Dolgozatok:

Olvasson be a 0 végjelig hónap sorszámokat. A program írja ki, hogy a hónap melyik évszakban található és számolja meg és írja ki, hogy hány téli hónapot adtunk meg.

```
#include <stdio.h>
#include <iostream>
int main(){
	setlocale (LC_ALL, "");
	int ho, telszam = 0;
	printf("Kérek egy hónap sorszámot: "); scanf("%d", &ho);
	while(ho){
		switch(ho){
			case 1: case 2: case 12: printf("tél\n"); telszam++; break;
			case 3: case 4: case 5: printf("tavasz\n"); break;
			case 6: case 7: case 8: printf("nyár\n"); break;
			case 9: case 10: case 11: printf("ősz\n"); break;
			default: printf("\n*** Hibás hónapszám ***\n\n");    break;
		}
		printf("Kérek egy hónap sorszámot: "); scanf("%d", &ho);
	}
	printf("\nA téli hónapok száma: %d\n", telszam);
}
```

A program kérjen be karaktereket EOF (ctrl z) végjelig. Határozza meg, hogy hány darab kisbetű, hány darab nagybetű és hány darab szám karaktert adtunk meg.

```
#include <stdio.h>
#include <iostream>
int main(){
	setlocale (LC_ALL, "hun");
	char c;
	int nagyb=0, kisb=0, szam=0, egyeb=0;
	printf("Kérek karaktereket EOF-ig:\n");
	while((c=getchar())!=EOF){
		if(c>='0'&&c<='9') szam++;
		else if(c>='a'&&c<='z') kisb++;
			else if(c>='A'&&c<='Z') nagyb++;
				else egyeb++;
	}
	printf("A karakterek száma:\nszám: %d\nkisbetű: %d\nnagybetű: %d\negyéb: %d\n",
		szam, kisb, nagyb, egyeb);
}
```

A program kérjen be tetszőleges számú hőmérsékletet addig, amig az abszolút 0 foknál (-273,16°C) hidegebb hőmérsékletet nem adunk meg. Határozza meg, hogy hányszor adtunk meg fagypont alatti, illetve forráspont feletti hőmérsékletet.

```
#include <stdio.h>
#include <iostream>
int main(){
	setlocale(LC_ALL, "hun");
	float hom;
	int felett = 0, alatt = 0;
	do{
		printf("Kérem a homérsékletet: "); scanf("%f",&hom);
		if (hom > 100) felett++;
		else if (hom < 0) alatt++;
	} while (hom >= -273.16);
	printf("Fagypont alatti hőmérséklet: %d\nForráspont feletti hőmérséklet: %d\n",alatt-1, felett);
}
```

Számrendszerek

Minden nem negatív szám felírható a következő alakban:

$$a_k \cdot 10^k + a_{k-1} \cdot 10^{k-1} + \ldots + a_1 \cdot 10 + a_0 + a_{-1} \cdot 10^{-1} + \ldots + a_{-l} \cdot 10^{-l}$$

ahol: k, j = 0, 1, 2, … és a_k … a_0 … a_{-l} 9-nél nem nagyobb, nem negatív, egész szám.

Ugyanígy egyértelműen felírható ez a szám kettes számrendszerben is:

$$b_m \cdot 2^m + b_{m-1} \cdot 2^{m-1} + \ldots + b_1 \cdot 2 + b_0 + b_{-1} \cdot 2^{-1} + \ldots + b_{-n} \cdot 2^{-n}$$

ahol: m, n = 0, 1, 2, … és b_m … b_0 … b_{-n} értéke 0 vagy 1.

A félreértések elkerülése érdekében a számrendszer alapszámát indexben jelezzük. Például:

$11_{10} = 1011_2$ vagy $1010_2 = 10_{10}$

A kettes számrendszerbeli (bináris) szám egy helyi értékét bitnek nevezzük. (Bit: az angol binary digit szavak összevonásából származik, de „pici”, „falat” jelentése is van.) Ennek megfelelően a fenti képlettel felírt bináris szám ábrázolására m+n+1 bit szükséges.

A **bit** az informatikában használatos legkisebb információhordozó egység, az informatika alapegysége, értéke: 0 vagy 1.

A tízes számrendszerben ezresével csoportosíthatjuk a számokat és ezeknek a nagyságrendeknek külön nevet adhatunk. A bináris számoknál is ezt a hagyományt követik, bár itt nem pontosan ezerszeres a váltószám. Például: 1 kg = 10^3 g, de 1 kbit (kilobit) = 2^{10} = 1024 bit. Ennek megfelelően felírható a következő táblázat:

- 1024 bit = 1 kbit (kilobit) = 2^{10} bit
- 1024 kbit = 1 Mbit (megabit) = 2^{20} bit
- 1024 Mbit = 1 Gbit (gigabit) = 2^{30} bit
- 1024 Gbit = 1 Tbit (terabit) = 2^{40} bit

A nyolcbites (nyolcjegyű) bináris számot byte-nak (bájt) nevezzük. A bájt a digitális információfeldolgozás alapegysége. A memóriák és a háttértárak kapacitását is bájtban mérik. Az előbbihez hasonlóan itt is használatos a kB (kilobájt), MB (megabájt) stb. elnevezés.

Bináris számrendszerben a számok sokkal hosszabbak, mint a tízes számrendszerben. Láttuk, hogy $10^3 \approx 2^{10}$, ami azt jelenti, hogy 10 helyi érték hosszúságú bináris szám felel meg 3 helyi értékű decimális számnak. A könnyebb kezelhetőség érdekében bevezették a nyolcas (oktális) és a tizenhatos (hexadecimális) számrendszereket. Azért éppen ezeket, mert a számrendszerek között egyszerű az átváltás ($8 = 2^3$; $16 = 2^4$).

Az oktális számrendszer előnye, hogy csak olyan jelöléseket (alaki értékeket) használunk, amelyek ismertek a tízes számrendszerben. Továbbá a nyolc közel áll a tízhez, így a nyolcas számrendszerben történő ábrázolás nem sokkal hosszabb, mint a tízes számrendszerbeli.

A kettes és a nyolcas számrendszerek közötti átváltás egyszerűen elvégezhető. A bináris számot jobbról bithármasokba (triádokba) csoportosítjuk, egy-egy csoport egy oktális számjegynek felel meg. A visszaalakítás is egyszerű, mivel a nyolcas számrendszerben megadott szám egy számjegye megfelel egy bináris triádnak.

Az oktális számrendszer hátránya, hogy egy bájt nem egész számú triádból áll, ami a programozásnál jelent némi gondot. Ezt a hátrányt küszöböli ki a tizenhatos (hexadecimális) számrendszer.

A hexadecimális számok és a bináris számok között is könnyű a konverzió, csak itt négyes csoportokat kell alkotni. További előny, hogy egy bájt értéke felírható két hexadecimális számjegy segítségével (8 biten $2^8 = 16^2$ különböző érték jeleníthető meg). Hátránya, hogy a tizenhatos számrendszerben 16 különböző jelre van szükség. Mivel a tízes számrendszerben tíz (0, 1, 2,…9) darab jelünk van a számok ábrázolására, így a tíztől a tizenötödik elem jelölésére új szimbólumokat kellett bevezetni:

$10_{10} = A_{16}$; $11_{10} = B_{16}$; $12_{10} = C_{16}$; $13_{10} = D_{16}$; $14_{10} = E_{16}$; $15_{10} = F_{16}$;

Változók

Változó lehet: egyszerű és összetett (többszörösen összetett).

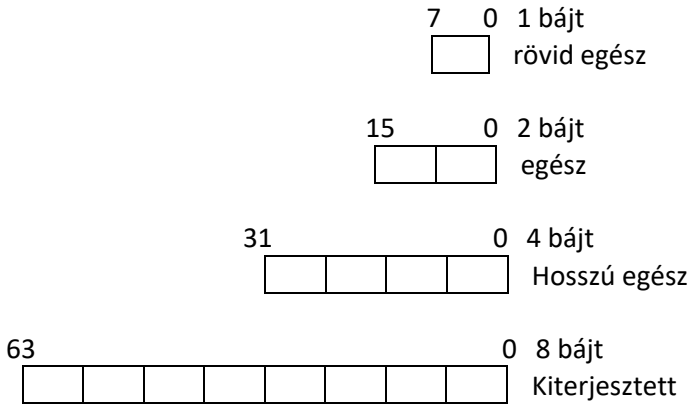
Típusa:

- Egész: ide tartoznak az egész számok, a karakterkódok, a logikai értékek, de például a képpixel színe is. E számokkal elsősorban a CPU aritmetikai és logikai egysége végez műveletet.
- Tört (lebegőpontos) számok, amelyek feldolgozása az FPP (floating point processor) feladata.

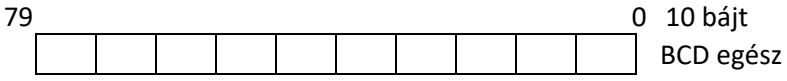
Fogalma: adattárolásra szolgáló hely az operatív memóriában, tartalma módosítható.

Működése: tartalma nem törölhető, csak felülírható, az adat kiolvasásakor tartalma nem változik. Deklaráláskor (ha nincs automatikus kezdőértékadás) értéke meghatározhatatlan.

Egész típusú számok:



Pakolt egész számok:



Előjeles számok esetén mindig a legnagyobb helyiértékű bit jelenti az előjelet:

- pozitív, ha a bit = 0,
- negatív, ha a bit = 1.

Tudjuk, hogy a szám és ellentettjének (additív inverzének) összege nulla: x + (-x) = 0.

Példa:

43_{10} =	0	0	1	0	1	0	1	1
	1	1	0	1	0	1	0	0

1-es komplemens

1	1	0	1	0	1	0	1

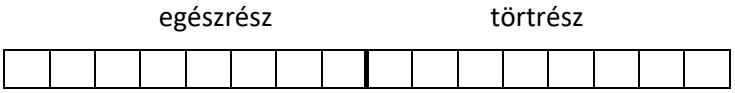
2-es komplemens

Adjuk össze az eredeti számot és a 2-es komplemensét:

43_{10} =	0	0	1	0	1	0	1	1
-43_{10} =	1	1	0	1	0	1	0	1
	0	0	0	0	0	0	0	0

Tört (fixpontos) számok:

Alakítsuk át a $123{,}24_{10}$-et 2-es számrendszerre:



123	maradék	0,24	egészrész
61	1	0,48	0
30	1	0,96	0
15	0	0,92	1
7	1	0,84	1
3	1	0,68	1
1	1	0,36	1
0	1	0,72	0
		0,44	1

0	1	1	1	1	0	1	1	0	0	1	1	1	1	0	1

= 123, 23828125

Tört (lebegőpontos) számok:

A fixpontos ábrázolás hátrányait küszöböli ki az úgynevezett lebegőpontos számábrázolás, amely bizonyos szempontból rugalmasabban, de bonyolultabban kezeli a számokat. A lebegőpontos ábrázolás alapja az, hogy a számok hatványkitevős alakban is felírhatók:

$$\pm m \cdot p^k$$

ahol: p: a számrendszer alapszáma (esetünkben 2),

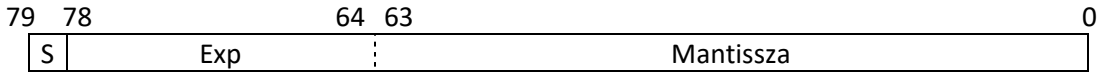
m: mantissza,

k: karakterisztika.

A hatványkitevős forma egyértelműsége érdekében elfogadtak egy közös elvet: az m mindig kisebb, mint 1, és a tizedesponttól (kettedespont) jobbra eső első számjegy nem lehet nulla.

$$\frac{1}{p} \le m < 1$$

(Például: tízes számrendszerben $0{,}1 \le m < 1$, kettes számrendszerben $0{,}5 \le m < 1$.)



Egyszeres pontosság 4 bájt - Short Real Number

Dupla pontosság 8 bájt - Long Real Number

Kiterjesztett pontosság 10 bájt - Real Number

A számítógépes számábrázolásnak korlátjai vannak, ugyanis a változók (amelyekben a számokat tároljuk) csak meghatározott típusúak és méretűek lehetnek.

Példák:

1. Az egész típusú számok ábrázolása, előjelkezelés, számok ellentettje.

   Példa:

   Vizsgáljuk meg, hogy mi történik, ha egy egész (byte-os) változónak az értelmezési tartományánál nagyobb (vagy kisebb) értéket adunk meg. Ismert, hogy egy előjel nélküli byte értéke maximum 255 lehet. Írjunk programot, amelyik 250-től egyesével növeli a byte értékét.

   Megbeszélés:

   1. Byte típusú változó értelmezési tartománya.
   2. Előjeles és előjel nélküli változók.
   3. Túlcsordulás, alulcsordulás.

   Megoldás:

```
#include <stdio.h>
int main(){
        char i; unsigned char j=250;
        for(i=0; i<10; j++, i++) {
                printf("%d\n",j);
}       }
```

   Példa:

   Módosítsuk a feladatot úgy, hogy 5 kezdőértéktől egyesével csökkentjük a számot. Vizsgáljuk meg, mi történik előjeles szám esetén.

2. Lebegőpontos számok ábrázolása.

   Példa:

   Ismert, hogy egy lebegőpontos szám nagyságrendileg $10^{\pm 38}$ körül van. Írjunk programot, amelyik 10^{35}-től többször tízszeresére növeli a szám értékét.

   Megbeszélés:

   1. Egész és törtrész átalakítása külön-külön.
   2. Véges tizedes tört nem biztos, hogy átalakítható kettedes törtté a rendelkezésre álló bitszámon. Számábrázolási pontosság.
   3. Lebegőpontos alak, fix és lebegőpontos ábrázolás összehasonlítása.
   4. Lebegőpontos alul- és túlcsordulás.

   Megoldás:

```
#include <stdio.h>
int main(){
        char i;
        float j=1E35;
        for(i=0; i<10; j*=10, i++) {
                printf("%f\n",j);
}       }
```

   Megbeszélés:

   1. Vizsgáljuk meg az eredményt! Látható a számábrázolási pontosság és a hibaüzenet túlcsordulás esetén.
   2. Módosítsuk a feladatot úgy, hogy 10^{-35} kezdőértéket adunk, és tízzel többször elosztjuk a számot.

Feladatok:

1. Kérjünk be karaktereket egyesével a scanf() függvénnyel a SPACE (szóköz) végjelig, majd írjuk ki a karaktert az ASCII kódjával.
2. Kérjünk be karaktereket a getchar() függvénnyel az EOF végjelig, számláljuk meg, hogy hány karaktert adtunk meg.
3. Kérjünk be szám karaktereket és írjuk ki azokat a nevükkel (1 – egy stb.). Használjuk a getchar() függvényt. Nem szám karakter megadásakor adjunk hibajelzést.
4. Kérjünk be szám karaktereket, írjuk ki, hogy páros vagy páratlan számot adtunk meg, valamint számláljuk meg, hogy hány páros és hány páratlan számot adtunk meg. A ciklust az EOF-ig tartson. Használjuk a getchar() függvényt. Nem szám karakter megadásakor adjunk hibajelzést.
5. Kérjünk be szám karaktereket és alakítsuk át azokat egész számmá. A ciklust az EOF-ig tartson. Nem szám karakter megadásakor adjunk hibajelzést.
6. Készítsen programot, amely bekér egy karaktert, majd kiírja az ASCII kódját, decimálisan, oktálisan és hexadecimálisan.
7. Kérjünk be szám karaktereket ENTER-ig, majd alakítsuk át egész számmá.
8. Készítsen programot, amely bekér egy számot majd átalakítja bináris számmá és kiírja azt.
9. Készítsen programot a másodfokú egyenlet megoldásához. Az együtthatókat billentyűzetről kérje be.
10. Kérjünk be egy egész számot, majd írjuk ki a számjegyeket fordított sorrendbe.
11. Határozza meg két szám legnagyobb közös osztóját. Használja a korábban ismertetett euklideszi algoritmust.
12. Vizsgáljuk meg egy bekért számról, hogy az prímszám-e.