridicarea unui număr la o putere întreagă trebuie să se genereze un şir care va include valorile **a** având ca exponenţi puteri ale lui 2, adică trebuie să se genereze următorul şir:

$$a, a^2, a^4, a^8, a^{16}, a^{32}, \ldots\ldots$$

Termenii acestui şir se generează printr-un număr relativ mic de operaţii, astfel:

```
f = a;
f = f * f;
```

Ţinând cont de observaţia de mai sus, $\mathbf{a^n}$ se poate calcula făcând apel la termenii acestui şir, alegându-i numai pe aceea ai căror exponenţi însumaţi dau puterea **n**. Pentru

aceasta se face operaţia n&1, şi dacă rezultatul este 1, primul termen este a. Se deplasează **n** spre dreapta cu o poziţie. Dacă n&1=1, se selectează a^2, s.a.m.d.

Să se scrie un program care citeşte un număr real şi un număr întreg şi afişează rezultatul ridicării numărului real la numărul întreg.

Problema nr. 2.2

Să se scrie o funcţie care folosind numai operatori de lucru pe bit calculează valoarea în octal a unui număr întreg primit ca parametru şi o depune într-un şir de caractere primit ca al doilea parametru. Numărul octal este reprezentat într-un şir de caractere (care este al doilea parametru al funcţiei) şi este precedat de caracterul 0. Se va ţine seama de faptul că o cifră octală poate fi reprezentată pe 3 biţi.

Să se scrie o funcţie care folosind numai operatori de lucru pe bit calculează valoarea în hexazecimal a unui număr întreg primit ca parametru şi o depune într-un şir de caractere primit ca al doilea parametru. Numărul hexazecimal este reprezentat într-un şir de caractere (care este al doilea parametru al funcţiei) şi este precedat de caracterele 0x. Se va ţine seama de faptul că o cifră hexazecimală poate fi reprezentată pe 4 biţi.

Să se scrie un program care citeşte de la tastatură un întreg fără semn şi folosind cele două funcţii de mai sus, afişează reprezentarea hexazecimală sau octală a numărului. Opţiunea de afişare este dată de utilizator.

Problema nr. 2.3

Să se scrie un program care determină şi afişează toate numerele prime inferioare unui număr dat (prestabilit), folosind ciurul lui Eratostene. Metoda constă din eliminarea succesivă a numerelor neprime prin parcurgerea următorilor paşi:

1. Se elimina numărul 1 (prin definiţie nu este prim).

2. Apoi se caută (pornind de la ultimul număr prim considerat - iniţial 1) primul număr neeliminat (acesta este prim) şi se elimina toţi multiplii săi (din intervalul 1-n).

3. Se repetă pasul 2 până când numărul prim considerat este mai mare decât sqrt(n) adică radical din n. Funcţia sqrt are prototipul în math.h.

Pentru rezolvare nu se vor folosi vectori, fiecărui număr natural fiindu-i asociat un bit a cărei valoare este 1 dacă numărul este prim şi 0 dacă numărul nu este prim.