Programozás II.
Segédlet az első dolgozathoz

Tartalomjegyzék

1. fejezet

Bevezető

Ez a segédlet a SZTE Programozás II. kurzusához készült. A jegyzetben található anyag az első előadásdolgozathoz készült. Az itt érintett témák nem feltétlen fedik le a dolgozat teljes anyagát, így ajánlott az előadó által közzétett weboldalak tartalmát is böngészni! A jegyzetben előfordulhatnak hibák, elírások, hiányosságok. Az ilyeneket kérem jelezzétek felém az antal@inf.u-szeged.hu címen.
A jegyzet elkészüléséhez Horváth István jegyzetét vettem alapul, ezúton is köszönet neki! A 4. fejezet Magyar Gergő munkája.
Mindenkinek sikeres felkészülést!

Szeretnénk megköszönni mindenkinek, aki 2014 őszében a jegyzet lapozása közben talált hibákat jelezte felénk!

A készítők: Antal Gábor, Magyar Gergő

A jegyzet változata: 2015. október 3.

2. fejezet

Számrendszerek közötti átváltások

A számítógépes számábrázoláshoz szükséges ismernünk azokat a műveleteket, amelyekkel átválthatjuk számainkat olyan alakra, ahogy azt a számítógép is kezeli. Minden számrendszert az alapjáról nevezünk el (pl.: kettes számrendszer alapja a 2). Továbbá minden számrendszerben a számrendszer alapja - 1 szám van (tehát a kettes számrendszerben a 0, illetve az 1 számokat használhatjuk). Azonban, mi történik akkor, amikor a számrendszerünk alapja ≥ 10 ? Ebben az esetben kerülnek bevezetésre a betűk, az 'A' betűtől kezdődően (megengedett az 'a' használata is). Az A értéke 10, a B értéke 11, C értéke 12, stb. Azonban hogyan is történik egy átváltás?

2.1. Tízes számrendszerből tetszőleges számrendszerbe

Ez viszonylag egyszerű művelet, de mindenképpen igényel egy kis gyakorlást. Az átváltáshoz az alábbi algoritmust kell alkalmazni:

1. Osszuk az eredeti számot a kívánt számrendszer alapjával.
2. Jegyezzük fel a maradékot a jobb oldalra.
3. Az osztás eredményét írjuk a szám alá.
4. Ismételjük a fenti folyamatot, amíg nem kapunk eredménynek nullát.

2.1.1. Példa

Váltsuk át a 489-et kettes számrendszerbe!

Az algoritmus:

1. Osszuk a 489-et 2-vel.
2. Jegyezzük fel a maradékot a jobb oldalra.
3. Az osztás eredményét írjuk a szám alá.
4. Ismételjük a fenti folyamatot, amíg nem kapunk eredménynek nullát.

Az eredményt a jobb oldal alulról felfele történő olvasásával kapjuk meg, tehát a 489 binárisan 111101001.

$489_{10} = 111101001_2$

489	1
244	0
122	0
61	1
30	0
15	1
7	1
3	1
1	1
0	

2.1.2. Példa

Váltsuk át a 723-at 8-as számrendszerbe!

Az algoritmus:

1. Osszuk a 723-at 8-cal.
2. Jegyezzük fel a maradékot a jobb oldalra.
3. Az osztás eredményét írjuk a szám alá.
4. Ismételjük a fenti folyamatot, amíg nem kapunk eredménynek nullát.

723	3
90	2
11	3
1	1
0	

Az eredményt a jobb oldal alulról felfele történő olvasásával kapjuk meg, tehát a 723 oktálisan 1323.
$723_{10} = 1323_8$

2.1.3. Példa

Váltsuk át a 723-at 16-as számrendszerbe!

Az algoritmus:

1. Osszuk a 723-at 16-tal.
2. Jegyezzük fel a maradékot a jobb oldalra.
3. Az osztás eredményét írjuk a szám alá.
4. Ismételjük a fenti folyamatot, amíg nem kapunk eredménynek nullát.

Az eredményt a jobb oldal alulról felfele történő olvasásával kapjuk meg, tehát a 723 hexadecimálisan 2D3.
$723_{10} = 2D3_{16}$

723	3
45	13 (=D)
2	2
0	

2.1.4. Feladatok

Váltsuk át az alábbi számokat bináris, oktális, hexadecimális számrendszerbe!
13, 254, 168, 698, 1578, 603, 4088, 5961, 15249

2.2. Tetszőleges számrendszerből tízesbe

Bármilyen számrendszerben is vagyunk, tízes számrendszerbe egyszerű lesz átváltani, hiszen csak meg kell szoroznunk a számjegyeket a következő módon: jobbról balra haladva szorozzunk be a számot a számrendszer alapjának megfelelő hatványával. A szám jobbról első számjegye jelenti a 0. hatványt, mellette balról az első hatvány, stb...

Az átváltás egy másik, gyorsabb módja lehet a Horner-elrendezés használata, amelyre ez a jegyzet nem tér ki. [1]

2.2.1. Példa

Váltsuk át a 723-at decimálisból decimális számrendszerbe:

Talán papíron gyorsabb, illetve átláthatóbb az egészet táblázat-szerűen megvalósítani

10^2	10^1	10^0
7	2	3

$$723_{10} = 3 \cdot 10^0 + 2 \cdot 10^1 + 7 \cdot 10^2$$

2.2.2. Példa

Váltsuk át a 111101001 számot binárisból decimális számrendszerbe:

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	0	1	0	0	1

$$111101001_b = 1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 0 \cdot 2^4 + 1 \cdot 2^5 + 1 \cdot 2^6 + 1 \cdot 2^7 + 1 \cdot 2^8 = 489$$

[1] A Horner-elrendezésről bővebben: `http://www.cs.elte.hu/~ewkiss/bboard/11o.n/Alg1_print_2.pdf`

2.2.3. Példa

Váltsuk át a 1323 számot oktális számrendszerből decimális számrendszerbe:
$1323_8 = 3 \cdot 8^0 + 2 \cdot 8^1 + 3 \cdot 8^2 + 1 \cdot 8^3 = 723$

2.2.4. Példa

Váltsuk át a 2D3 számot hexadecimálisból decimális számrendszerbe:
$2D3_h = 3 \cdot 16^0 + 13 \cdot 16^1 + 2 \cdot 16^2 = 723$

2.2.5. Feladatok

Váltsuk át az alábbi számokat tízes számrendszerbe:

- hexadecimális számok: 0f36 88 832c 31 9e8a82 2c2f 080a8a 48cd de 56 da 713 cbe e05322 334cc
- oktális számok: 214, 251, 265, 342 346 151 116 1331 024 130
- bináris számok: 01100101001, 01110011, 1010010, 10011, 01001010, 00000001, 11101, 1101001 0111100111, 010110110

2.3. Nem tízesből nem tízesbe

Ha ilyen feladatot kapunk, akkor az egyik módszer az, hogy először a számot átváltjuk 10-es számrendszerbe, majd onnan a kívánt számrendszerbe, az előzőleg bemutatottaknak megfelelően. Ez elég időigényes. Ha 2 hatvány számot között kell átváltani (2-es, 8-as, 16-os) az átváltásnak van egy egyszerűbb módja is.

2.3.1. Kettesből nyolcasba

Mivel, 8 a kettő harmadik hatványa ($8 = 2^3$), ezért az eredeti kettes számrendszerbel szám számjegyeit hármasával csoportokba fogjuk, majd az így kapott csoportok értékeit kiszámoljuk. Ez fogja megadni a szám 8-as számrendszerbeli alakját.

2.3.1.1. Példa

$1011010011_2 = 1|011|010|011$

Majd ezeknek a csoportoknak kiszámoljuk az értékét:

2^0	2^2	2^1	2^0	2^2	2^1	2^0	2^2	2^1	2^0
1	0	1	1	0	1	0	0	1	1
1		3			2			3	

Tehát: $1011010011_2 = 1323_8$

2.3.1.2. Példa

Váltsuk át a 11101101 bináris számot nyolcas számrendszerbe! Kezdjük a hármasokra bontással, az alábbi módon: $11101101_2 = 11|101|101$

Majd pedig számoljuk ki a csoportok értékét, ami megadja a szám 8-as számrendszerbeli alakját.

2^1	2^0	2^2	2^1	2^0	2^2	2^1	2^0
1	1	1	0	1	1	0	1
3			5			5	

Tehát: $11101101_2 = 355_8$

2.3.2. Nyolcasból kettesbe

Az előző módszer fordítottja. A nyolcas számrendszerbeli szám számjegyeit egyenként visszaváltjuk 3 jegyből álló, kettes számrendszerbeli megfelelőjükre. Ez újfent nem túl bonyolult feladat, de érdemes gyakorolni, hogy a kettő hatványai gyorsan menjenek.

2.3.2.1. Példa

Váltsuk át az 1323_8 számot kettes számrendszerbelire!

1	3	2	3
001	011	010	011

$1323_8 = 001011010011_2$

A szám elején lévő nullákat elhagyhatjuk (mivel nincs jelentőségük), így máris szebb lesz: $1323_8 = 1011010011_2$

2.3.3. Kettesből tizenhatosba

Mivel, 16 a kettő negyedik hatványa ($16 = 2^4$), ezért az eredeti kettes számrendszerbel szám számjegyeit négyesével csoportokba fogjuk, majd az így kapott csoportok értékeit kiszámoljuk. Ez fogja megadni a szám 16-os számrendszerbeli alakját.

2.3.3.1. Példa

$1011010011_2 = 10|1101|0011$

Majd ezeknek a csoportoknak kiszámoljuk az értékét:

2^1	2^0	2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
1	0	1	1	0	1	0	0	1	1
2				13				3	

Tehát: $1011010011_2 = 2D3_{16}$

2.3.3.2. Példa

Váltsuk át a 11101101 bináris számot hexadecimális számra! Kezdjük a négyesekre bontással, az alábbi módon: $11101101_2 = 1110|1101$

Majd pedig számoljuk ki a csoportok értékét, ami megadja a szám 16-os számrendszerbeli alakját.

2^1	2^0	2^2	2^1	2^0	2^2	2^1	2^0
1	1	1	0	1	1	0	1
14 (E)				13 (D)			

Tehát: 11101101_2 =ED$_{16}$

2.3.4. Tizenhatosból kettesbe

Az előző módszer fordítottja. A hexadecimális szám számjegyeit egyenként visszaváltjuk 4 jegyből álló, kettes számrendszerbeli megfelelőjükre.

2	D (13)	3
010	1101	0011

$2D3_{16} = 01011010011_2$

3. fejezet

Bitenkénti műveletek

Általános érvényű szabály, hogyha a kapott számok nem kettes számrendszerben vannak, akkor azzal kell kezdenünk, hogy átváltjuk kettes számrendszerbeli számmá. (Erre már rengeteg példát láttunk.) Ha kettes számrendszerbeli számok vannak, akkor nincs már más hátra: el kell végezni a műveletet.

3.1. Negálás

Jelölése: Minden bitet az ellenkezőjére módosítunk. Ahogy az igazságtáblában is láthattuk, 0-ból 1 lesz, 1-ből pedig 0.

3.1.1. Példa

Negáljuk a 1101011-et! Mindenhol, ahol egyest látunk, írjuk át nullára. Ahol nullát látunk, azt írjuk át egyesre. $1101011 = 0010100$

3.1.2. Példa

Negáljuk a 1111000010101000-et! Mindenhol, ahol egyest látunk, írjuk át nullára. Ahol nullát látunk, azt írjuk át egyesre. $1111000010101000 = 0000111101010111$

3.1.3. Feladatok

Negáljuk a következő számokat: 101101011010011_2, 230_{10}, 1011011_2, $CB1_h$, 11300_d, 2653_8 , $9F62_h$, 1265_8, 3625_{10}

3.2. Bitenkénti AND

Jelölése: &
Két szám bitenkénti és-elésén értjük azt, amikor veszünk két számot, és bitenként végrehajtjuk az AND (és) műveletet az AND igazságtáblája alapján. Megjegyzés: Amennyiben a két szám hossza eltér, akkor a rövidebb szám elé tetszőleges nullákat írhatunk, hiszen az a szám értékén nem változtat.

3.2.1. Példa

Legyen az egyik szám 12, a másik 9. Mivel egyik szám sem kettes számrendszerbeli, ezért először váltsuk át őket: $12_{10} = 1100_2$ $9_{10} = 1001_2$

A számokat sikerült megfelelő alakra hozni, hajtsuk rajta végre az AND műveletet:

$$\begin{array}{ll} & 1100 \\ \& & 1001 \\ \hline & 1000 \end{array}$$

3.2.2. Példa

Hajtsuk végre a bitenkénti és műveletet az alábbi két számon: 10111011_2, 1101111_2

$$\begin{array}{ll} & 10111011 \\ \& & 01101111 \\ \hline & 00101011 \end{array}$$

3.2.3. Feladatok

Hajtsuk végre a bitenkénti és műveletet az alábbi számokból vett tetszőleges számpáron: 359_{10}, 0100111111_2, 3571_8, 3521_{10}, $6\text{DC}5_h$, 243_{10}, 1111111_2, 42456_8, $\text{ABC}4_{16}$

3.3. Bitenkénti OR

Jelölése: |
Két szám között végrehajtjuk a bitenkénti OR műveletet (vagy). Az OR művelet végrehajtásának módját láthattuk az OR művelet igazságtáblájánál.

3.3.1. Példa

Legyen az egyik szám 12, a másik 9. Mivel egyik szám sem kettes számrendszerbeli, ezért először váltsuk át őket: $12_{10} = 1100_2$ $9_{10} = 1001_2$

A számokat sikerült megfelelő alakra hozni, hajtsuk rajta végre az OR műveletet:

$$\begin{array}{ll} & 1100 \\ | & 1001 \\ \hline & 1101 \end{array}$$

3.3.2. Példa

Hajtsuk végre a bitenkénti vagy műveletet az alábbi két számon: 10111011_2, 1101111_2

```
  10111011
| 01101111
----------
  11111111
```

3.3.3. Feladatok

Hajtsuk végre a bitenkénti vagy műveletet az alábbi számokból vett tetszőleges számpáron: 359_{10}, 0100111111_2, 3571_8, 3521_{10}, $6DC5_h$, 243_{10}, 1111111_2, 42456_8, $ABC4_{16}$

3.4. Bitenkénti XOR

Két szám között végrehajtjuk a bitenkénti XOR műveletet, azaz a kizáró vagy műveletet, a XOR igazságtáblájánál látottak szerint.

3.4.1. Példa

Legyen az egyik szám 12, a másik 9. Mivel egyik szám sem kettes számrendszerbeli, ezért először váltsuk át őket: $12_{10} = 1100_2$ $9_{10} = 1001_2$

A számokat sikerült megfelelő alakra hozni, hajtsuk rajta végre az XOR műveletet:

```
      1100
XOR   1001
----------
      0101
```

3.4.2. Példa

Hajtsuk végre a bitenkénti kizáró vagy műveletet az alábbi két számon: 10111011_2, 1101111_2

```
      10111011
XOR   01101111
-------------
      11010100
```

3.4.3. Feladatok

Hajtsuk végre a bitenkénti kizáró vagy műveletet az alábbi számokból vett tetszőleges számpáron: 359_{10}, 0100111111_2, 3571_8, 3521_{10}, $6DC5_h$, 243_{10}, 1111111_2, 42456_8, $ABC4_{16}$

3.5. Negatív számok ábrázolása - kettes komplemens képzés

Ha egy változónak nem adjuk meg, hogy az unsigned legyen, ha negatív értéket adunk neki, akkor azt a számítógépes számábrázolásban valahogyan jelölni kell. Erre a célra vezették be először az előjelbitet. 0 jelenti a pozitív, 1 a negatív számot. Így azonban a 0 értéket kétféleképpen is

ábrázolhatjuk (+0, -0). Ez problémákhoz vezet, illetve így a kivonás műveletet is elég bonyolult lett volna implementálni, így kerültek bevezetésre az egyes, illetve kettes komplemens képzés. A negatív számok ábrázolására kettes komplemenst használunk. A művelet egyszerű:

- Az eredeti szám minden bitjét negáljuk (ez az egyes komplemens)
- Adjunk hozzá az így kapott számhoz egyet. (kettes komplemens)

Ennyi, elkészültünk.

A visszaváltás módja hasonló, csak az előző művelet fordítottját kell megvalósítani: a kettes komplemensből kivonunk egyet, majd minden bitet negálunk.

3.5.1. Példa

Ábrázoljuk a kettes komplemens módszerével a -723 számot! Természetesen itt is szükséges az átváltás, amennyiben nem bináris számot kapunk! $723_{10} = 1011010011_2$ Ha ezt 32 biten tároljuk, akkor az így néz ki: 00000000000000000000001011010011

- Negáljuk a szám minden bitjét:
  00000000000000000000001011010011 = 11111111111111111111110100101100 (kész az egyes komplemens)
- Adjunk hozzá egyet, ezáltal képezve a kettes komplemenst:
  11111111111111111111110100101101

Ezáltal meg is van a -723 gépi ábrázolása.

3.5.2. Példa

Ábrázoljuk a kettes komplemens módszerével a -72 számot! Természetesen itt is szükséges az átváltás, amennyiben nem bináris számot kapunk! $72_{10} = 1001000_2$ Ha ezt 32 biten tároljuk, akkor az így néz ki: 00000000000000000000000001001000

- Negáljuk a szám minden bitjét:
  00000000000000000000000001001000 = 11111111111111111111111110110111 (kész az egyes komplemens)
- Adjunk hozzá egyet, ezáltal képezve a kettes komplemenst:
  11111111111111111111111110111000

Ezáltal meg is van a -72 gépi ábrázolása.

3.5.3. Feladatok

Ábrázoljuk a kettes komplemens módszerével az alábbi számokat: -63, -123, -765, -999, -1035, -1563, -4792, -6821

3.6. Bitléptetés - előjeltelen számok

A bitléptetés művelete gyakorlatilag megegyezik a szorzás/osztás műveletével. A balra való léptetés megfelel kettővel való szorzásnak, míg a jobbra léptetés megfelel kettővel való osztásnak. Minden esetben, ha egy új szám jön be, az a 0 lesz.

3.6.1. Balra léptetés

szám « x
Assembly-ben: SHL (SHift Left) Ebben az esetben balra tolunk, méghozzá x bitnyit. Vegyük például a 11011 számot.

3.6.1.1. Példa

11011 « 4 (Ez azt jelenti, hogy 4 bittel "balra toljuk" a számot.)
A szám értéke eltolás előtt: 00000000000000000000000000011011
A szám értéke eltolás után : 00000000000000000000000110110000
Ilyenkor a szám bal oldalán lévő számjegyek elvesznek, a jobb oldalon pedig nullák jöttek be.

3.6.1.2. Példa

723 « 2
A 723 decimális szám bináris alakban, 32 biten ábrázolva: 00000000000000000000001011010011
Léptessük balra kettővel: 00000000000000000000101101001100
Ilyenkor a szám bal oldalán lévő számjegyek elvesznek, a jobb oldalon pedig nullák jöttek be.

3.6.2. Jobbra léptetés

szám » x
Assembly-ben: SHR (SHift Right) Ezesetben a biteket jobbra toljuk. Vegyük például a 000001110110 számot.

3.6.2.1. Példa

000001110110 » 4
A szám értéke eltolás előtt: 000001110110 (118)
A szám értéke eltolás után : 000000000111 (7) Ami ekvivalens a $118/2/2/2/2$, természetesen törtrész nélkül számolva.

3.6.2.2. Példa

A 723 decimális szám bináris alakban, 32 biten ábrázolva: 00000000000000000000001011010011
Léptessük jobbra, hárommal: 00000000000000000000000001011010
(Jobb oldalról "elveszik" 3 darab bit. Bal oldalról viszont bejön 3 darab 0). || Így $723 >> 3 = 90$
(Az eredmény ugyanaz, mint $723/2/2/2$, természetesen törtrész nélkül számolva)

3.7. Bitléptetés - előjeles számok

A előjeles bitléptetés lényege hasonló az előjeltelen számokkal való léptetéshez. Viszont, ha itt is ugyanazt a módszert alkalmaznánk, az eredeti előjel jobbra léptetésnél eltűnne. Ezért az előjeles számok bitléptetéséhez külön gépi utasítások léteznek. Ezeket C/C++ szintjén ezt nem érzékeljük, azonban más Assembly utasításnak felelnek meg az alábbiak:

3.7.0.3. Balra léptetés

negatív szám « x
Assemblyben SAL (Shift Arithmetic Left) A balra léptetés pontosan ugyanúgy működik előjeles szám esetén is, mint előjeltelennél!

3.7.0.4. Példa

-723 « 2
A 723 bináris értékét már kiszámoltuk korábban. Ez 32 biten ábrázolva:
00000000000000000000001011010011

Mivel negatív számról van szó, ezért vennünk kell a kettes komplemensét a számnak:
11111111111111111111110100101101

Léptessük balra ezt a számot kettővel: 11111111111111111110100101101100

Az eredmény tehát: $-723 << 2 = -2892$ Ez megegyezik a $-723 \cdot 2 \cdot 2$ értékével.

3.7.0.5. Jobbra léptetés

negatív szám » x
Assemblyben SAR (Shift Arithmetic Right)

3.7.0.6. Példa

-723 » 3
Az előbbi példánál láthattuk, hogy a -723 szám értéke: 11111111111111111111110100101101 Ezt az előbbiekhez hasonlóan nem tolhatjuk el jobbra, mivel a szám elvesztené az előjel információit. Ezért, ha előjeles számról van szó, gépi szinten a SAR utasítás hívódik meg, mely balról egyest fog behozni. Így a 3-mal való jobbra tolás eredménye: 11111111111111111111111110100101 Vagyis, jobbról elvesztettünk 3 bitet, és balról 1-esek jöttek be. Az eredmény tehát: $-723 >> 3 = -91$

Megjegyzés: Látható, hogy előjeltelen 723 esetén a jobbra 3-mal eltolás eredménye 90 volt. Itt az eredmény -91, vagyis abszolút-értékben 1 az eltérés. Ezt a kettes komplemens számábrázolás okozza. Az eredmény ugyanaz, mint $723/2/2/2$, de itt az előjeltelen példához képest nem lefelé "kerekítődött" a szám értéke, hanem felfelé, így lett az eredmény -91 lett.

3.7.0.7. Feladatok

Oldjuk meg az alábbi feladatokat: $765 >> 6$
$269 << 3$
$-765 >> 6$

$-191 >> 4$
$3869 >> 2$
$2649 << 5$
$-3899 >> 2$
$-1069 >> 5$

4. fejezet

Alul -, és túlcsordulás

Itt annyit kell tudni, hogy a számok egy számkörön vannak ábrázolva, tehát ha túl- vagy alulcsordul, akkor az ábrázolt szám legkisebb vagy legnagyobb értékét veszi fel.

4.1. Alulcsordulás

Dolgozatban az alábbi példa képzelhető el:

```
unsigned int a = -5;
int b = ((~(a << 5)) & (a << 2)) | (a << 3);
std::cout << b << std::endl;
```

Milyen szám íródik ki a képernyőre, ha 32-biten ábrázoljuk?

```
uint a = -5;
Ekkor az a-nak a bináris értéke: 1111 1111 1111 1111 1011
int b = ((~(a << 5)) & (a << 2)) | (a << 3);
a << 5
1111 1111 1111 0110 0000
~(a << 5)
0000 0000 0000 1001 1111
a << 2
1111 1111 1111 1110 1100
a << 3
1111 1111 1111 1101 1000

  0000 0000 0000 1001 1111
& 1111 1111 1111 1110 1100
--------------------------
  0000 0000 0000 1000 1100
| 1111 1111 1111 1101 1000
--------------------------
  1111 1111 1111 1101 1100

-36 //2-es komplemens!!!
```

Egy gyors példa, hogyan is működik ez a 2-es komplemens. (Negatív számoknál használjuk) Legyen a fent kiszámolt 1111 1111 1111 1101 1100 bináris szám. Ennek a legmagasabb helyiértéken szereplő bitje az előjel bit.

```
  1 111 1111 1111 1101 1100
Algoritmus:
  1. lépés: Vedd az általad kiszámolt bináris szám negáltját,
            itt a fenti példát tekintjük annak:
      0 000 0000 0000 0010 0011
  2. lépés: Add hozzá ezt:
      0 000 0000 0000 0000 0001
  3. lépés: Végül 1. lépés + 2. lépés
        0 000 0000 0000 0010 0011
      + 0 000 0000 0000 0000 0001
      ---------------------------
        0 000 0000 0000 0010 0100

Ez így 36.
```

Attention! Fent az a értéke negatív volt! Ezért $36 \cdot (-1) = -36$

5. fejezet

Igazságtáblák

5.1. NOT

NOT - negálás, tagadás. Az eredeti érték ellentettjét képzi.
Jelölése C-ben: ! (logikai művelet), ~(bitenkénti művelet)
Igazságtáblája:

bemenet	kimenet
A	NOT A
0	1
1	0

5.2. OR

OR - vagy művelet. Akkor igaz, ha LEGALÁBB az egyik feltétel igaz.
Jelölése C-ben: | (bitenkénti), || (logikai)
Igazságtáblája:

bemenet		kimenet
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

5.3. XOR

XOR - kizáró vagy művelet. Akkor igaz, ha PONTOSAN az egyik feltétel igaz.
Jelölése C-ben: ^(bitenkénti)
Igazságtáblája:

bemenet		kimenet
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

5.4. AND

AND - és művelet. Csak és kizárólag akkor igaz, ha mindkét feltétel igaz.
Jelölése C-ben: & (bitenkénti), && (logikai)
Igazságtáblája:

bemenet		kimenet
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

5.5. NAND

NAND - negált és művelet. Pontosan akkor NEM igaz, ha mindkét feltétel teljesül.
Igazságtáblája:

bemenet		kimenet
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

5.6. NOR

NOR - negált vagy művelet. Pontosan akkor igaz, ha egyik feltétel sem teljesül.
Igazságtáblája:

bemenet		kimenet
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

5.7. XNOR

XNOR - negált kizáró vagy művelet. Pontosan akkor teljesül, ha vagy egyik feltétel sem teljesül, vagy pedig mindkettő teljesül.
Igazságtáblája:

bemenet		kimenet
A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

6. fejezet

Logikai műveletek

Megjegyzés: A C/C++ nyelvcsaládban a logikai műveletek kiértékeléséhez úgynevezett short-circuit evaluationt használnak, ami gyakorlatilag annyit jelent, hogy a kifejezést csak addig értékeljük ki, amíg egyértelművé nem válik az értéke. Például, ha van egy logikai kifejezés, ami úgy kezdődik, hogy "true || bármi"', akkor az első true után befejeződik a kiértékelés, hiszen "'igaz vagy bármi más"' az mindenképpen igaz lesz.

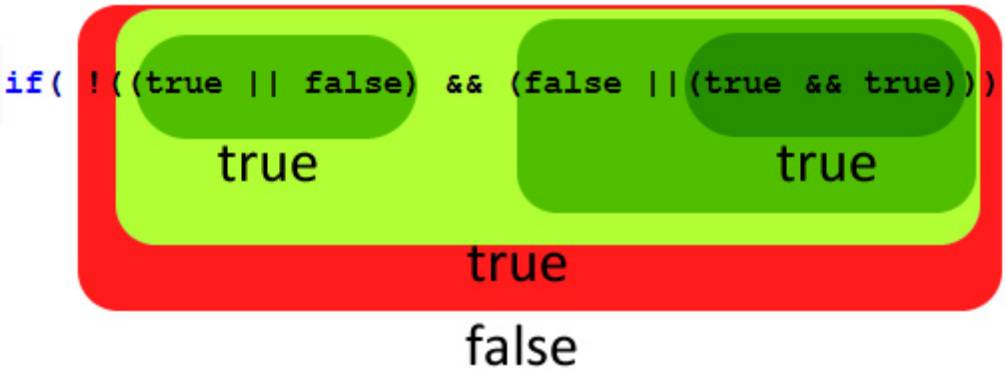
6.1. Végrehajtásukról

A logikai műveleteknél csak az igazságtáblát kell ismernünk, és könnyű dolgunk lesz. A teljes kifejezés kiértékeléséhez a részkifejezéseket kell kiértékelnünk. Amennyiben a részkifejezéseken belül is vannak kifejezések, akkor ezt a műveletet iteráljuk. Tehát minden esetben a legegyszerűbb kifejezés kiértékelésével kezdenünk, majd belülről haladni kifelé. Tekintsünk néhány példát!

A true értékét megfeleltethetjük 1-gyel, a false értékét pedig 0-val.

6.2. Példa

Vegyük a következő if kifejezést: if(!((true || false) && (false ||(true && true)))
Kezdjük el rajta végrehajtani a logikai műveleteket! Mindig a legbelső részkifejezést értékeljük ki, majd haladjunk kifelé. Így megkaphatjuk a teljes kifejezés értékét.
Kezdjük a $(true||false)$ kiértékelésével, valamint a (true && true) kiértékelésével. Ezután, az eredményt behelyettesíthetjük:
if(!((true) && (false ||(true)))
Haladjunk tovább, a (false || true) kiértékelésével:
if(!((true) && (true))
Haladjunk tovább a (true && true) kiértékelésével:
if(!((true))
Végül jöjjön a legkülső kifejezés, a negálás:
if(false)
Eljutottunk az if-ben lévő logikai kifejezés teljes kiértékeléséhez, amely false lesz. A kiértékelés szemléletes látható az alábbi ábrán:



6.3. Példa

A fentieket alkalmazva hajtsuk végre a következő if-ben lévő kifejezés kiértékelését:
if((true && false) || (false && (false || ((false && true) || true))))
if((false) || (false && (false || ((false) || true))))
if((false) || (false && (false || (true))))
if((false) || (false && (true)))
if((false) || (false))
if((false))

6.4. Példa

A fentieket alkalmazva hajtsuk végre a következő if-ben lévő kifejezés kiértékelését:
if(!(true) || !(false && (true || false)))
if(!(true) || !(false && (true)))
if(!(true) || !(false))
if(false || true)
if(true)

6.5. Feladatok

if((true && false) && (false || (false || ((false && true) || true))))
if(!(true) && !(false || (true || false)))
if((true || !(false && (false || true))) && false)
if((!(false || (true || false)) || !(false && false)) && false)

6.6. Összetettebb példa

Ezek egyszerű példák voltak, amelyekben nem fordultak elő változó értékek. És mit tegyünk, ha esetleg olyan kifejezést kapunk, amelyben nem true/false értékek vannak, hanem változók? Aggodalomra természetesen nincs ok, mindössze az egész kifejezés igazságtábláját kell felírnunk, a változók lehetséges értékei mellett. Megjegyzés: minden ilyen esetben 2^n darab sornak kell lennie a táblázatban (2 változó esetén négy sor, 3 esetén nyolc, stb...). A változók lehetséges értéke igaz (jelölhetjük i-vel, t-vel, vagy simán 1-gyal), vagy hamis (h, f, vagy 0) lehet. A példákban én számokkal fogom reprezentálni ezeket az értékeket.

Értékeljük ki a $(a||!b)\&\&(!a||!b)$ logikai kifejezést! Ehhez fel kell írnunk az igazságtáblákat, a változók lehetséges értékeivel (ezeknek külön-külön oszlop szükséges), valamint a teljes kifejezés értékének is szükséges lesz egy oszlop. (Aki nem biztos magába, az a tanult módon felírhatja a rekurzívan, belülről kifele haladva a részkifejezések értékét is.)

a	b	(a ‖ !b) && (!a ‖ !b)
0	0	1
1	1	0
0	1	0
1	0	1

A megoldás a táblázat utolsó oszlopában van, a bal oldalt található lehetséges értékek mellett.

6.7. Összetettebb példa

A következő példában értékeljük ki a $a||(!c\&\&b)$ kifejezést a változók lehetséges értékei mellett. Ehhez ismét fel kell írnunk a változók lehetséges értékeit egy táblázatba. Itt már három változónk van, tehát $2^3 = 8$ sort kell készítenünk Mivel itt már három változó is van, ezért talán célszerű lehet (főleg kevesebb gyakorlattal) a részkifejezéseket is felírnunk. Ezt talán úgy a legcélravezetőbb, ha felírjuk a változók lehetséges értékeit, majd felírjuk a részkifejezéseket, majd pedig az egész kifejezést, és mindig a szükséges oszlopokban lévő értékek között hajtjuk végre a műveletet.

A példában először a változókat írtam fel, majd a jobb oldalon található (egyedüli) részkifejezést. Végül pedig a teljes kifejezést írtam fel, aminek értékét az előtte lévő részkifejezés oszlopban, illetve az az "‘a"’ oszlopában található értékek között elvégzett logikai vagy művelet segítségével írtam fel.

a	b	c	(!c && b)	a ‖ (!c &&b)
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
1	0	0	0	1
1	1	0	1	1
1	0	1	0	1
0	1	1	0	0
1	1	1	0	1

A megoldás a táblázat utolsó oszlopában van, a bal oldalt található lehetséges értékek mellett.

6.8. Összetettebb példa

Értékeljük ki a $(!a\&\&b)||(c\&\&(!b||a))$ kifejezés értékét! Ehhez ismét fel kell írnunk a változók lehetséges értékeit egy táblázatba. Itt már három változónk van, tehát $2^3 = 8$ sort kell készítenünk. Majd ismét felírhatunk részkifejezéseket, amennyiben nem vagyunk biztosak a dolgunkban, végül pedig a teljes kifejezés értékét is fel kell írnunk, amelyeket a megfelelő részkifejezések oszlopában található értékeken elvégzett logikai és művelet segítségével írtam fel.

a	b	c	(!b ‖ a)	(c && (!b ‖ a)	(!a && b)	(!a && b) ‖ (c && (!b ‖ a))
0	0	0	1	0	0	0
0	0	1	1	1	0	1
0	1	0	0	0	1	1
1	0	0	1	0	0	0
1	1	0	1	0	0	0
1	0	1	1	1	0	1
0	1	1	0	0	1	1
1	1	1	1	1	0	1