2.3. A C nyelv utasításai

A C szabvány hét csoportban osztályozza a C nyelv utasításait:

Csoport	Kulcsszavak, ill. jelölések
Kifejezés utasítás	
Üres utasítás:	**;**
Összetett utasítás:	**{}**
Szelekciós utasítások:	**if** **else** **switch**
Cimkézett utasítások:	**case** **default**
Vezérlésátadó utasítások:	**break** **continue** **return** **(goto)**
Iterációs (ciklus) utasítások:	**do** **for** **while**

Az utasítások részletes tárgyalásánál nem a fenti csoportosítást követjük, hanem az egyes algoritmus típusokhoz illesztjük a megfelelő utasításokat.

2.3.1. Utasítások és blokkok

Tetszőleges - a C szabványnak megfelelő kifejezés utasítás lesz, ha pontosvesszőt (;) helyezünk mögé. Nézzünk néhány kifejezés utasítást:

```
kerulet = a + b + c;     //értékadás
i++;                     //i növelése 1-gyel
a = b = c = 3;           //többszörös értékadás
nagyobb (a,b);           //void tip. függvény hívása
z = sin (alfa) + 1.25    //függvényt hívó kifejezés
```

A kifejezés utasítás végrehajtása a kifejezésnek a megfelelő szabályok szerinti kiértékelését jelenti. Mielőtt a következő utasításra adódik a vezérlés, a teljes kiértékelés (a mellékhatásokkal együtt) végbemegy.

Az üres utasítás egyetlen pontosvesszőből áll, használatára akkor van szükség, amikor nincs szükség semmilyen tevékenység végrehajtására, de a szintaktikai szabályok szerint a program adott pontján utasításnak kell szerepelnie. Gyakran használjuk szelekciós és ciklus utasításokban.

A kapcsos zárójeleket használjuk arra, hogy a logikailag összetartozó deklarációkat és utasításokat egyetlen összetett utasításba vagy blokkba csoportosítsuk. Ahol egyetlen utasítás megadását engedélyezi a C szabvány, ott utasítás blokk is elhelyezhető.

Az összetett utasítás (blokk) általános formája:

```
{
        lokális definíciók és deklarációk
        utasítások
}
```

A következő esetekben használjuk:

- Amikor több logikailag összefüggő utasítást egyetlen utasításként kell kezelni.
- Függvények törzseként
- Definíciók és deklarációk érvényességének lokalizálására.

2.3.2. Szelekciók

2.3.2.1. feladat:
Készítsen programot, amely bekér egy egész számot, s ha ez páratlan szám, akkor kiírja: Páratlan !

Megoldás:

Pontosítsuk a feladatot:
1. Legfeljebb mekkora lehet a szám ? Válasz: max. 3 jegyű, pozitív vagy negatív.
2 Ha páros számot kaptunk, akkor ne csináljunk semmit ? Válasz: páros szám esetén nincs semmi teendő !

Pszeudokód:

```
Program
    Változók: szam    (-999 ... +999)
     Be: szam
     Ha (szam mod 2 != 0) akkor
          Ki: "Páratlan !"
Program vége
```

A pszeudokódban a továbbiakban nem feltétlenül kell használjuk az **akkor** kulcsszót, mivel a C-ben nincs ennek megfelelő utasítás.

Kódoljuk az algoritmust :

2.3.2.1. mintapélda, **egyágú szelekció**, az if utasítás

```
#include <stdio.h>
#include <conio.h>
void main (void)
{
  // Egyágú szelekció: a bekért szám páros-e
  int szam;
  printf ("\nKérek egy egész számot -999 és +999 között:");
  scanf ("%d", &szam);
  if (szam % 2 != 0)                //if (feltétel)
     printf ("\nPáratlan!");        //   utasítás
  getch ();
}   //program vége
```

A fenti kódban egy új utasítást használtunk:

az if utasítás:

if (kifejezés)
utasítás-blokk

Az if utasítás a C nyelv legegyszerűbb vezérlési szerkezete. Segítségével egy utasítás, vagy utasítás-blokk végrehajtását egy kifejezés (feltétel) értékétől tehetjük függővé. Az utasítás(-blokk) csak akkor hajtódik végre, ha a kifejezés értéke nem nulla (igaz).
A fenti példában az

```
if (szam % 2 != 0)
     printf ("\nPáratlan!");
```

programsorok végrehajtásakor a rendszer először kiértékeli az **if** utasítás után zárójelben megadott kifejezés értékét. A zárójelen belül egy logikai relációs kifejezés áll. A kifejezés bal oldalán álló aritmetikai művelet eredményét kell összehasonlítani a jobboldalon álló 0 (nulla) értékkel. Ha a "nem azonosság" fennáll (vagyis a szám 2-vel való osztása utáni maradék nem nulla), akkor a relációs kifejezés értéke 1, és a feltételtől függő printf utasítás végrehajtásra kerül, vagyis kiíródik a képernyőre új sorban a

```
Páratlan!
```

szöveg.

2.3.2.2. feladat:
Készítsen programot, amely bekér egy (-999 és +999 közötti) egész számot, s ha ez páratlan szám, akkor kiírja: "Páratlan !", ha pedig páros a szám, akkor kiírja: "Páros!"

Megoldás:

Pszeudokód:

Program

Változók: eszam (-999 ... +999)
Be: eszam
Ha (eszam mod 2 != 0) **akkor**
Ki: "Páratlan !"
Egyébként
Ki: "Páros!"
Program vége

Kódoljuk az algoritmust :

2.3.2.2. mintapélda, **kétágú szelekció**, az if utasítás

```
#include <stdio.h>
#include <conio.h>
void main (void)
{
  // Kétágú szelekció: A bekért szám páros v. páratlan
  int szam;
  printf ("\nKérek egy egész számot -999 és +999 között:");
  scanf ("%d", &szam);
  if (szam % 2 != 0)
     printf ("\nPáratlan!");
  else
     printf ("\nPáros!");
  getch ();
}   // program vége
```

A fenti kódban az if utasítás teljes formáját használtuk:

```
if (kifejezés)
   utasítás-blokk 1
else
   utasítás-blokk 2
```

Ha a kifejezés értéke nem nulla, akkor az utasítás-blokk-1 utasításai hajtódnak végre, egyébként pedig az utasítás-blokk 2 utasításai. (Ne felejtsük el, hogy az utasítás-blokkot kapcsos zárójelek közé ({}) kell helyezni !)

Vegyük észre, hogy mivel az if utasítás feltétele egy kifejezés nem nulla, vagy nulla voltának tesztelése, a kód kézenfekvő módon egyszerűsíthető, ha az

```
     if (kifejezés != 0)
helyett az
     if (kifejezés)
```

alakot használjuk. A feltétel kifejezést körülvevő zárójelet mindig ki kell tenni !

A 2.3.2.2. mintapéldában szereplő if utasítás tehát az alábbi egyszerűbb formában is felírható és ugyanazt az eredményt szolgáltatja:

```
if (szam % 2)
    printf ("\nPáratlan!");
else
    printf ("\nPáros!");
```

Figyeljük meg, hogy itt az **if-else** szerkezetben szereplő utasítás-blokkok csak egyetlen utasítást tartalmaznak, melyeket pontosvessző zár le. Az **if** utáni feltétel záró zárójele után viszont nem szabad kitenni a pontosvesszőt, az a fordító számára egy üres utasítást jelentene.

2.3.2.3. feladat:
Készítsen programot, amely bekéri egy autó által megtett utat (kilométerben) és az út megtételéhez szükséges időt (másodpercben), majd kiírja az átlagsebességet km/h-ban.

Megoldás:

Pontosítsuk a feladatot:

A sebesség = megtett_út / eltelt_idő : nullával való osztást kerülni kell)
A másodpercben megadott időt át kell váltani órára.

Pszeudokód:

Program
Változók: s (megtett út), pozitív valós szám
t (eltelt idő), pozitív valós szám
v (sebesség), pozitív valós szám
Be: s, t [s km-ben, t mp-ben]
Ha (t <= 0) vagy (s <= 0) **akkor**
Ki: "Hibás adat !"
Egyébként
t = t /3600
v = s/t
Ki: "Az átlagsebesség:", v
Program vége

Kódoljuk az algoritmust :

2.3.2.3. mintapélda, **átlagsebesség számítás**

```
#include <stdio.h>
#include <conio.h>
void main (void)
{
  // Kétágú szelekció: átlagsebesség számítás (adott: út, idő)
  float s,t,v;    //út, idő, sebesség
  printf("\nKérem a megtett utat (km-ben):");
  scanf ("%f",&s);  fflush (stdin);
  printf("\nKérem az eltelt időt (mp-ben):"); scanf ("%f",&t);
  if ((s <= 0) || (t <= 0))
     printf ("\nHibás adat !");
  else
     {                 //elseághoz tartozó utasításblokk kezdete
```

```
      t = t / 3600.;          //az idő átváltása mp-ről órára
      v = s / t;
      printf ("\nAz átlagsebesség: %8.2f km/óra", v);
     }               //elseághoz tartozó utasításblokk vége
  getch ();
}   //program vége
```

Áttekinthetőség: tegyünk megjegyzést mindegyik záró kapcsos zárójel, } mögé !

Tesztelés: futtassuk a programot az alábbi adatokkal:

S	-5	500	100	2
T	1	0	3600	60
várt eredmény	*Hibás adat*	*Hibás adat*	*100 km/óra*	*120 km/óra*

2.3.2.4. feladat

Készítsen programot, amely bekéri egy felnőtt férfi testmagasság (cm-ben) és testsúly (kg-ban) adatait.
Ha a magasság 100 cm fölötti, akkor megvizsgálja, hogy túlsúlyos-e az illető:
ha a kg-ban mért súlya nagyobb, mint a cm-ben mért magasság 100 fölötti része,
akkor kiírja: "Túlsúlyos, fogynia kell!".
Ha a magasság 100 alatti érték, akkor írja ki a program, hogy "Gyerekekkel nem foglalkozom !"

Megoldás:

Pontosítsuk a feladatot:

Ha 100 cm fölött van a magasság és nem túlsúlyos, akkor mit tegyünk ?

Pszeudokód:

Program
Változók: magassag (cm-ben), pozitív egész
suly (kg-ban), pozitív egész
Be: magasság, suly
Ha (magassag > 100) **akkor** *//1. Ha*
Ha (suly > magassag -100) **akkor** *//2. Ha*
Ki: "Ön túlsúlyos, fogynia kell !"
Egyébként
Ki: "Gyerekekkel nem foglalkozom !"
Program vége

Kódoljuk az algoritmust :

A kódolásnál ügyeljünk arra, hogy melyik **Ha**ághoz tartozik az **Egyébként** ág ! A pszeudokódban ezt a tagolással hangsúlyoztuk: az **Egyébként** ág az első **Ha** ághoz tartozik !

2.3.2.4.. mintapélda, **egymásba ágyazott ciklusok**

```
#include <stdio.h>
#include <conio.h>
void main (void)
{
  //Testmagasság és testsúly alapján túlsúlyosság vizsgálata
  int magassag, suly;
  printf("\nKérem a testmagasságot cm-ben:");
  scanf("%d",&magassag);
  fflush (stdin);
  printf("\nKérem a testsúlyt      kg-ban:"); scanf("%d",&suly);
  if (magassag > 100)                 //1. if
     {
       if (suly > magassag - 100)     //2. if
          printf ("\nÖn túlsúlyos, fogynia kell !");
     }                    //1. if-hez tartozó utasításblokk vége
  else                                //első if-hez tartozó else
     printf ("\nGyerekekkel nem foglalkozom !");
  printf ("\n\nKöszönöm az együttműködést !");
  getch ();
}    // program vége
```

Áttekinthetőség:

- *A beszédes változónevekhez (magasság, suly) nem kell külön magyarázat, megjegyzés.*
- *Ha több if utasítás is szerepel a programben, célszerű ezeket a megjegyzésben beszámozni, s az else ághoz beírni, melyik if-hez tartozik*
- *A suly > magassag - 100 relációs kifejezésben nem szükséges zárójelezni a magassag - 100 kifejezést*
- *Ha van olyan bemeneti adat kombináció, amely esetén a programnak nincs semmilyen kimenete, a program végére tegyünk egy "elköszönő" kiírást, így legalább tudjuk, hogy lefutott a program.*

Az If utasítások egymásba is ágyazhatók. Próbáljuk ki, jól működik-e a fenti program, ha elhagyjuk az első if-et követő {} zárójelezést:

```
if (magassag > 100)                 //1. if

  if (suly > magassag - 100)       //2. if
     printf ("\nÖn túlsúlyos, fogynia kell !");

else
    printf ("\nGyerekekkel nem foglalkozom !");
```

A fordító az **else** ágat minden esetben a hozzá legközelebb eső **if** utasításhoz rendeli !

Egymásba ágyazott if utasítások esetén kétféle módon érhetjük el, hogy a fordító is úgy értelmezze az utasításokat, ahogyan mi elgondoltuk:

1. Minden **if** utasításhoz csatolunk **else** ágat, akkor is, ha az **else** ágon nincs semmi teendő, a fenti példa tehát az alábbi formában működik jól:

```
if (magassag > 100)                    //1. if
     if (suly > magassag - 100)        //2. if
          printf ("\nÖn túlsúlyos, fogynia kell !");
     else ;                  //az else utáni ; egy üres utasítást jelent !
else
     printf ("\nGyerekekkel nem foglalkozom !");
```

2. A "belső" **if** utasítást {} zárójelek közé tesszük, így az egy külön utasításblokkot alkot. A 2.3.2.4. mintapéldában ezt a megoldást választottuk.

2.3.2.5. feladat
Készítsen programot, amely bekér egy életkor adatot, majd kiírja,
hogy az illető gyerek (0-14. év), kamasz (15-23. év), felnőtt (24-62.év), vagy idősebb (63-)...

Megoldás:

Pontosítsuk a feladatot: Negatív életkort is megadhat a felhasználó ?

Pszeudokód:

Program
Változók: életkor, pozitív egész
Be: életkor
Ha (életkor < 0) **akkor**
Ki: "Hibás adat !"
Egyébként ha (életkor < 15) **akkor**
Ki: "Gyerek vagy még !"
Egyébként ha (életkor < 24) **akkor**
Ki: "Kamaszkorban a legnehezebb !"
Egyébként ha (életkor < 63) **akkor**
Ki: "A felnőtteknek sem könnyű ..."
Egyébként
Ki: "Idősebbek hamarabb abbahagyhatják, de nem a tanulást !"
Program vége

Kódoljuk az algoritmust :

Hogyan kódoljuk az **Egyébként ha** szerkezetet ?

Az egymásba ágyazott **if** utasítások gyakran használt formája, amikor az **else** ágakban szerepel újabb **if** utasítás:

```
if (kifejezés1)
    utasítás-blokk 1
```

```
else if (kifejezés2)
    utasítás-blokk 2
else if (kifejezés3)
    utasítás-blokk 3
else
    utasítás-blokk 4
```

Ezzel a szerkezettel a program többirányú elágaztatását valósítjuk meg. Ha bármelyik kifejezés igaz, akkor a hozzákapcsolt utasítás-blokk utasításai kerülnek végrehajtásra. Ha egyik feltétel sem teljesül, akkor a program az utolsó **else** utasítást követő utasítás-blokk utasításait hajtja végre. Az **else if** szerkezetből természetesen akárhány lehet.

2.3.2.5.. mintapélda, **az else-if szerkezet**

```
#include <stdio.h>
#include <conio.h>
void main (void)
{
  //Életkor alapján besorolás (gyerek, kamasz, felnőtt, idősebb)
  int eletkor;
  printf ("\nKérem az életkorát (év):"); scanf ("%d",&eletkor);
  if (eletkor < 0)
     printf ("\nHibás adat !");
  else if (eletkor < 15)              // eletkor:  0...14
     printf ("\nGyerek vagy még !");
  else if (eletkor < 24)              // eletkor: 15...23
     printf ("\nKamaszkorban a legnehezebb!");
  else if (eletkor < 63)              // eletkor: 24...62
     printf ("\nA felnőtteknek sem könnyű ...");
  else                                //eletkor: 62 fölött
     printf ("\nIdősebbek hamarabb abbahagyhatják... !");
  getch ();
}    // program vége
```

2.3.2.6. feladat

Készítsen programot, amely megvalósítja a legegyszerűbb kalkulátor funkciókat (összeadás, kivonás szorzás, osztás)

Megoldás:

Pontosítsuk a feladatot:

- Csak a négy alapműveletre kell működnie a programnak, más karakter esetén hibajelzést ad.
- Mi történjen, ha nullával való osztás a kijelölt művelet ? Semmi esetre sem szabad megengedni, hogy a program végrehajtson egy nullával való osztás műveletet, ezt ki kell védeni. Ilyenkor írja ki a program, hogy a művelet nem értelmezett.

Pszeudokód:

Program

Változók: x, y, eredmeny (a két operandus, és az eredmény,
tetszőleges valós számok)
op: karakter típusú változó a műveleti jel tárolására
siker: egész típusú segédváltozó, kezdőértéke = 1
értéke 1, ha a műveletet el tudtuk végezni, 0, ha nem
és -1 az értéke, ha a műveleti jel nem értelmezhető.

Be: x, op, y [szóközök nélkül !]

Ha (op == '+') **akkor** // összeadás ?
eredmeny = x + y
Egyébként ha (op == '-') **akkor** // kivonás ?
eredmeny = x - y
Egyébként ha (op == '*') **akkor** // szorzás ?
eredmeny = x * y
Egyébként ha (op == '/') **akkor** // osztás ?
Ha (y <>0) **akkor**
eredmeny = x / y
Egyébként
siker = 0 //az osztás nem végezhető el
Egyébként
siker = -1 //hibás műveleti jel érkezett
Ha (siker == 1) **akkor**
Ki: x, op, y, eredmeny
Egyébként ha (siker == 0) **akkor**
Ki: "A művelet nem végezhető el"
Egyébként
Ki: "Hibás műveleti jel "

Program vége

Kódoljuk az algoritmust a már megismert **else if** szerkezetekkel, a kód megírását az olvasóra bízzuk.

Az **else if** szerkezet egy speciális esete, amikor a használt kifejezések egyenlőség vizsgálatokat (==) tartalmaznak. Az ilyen esetekben, amikor konstans értékek egyenlősége alapján ágaztatjuk el a programot, a **switch** utasítás sokkal áttekinthetőbb megoldási lehetőséget biztosít.

A **switch** utasítás többirányú programelágaztatást tesz lehetővé olyan esetekben, amikor egy egész kifejezés értékét több konstans értékkel kell összehasonlítanunk. Az utasítás általános alakja:

```
switch (kifejezés)
{
  case konstans kifejezés: utasítások;  <break;>
  ........
  case konstans kifejezés: utasítások;  <break;>
```

```
    default: utasítások;
}
```

A **switch** utasítás először kiértékeli a kifejezést, majd a vezérlést átadja arra a **case** cimkére (esetre), amelyben a konstans kifejezés értéke megegyezik a kiértékelt kifejezés értékével. A program futása ettől a ponttól folytatódik. Amennyiben egyik **case** utáni konstans sem egyezik meg a kifejezés értékével, akkor a program futása a **default** cimkével megjelölt utasítástól folytatódik. Ha nem használunk **default** cimkét, akkor a vezérlés a **switch** utasítás blokkját záró } jel utáni utasításra adódik. A **default** cimke bárhol elhelyezkedhet a **switch** blokkjában, tehát nem feltétlenül az utolsó cimke.

Amikor a **case** vagy a **default** cimkénél belépünk a **switch** utasítás törzsébe, akkor attól a ponttól kezdve a megadott utasítások sorban végrehajtódnak a blokkot záró zárójel eléréséig, függetlenül attól, hogy közben hány **case** cimkét találunk. Általában azonban csak a belépési cimkéhez tartozó utasításokat kell a **switch** törzséből végrehajtani, ezért az adott esethez tartozó utasítások végrehajtása után a **break** utasítással kilépünk a **switch** utasítás utáni utasításra.

A **switch** utasításban megadható esetek száma implementációfüggő, - az ANSI C szabvány legalább 256 esetet javasol. Minden egyes esethez tartozó konstans kifejezésnek egyedi (nem ismétlődő) esettel kell rendelkeznie.

Mielőtt az előbbi példánkat átírnánk a **switch** utasítást használva az **else if** szerkezetek helyett, írjuk át a pszeudokódot is egy ennek megfelelő formára:

A 2.3.2.6. feladat megoldásának pszeudokódja

Program
Változók: x, y, eredmeny (a két operandus, és az eredmény,
tetszőleges valós számok)
op: karakter típusú változó a műveleti jel tárolására
siker: egész típusú segédváltozó, kezdőértéke = 1
értéke 1, ha a műveletet el tudtuk végezni, 0, ha nem
és -1 az értéke, ha a műveleti jel nem értelmezhető.
Be: x, op, y [szóközök nélkül !]

Elágazás (op) **szerint**
'+' **esetén :** eredmeny = x + y **kiugrás** //összeadás
'-' **esetén :** eredmeny = x - y **kiugrás** //kivonás
'*' **esetén :** eredmeny = x * y **kiugrás** //szorzás
'/' **esetén : Ha** (y <>0) **akkor** //osztás
eredmeny = x / y
Egyébként
siker = 0 //az osztás nem végezhető el
Más esetben:
siker = -1 //hibás műveleti jel érkezett
Elágazás vége

Elágazás (siker) **szerint**
1 **esetén : Ki :**x, op, y, eredmeny **Kiugrás**
0 **esetén : Ki:** "A művelet nem végezhető el" **Kiugrás**
-1 esetén : Ki: "Hibás műveleti jel "
Elágazás vége
Program vége

Már a pszeudokód is áttekinthetőbb, ha mindjárt a többágú elágaztatás kapcsolóval megoldott változatára gondolunk, nézzük meg most a kódolt programot:

2.3.2.6.. mintapélda, **többágú elágaztatás, a switch utasítás**

```
#include <stdio.h>
#include <conio.h>
void main (void)
{
  //Egyszerű kalkulátor a négy alpműveletre
  double x, y, eredmeny = 0;
  char op;              //az operátor karaktere, +,-,*,/
  int siker = 1;
  printf ("\nEz egy egyszerű kalkulátor !");
  printf ("\n\nKérem az elvégzendő műveletet szóközök nélkül:");
  scanf ("%lf%c%lf", &x, &op, &y);
  switch (op)
  {
    case '+': eredmeny = x + y; break;
    case '-': eredmeny = x - y; break;
    case '*': eredmeny = x * y; break;
    case '/':
       if (y)
           eredmeny = x / y;
       else
           siker = 0;
       break;
    default : siker = -1;
  }   //switch op vége

  switch (siker)
  {
    case 1 : printf("%.2lf %c %.2lf = %.2lf", x,op,y,eredmeny);
             break;
    case 0 : printf ("\nA művelet nem végezhető el."); break;
    case -1: printf ("Hibás műveleti jel !");
  }   //switch siker vége
  getch ();
}    // program vége
```

Gyakori hiba: a **break** véletlen elhagyása, ekkor a **switch** utasítás törzsének valamennyi utasítása, amely a bekapcsolódási pont után van, végrehajtódik. A következő példában a switch utasításnak éppen ezt a tulajdonságát használjuk ki.

2.3.2.7. feladat
Készítsen programot, amely egy hónap, nap adatokkal megadott dátumról megállapítja, hogy az az évnek hányadik napja. A programmal azt is közölni kell, hogy az adott év szökőév-e.

Megoldás:

Pontosítsuk a feladatot:
- A hónapot a nevével is megadhatom ? Válasz: nem, a hónap sorszámát kell megadni.
- Előfordulhat hibás adatbevitel ? Válasz: igen, és erre fel kell készülni, a felhasználó véletlenül is megadhat pl. egy 9. 31-es dátumot, vagy ki akarja próbálni, hogy mit tesz a program, ha 05.40-et adunk meg dátumként. Ezért mielőtt elkezdjük a számolást, ellenőrizzük, hogy valós adatokat kaptunk-e.

Elemezzük a feladatot:
Vegyünk egy konkrét példát: 04.12, vagyis április 12-e, nem szökőév.

Eltelt: 31 nap januárban + 28 nap februárban + 31 nap márciusban + 12 nap áprilisban

Számolhatunk "fordított" sorrendben is:
Eltelt: 12 nap áprilisban + 31 nap márciusban + 28 nap februárban + 31 nap januárban.

Vegyük ez utóbbi metódust: tehát adjuk hozzá a naphoz a hónapot megelőző hónapok maximális napszámát. Gondoljuk végig a határeseteket: januári dátum esetén csak a napot kell venni, a decemberi dátumnál pedig arra kell rájönnünk, hogy a december soha nem lesz "megelőző" hónap.

Pszeudokód:

Program
Változók: ho, nap, evnapja: pozitív egészek
szokoev: 1, ha szökőév, 0, ha nem.

Be : szokoev [0,1]
Be : ho [1..12], nap [1..31]

Elágazás (ho) **szerint** //a bevitt adatok ellenőrzése:ho-nap megfelelés
2 **esetén : Ha** (nap > 28 + szokoev) vagy (nap < 0) **akkor**
Ki : "Hibás a nap megadása !";
Kiugrás a programból (-1)
4, 6, 9, 11 **esetén:**
Ha (nap > 30) vagy (nap < 0) **akkor**
Ki : "Hibás a nap megadása ";

Kiugrás a programból (-1)
1,3,5,7,8,10,12 **esetén:**
Ha (nap > 31) vagy (nap < 0) **akkor**
Ki : "Hibás a nap megadása ";
Kiugrás a programból (-1)
Más esetben : Ki : "Hibás a hónap megadása !"
Kiugrás a programból (-1)
Elágazás vége

evnapja = nap //a megadott hónapban eltelt napok száma
Elágazás (ho - 1) **szerint** //ha eddig eljutott a program végrehajtása,
//már biztos, hogy jók a bevitt adatok
11 **esetén :** evnapja = evnapja + 30
10 **esetén :** evnapja = evnapja + 31
9 **esetén :** evnapja = evnapja + 30
8 **esetén :** evnapja = evnapja + 31
7 **esetén :** evnapja = evnapja + 31
6 **esetén :** evnapja = evnapja + 30
5 **esetén :** evnapja = evnapja + 31
4 **esetén :** evnapja = evnapja + 30
3 **esetén :** evnapja = evnapja + 31
2 **esetén :** evnapja = evnapja + 28 + szokoev
1 **esetén :** evnapja = evnapja + 31
Elágazás vége
Ki : evnapja
Program vége

Gondolatban játsszuk végig, hogyan működik a program a fent elemzett 04.12-i dátummal, amely nem szökőévben értendő. Miután a bevitel és az ellenőrzések megtörténtek, a kérdéses napok számát kezdjük el kiszámolni:

evnapja = 12

Elágazás következik ho-1, szerint. ho-1 aktuális értéke 3, tehát a 3-as cimkénél lépünk be az elágazás törzsébe:

3-as cimke: evnapja = 12 + 31 = 43
2-es cimke: evnapja = 43 + 28 = 71
1-es cimke: evnapja = 71 + 31 = 102 és ez a végeredmény.

Mivel nincs előírva kiugrás egyik cimkénél sem, ha egyszer beléptünk az elágazás törzs részébe, onnét kezdve végigfut a program.

2.3.2.7. mintapélda: a **switch utasítás**

```
#include <stdio.h>
#include <conio.h>
int main (void)
{
  //hó,nap dátumról megállapítja, az év hányadik napja
```

```
int ho, nap, evnapja;
int szokoev = 0;        //alapértelmezés: nem szökőév
char szoko;

printf("\nNyomja le az i billentyűt, ha szökőévben vagyunk:");
scanf ("%c",&szoko);
fflush (stdin);
printf ("\nAdja meg a hónapot (1-12):"); scanf ("%d",&ho);
fflush (stdin);
printf ("\nAdja meg a napot         :"); scanf ("%d",&nap);

szokoev = (szoko == 'i') || (szoko == 'I');
 //Ha nem i vagy I válasz érkezett, akkor nem szökőévként kezeljük !

switch (ho)        // a bevitt ho, nap adatok ellenőrzése
{
  case 2:  if ((nap < 0) || (nap > 28 + szokoev))
           {                              //február:max28(29)nap
             printf ("\nHibás a nap megadása");
             return (-1);                 //Kilépés a programból
           }   //if
           break;
  case 4:                  // 30 napos hónapok ellenőrzése
  case 6:
  case 9:
  case 11: if ((nap < 0) || (nap > 30))
           {
             printf ("\nHibás a nap megadása");
             return (-1);
           }
           break;
  case 1:                  // 31 napos hónapok ellenőrzése
  case 3:
  case 5:
  case 7:
  case 8:
  case 10: if ((nap < 0) || (nap > 31))
           {
             printf ("\nHibás a nap megadása");
             return (-1);
           }
           break;
  default : printf ("\nHibás hónap !");
            return (-1);
}  // switch ho

evnapja = nap;        //az adott hónapban eltelt napok
switch (ho - 1)       //az előző hónapokban eltelt napok
{                                        // hozzáadása
   case 11 : evnapja += 30;
```

```
     case 10 : evnapja += 31;
     case  9 : evnapja += 30;
     case  8 : evnapja += 31;
     case  7 : evnapja += 31;
     case  6 : evnapja += 30;
     case  5 : evnapja += 31;
     case  4 : evnapja += 30;
     case  3 : evnapja += 31;
     case  2 : evnapja += 28 + szokoev;
     case  1 : evnapja += 31;
  } //switch ho-1
  printf("\n%d. hó %2d. napja az év %3d. napja",ho,nap,evnapja);
  getch ();
}    // program vége
```

Összefoglalás

Szelekciók összefoglaló táblázata

Típus	Pszeudokód	C++ utasítások
Egyágú	**Ha (***Feltétel)* **akkor** Utasítás(ok) **Elágazás vége**	**if** (kifejezés) utasítás-blokk
Kétágú	**Ha (***Feltétel)* **akkor** *Utasítás(ok)1* **egyébként** *Utasítás(ok)2*	**if** (kifejezés) utasítás-blokk 1 **else** utasítás-blokk 2
Többágú	**Ha** (*Feltétel1)* **akkor** *Utasítás(ok)1* **Egyébként ha** (Feltétel2) **akkor** *Utasítás(ok)2* ..... **Egyébként** U*tasítás(ok)n+1*	**if** (kifejezés1) utasítás-blokk 1 **else if** (kifejezés2) utasítás-blokk 2 ..... **else** utasítás-blokk n+1

Többágú, kapcsoló típusú	**Elágazás** (kifejezés) **szerint**
	
	kf1 **esetén**: *utasítások*, <**kiugrás**>
	kf2 **esetén**: *utasítások*, <**kiugrás**>
	.....
	kfn **esetén**: *utasítások*, <**kiugrás**>
	Más esetben: *utasítások*
	
	Elágazás vége
		switch (kifejezés)
		{
		case kf1: utasítások;<break;>
		case kf2: utasítások;<break;>
		.....
		case kfn:utasítások;<break;>
		default:utasítások;
		}

		// kf jelentése: // konstans kifejezés

Mielőtt továbbmennénk, foglaljuk össze, milyen formai szabályokat tartottunk be a programok forrásszöveginek beírásánál, amelyek ugyan nem kötelezőek a program szintaktikája, a fordító szempontjából, de áttekinthetővé teszik programjainkat a magunk és mások számára is.

Betartásuk minden programozó számára ajánlott.

- A program első sorába megjegyzésként be kell írni a nevet, csoport-számot, EHA kódot.
- A program második sorában tömören meg kell adni a program által megoldott feladatot.
- A forrásprogram kiterjesztése mindig cpp legyen
- Használjon beszédes változóneveket.
- A változók deklarációja legyen rögtön a program elején, s csak ezután következhetnek a végrehajtható utasítások.
- A beviteli utasítást (scanf) mindig előzze meg egy printf utasítás, amellyel a beviendő adat tartalmára, esetleges korlátaira hívjuk fel a felhasználó figyelmét. Az összetartozó printf, scanf utasításokat egy sorban helyezzük el (Ha elférnek.)
- A program utolsó sorában helyezzen el egy getch (); függvényt, így a kiviteli képernyőt addig szemlélhetjük, amíg egy billentyűt le nem nyomtunk.
- A forrásszöveget tagolni kell, valamely utasításhoz tartozó utasításblokk utasításai beljebb kezdődnek.
- A nyitó kapcsos zárójel mindig az őt megelőző utasítás első karakteréhez igazodik, az alatta következő utasítások két karakterrel beljebb kezdődnek.
- A záró kapcsos zárójel a hozzá tartozó nyitó zárójellel azonos oszlopban helyezkedik el, külön sorban van. Mögötte megjegyzésként meg kell adni, hogy milyen utasításhoz tartozik.
- A forrásszöveg begépelésénél minden leírt nyitó zárójelhez azonnal be kell írni a csukó zárójelet is.
- Már néhány programsor bevitele után menteni kell a programot a megfelelő (helyi) tárolóra. A program fordítást nem szabad elindítani addig, míg a forrásszöveget el nem mentettük.
- Hálózati meghajtóról nem indítható a Borland C++.

Ellenőrző kérdések

- Általában miért nem szabad kitenni a pontosvesszőt (;) az **if** utáni feltétel csukózárójele után ? Ha mégis kitesszük, annak mi lesz a hatása ?
- Kötelező-e az **if** utasítás után **else** ágat is használni ?
- Egymásba ágyazott **if** utasításoknál hová tartozik bármely **else** utasítás ?
- Milyen speciális esetét ismeri az egymásba ágyazott ciklusoknak ?
- Kötelező-e a **switch** utasítás törzsében használni a **default** cimkét ?
- Hol helyezkedhet el a **default** cimke a **switch** utasítás törzsén belül ?
- Mi a hatása a **return** () utasításnak ?
- Mi a hatása s **witch** utasítás törzsében elhelyezett **break** utasításnak ?

Gyakorló feladatok

232.1 feladat
Írjuk ki két tetszőleges egész szám hányadosát és a maradékos osztás eredményét. Az osztó először az első, majd a második szám legyen. Feltétlenül legyen védelem a nullával való osztás ellen.

232.2. feladat
Egy beolvasott számnak írjuk ki az előjelét (pozitív, negatív vagy nulla).

232.3. feladat
Ha két beolvasott szám előjele megegyezik, akkor írjuk ki azokat, különben kérjünk másik kettőt.

232.4. feladat
Egy tetszőleges valós számról döntsük el, hogy egész szám-e. Csak ebben az esetben irassuk ki.

232.5. feladat
Vizsgáljuk meg. hogy osztható-e egy megadott egész szám egy másik megadott egész számmal.

232.6. feladat
Három tetszőleges számról döntse el a program, hogy lehetnek-e egy háromszög oldalai ?

232.7. feladat
Kérjen be a program két tetszőleges számot, majd írja ki a közöttük lévő relációt (Pl. 4.5 és 9 esetén írja ki : 4.5 < 9)

232.8. feladat
A program kérjen be egy karaktert, majd döntse el, hogy az azonos-e valamelyik rövid nagánhangzóval (a,e,i,o,ö,u,ü). Írja ki a vizsgálat eredményét pontosan.

232.9. feladat
A program kérje be, hogy a hétnek hányadik napja van, majd írja ki, hogy ez milyen nap (Az első nap a Hétfő.)

232.10. feladat
Egy vállalatnál minden dolgozó a belépéstől számított 5 évenként külön jutalmat kap. A program kérje be a szolgálatban eltöltött időt (év) és írja ki, hogy jár-e a jutalom.

232.11. feladat
Tetszőleges, nulla és egymillió közötti egész számról mondja meg a program, hogy hány jegyű.

232.12. feladat
A program kérjen be három egész számot és írja ki, hogy van-e közöttük két azonos szám.

232.13. feladat

A program döntse el három tetszőleges számról, hogy monoton sorrendben következnek-e. Próbálja meg egyetlen if utasítással megoldani a feladatot.

232.14. feladat
Három tetszőleges, de növekvő sorrendben megadott számról döntse el a program, hogy számtani sorozatot alkotnak-e. Adjon hibajelzést, ha nem növekvő sorrendben kapta a számokat.

232.15. feladat
Adott egy évszám, döntsük el, hogy szökőév-e.

232.16. feladat
Készítsen programot, amely alkalmas bármely másodfokú egyenlet megoldására. Pontosítsa a feladatot.

232.17. feladat
Tetszőleges, legfeljebb 4 jegyű egész számot írjunk fel római számokkal.

232.18. feladat
Készítsen programot, amely egy hónapról megmondja, hogy melyik évszakba esik itt, Európában.

232.19. feladat
Készítsen programot, amely egy 2001-es dátumról (hó, nap) megmondja, hogy milyen napra esik. (2001-ben január 1. hétfő).

232.20. feladat
Készítsen kalkulátor programot, amely az egész számok közötti alapműveletek (+, -, *, /, %) elvégzésére alkalmas. Védje ki a nullával való osztást.

232.21. feladat
Készítsen programot, amely bekér két egész számot és a < vagy a > relációjelet. Megállapítja, hogy a két szám között igaz-e a megadott reláció, majd az eredményt kiírja.
Pl a 3,7, < esetén írja ki: 3 < 7 IGAZ

232.22. feladat
A rend őrei a 6-os út Szekszádhoz közeli részén mérik a mellettük elhaladó autók sebességét. Készítsen programot, amely bekéri megengedett sebesség értékét és az elhaladó jármű sebességét (km-ben), és abban az esetben, ha a jármű túllépte a megengedett sebességhatárt, írja ki annak mért sebességét.

232.23. feladat
Egy kis kft titkárnője részére készítsünk programot, mely segít neki abban, hogy a lehető legkevesebb címlettel ki tudja adni a fizetéseket. A rendelkezésre álló címletek: 10000, 2000, 500, 200, 10, 1.

232.24. feladat
Készítsen programot, amely bekéri egy nyitott fűtési rendszerben keringő víz mennyiségét köbméterben, és az eltelt időt (óra, perc, másodpercben), majd megmondja, hogy mennyi vizet kell utántölteni, hogy ismét tele legyen a tartály. A rendszerből óránként 2 ezrelék párolog el.

232.25. feladat
Készítsen programot, amely bekér két betűt, majd az ÁBC-nek megfelelő (növekvő) sorrendben kiírja azokat.

232.26. feladat
Készítsen programot, amely bekér három tetszőleges számot, majd növekvő sorrendben kiírja azokat. Elemezze a feladatot, hány különböző eset lehetséges !

232.27. feladat
Készítsen programot, amely bekéri egy személy testmagasságát cm-ben, testsúlyát kg-ban, majd kiszámítja a testtömegindexet (BMI), melyet úgy kapunk, hogy a kilogrammban kifejezett testsúlyt elosztjuk a méterben megadott testmagasság négyzetével. Egy 175 cm magas és 62 kg súlyú személy BMI-je például 20,2, ha 77 kg, akkor 25,1 és ha 93 kg, akkor a BMI 30,4. Az orvostudomány a 20 és 25 közötti értéket tekinti normálisnak, 25 és 30 között túlsúlyról, 30 felett elhízásról van szó. Írassa ki a programban a kapott BMI-t és azt, hogy melyik csoportban tartozik ennek alapján az illető.

232.28. feladat
Egy Disco-ba aszerint lehet belépőjegyet kapni, hogy hány éves az illető. A belépőjegy egyben tombola is, fizetni érte csak a kapusnál kell majd, a legfeljebb húszévesek diákjegyet kapnak (200 Ft), az idősebbek felnőtt jegyet (300 Ft). A kapuban a diákjeggyel rendelkezők közül a lányoknak nem kell fizetniük a belépőjegyért, mindenki másnak a jegyen szereplő díjat. Készítsen programot, amely a szükséges adatok bekérése után megállapítja és kiírja, hog y az éppen belépő vendég fizet-e és ha igen, mennyit.

232.29. feladat
Készítsen programot, amely bekér két tetszőleges egész számot. Ha a két szám különbsége nem több 10-nél, akkor írassuk ki a két szám szorzatát, különben a kisebb szám és a nagyobb szám hányadosát 4 tizedesjegy pontossággal. (Mikor nem szabad elvégezni az osztást ?)

232.30. feladat
Szerencsejátékot játszunk ! A gép generál egy 101 és 200 közé eső véletlen számot. Tippeljük meg ezt az értéket. Ha éppen eltaláltuk, akkor 100.000 Ft a jutalom, ha maximum 10 egységnyi az eltérés, akkor 10.000 Ft a nyeremény és maximum 20 egység eltérés esetén is nyertünk 1.000 Ft-ot. Írassuk ki a gép által generált számot, a tippelt számot és a nyeremény összegét.

232.31. feladat
Kalandfilmhez 18 és 22 év közötti Oroszlán jegyben született férfiakat keresnek. Az adatokat adatbázisból töltik be és a személyi szám első 7 számjegye alapján válogatnak. Készítsen programot, amely egy személyi szám-rész alapján megállapítja, hogy az illetőt be kell-e hívni próbafelvételre. A személyi számot itt a billentyűzetről kell megadni.
(Oroszlán jegy: 07.23-08.23.)

2.3.3. Ciklusok

Gyakran találkozunk olyan feladattal, amikor többször ismétlődő tevékenységeket kell végrehajtani. Például amikor a bújócska játékban el kell számolni hangosan 50-ig, akkor a következőket kell tenni:

1. Gondolj az 1-es számra, takard el a szemedet
2. Mondd ki hangosan a gondolt számot
3. Növeld meg eggyel a gondolt számot
4. A gondolt szám nagyobb, mint 50 ? Ha nem, akkor folytasd a 2-es sorszámú feladattól.
5. Miután kimondtad hangosan az 50-et is, indulj megkeresni a társaidat.

A programozási nyelvekben bizonyos utasítások automatikus ismétlését biztosító programszerkezeteket iterációnak vagy ciklusnak nevezzük. Az ismétlés mindaddig tart, amíg az ismétlési, vagy lefutási feltétel igaz. Vannak olyan programszerkezetek is, ahol a ciklus akkor fejeződik be, amikor egy meghatározott kilépési feltétel igazzá válik, vagyis az ismétlés addig tart, amíg a kilépési feltétel hamis. A C nyelvben valamennyi ciklus-utasításban lefutási feltételt kell megfogalmazni.

A fenti kis feladat pszeudokóddal is leírható:

```
          Program
             Változók: gondolt_szam: pozitív egész szám
gondolt_szam = 1
Ciklus
      Ki: gondolt_szam                  // Ciklusmag utasításai
       gondolt_szam = gondolt_szam + 1
amíg   gondolt_szam <= 50
Ciklus vége
Ki: "Aki bújt, aki nem, megyek !"
   Program vége
```

Hátultesztelő ciklus

Pszeudokódja: **(**Lefutási feltétel megfogalmazása esetén:)

```
Ciklus
     Ciklusmag utasításai
amíg Lefutási feltétel
Ciklus vége
```

Működése:

A ciklusmag utasításait mindaddig kell végrehajtani. amíg a *Lefutási feltétel igaz*. Ha a *lefutási feltétel* hamissá válik, a végrehajtás a ciklus vége után folytatódik. A ciklusmag

utasításait egyszer mindenképpen végrehajtja a rendszer, mivel a lefutási feltétel kiértékelése a ciklusmag utasításainak végrehajtása után történik.

A C nyelvi programszerkezet:

```
do
{
   ciklusmag utasításai
}
while (kifejezés);    //kifejezés: lefutási feltétel
```

A **do-while** ciklus futása során mindig a ciklusmag utasításait követi a kifejezés kiértékelése. Amíg a kifejezés értéke nem nulla (igaz), addig új iteráció kezdődik. Amikor a kifejezés értéke a vizsgálat során először bizonyul nulla értékűnek (hamis), a vezérlés a **while** utáni utasításra adódik, a ciklus befejezi a működését.

Kódoljuk az előbbi kis feladatot:

2.3.3.1. mintapélda, **hátultesztelő ciklus: számlálás**

```
#include <stdio.h>
#include <conio.h>
void main (void)
//Hátultesztelő ciklus: számlálás adott értékig
{
  int gondolt_szam = 1;      //Kezdeti értékadás a deklarációban
  printf ("\n Kezdjük a számolást : ");
  do
  {
     printf ("%d, ",gondolt_szam);      // Ciklusmag utasításai
     gondolt_szam++;
  }  //do
  while (gondolt_szam <= 50);

  printf ("\n Aki bújt, aki nem, megyek !");
  getch ();
}    // program vége
```

Figyeljük meg a formai elemeket:

- a **do** után nem szabad pontosvesszőt tenni
- a **do** és a **while** között helyezkednek el a ciklusmag utasításai, ezek egy utasítás-blokkot alkotnak, kapcsos zárójelek {} között kell őket elhelyezni.
- a **while** új sorban kezdődik, utána zárójelben () kell megadni a kifejezés értékét, amely a lefutási feltételt megszabja. A zárójel után kötelező a pontosvessző ; .

A hátultesztelő ciklust gyakran alkalmazzuk ellenőrzött adatbevitelre. Ez azt jelenti, hogy addig folytatjuk egy meghatározott adat bekérését, amíg az az előírt határok közé nem esik.

2.3.3.2. feladat:
Készítsen programot, amely bekéri egy felnőtt testmagasság (110-210 cm)
és testsúly (30-150 kg) adatait, majd kiírja azokat. Csak a megadott határok közötti értékeket fogadhat el.

Megoldás:

Pontosítsuk a feladatot: Ha nem a megadott határok közötti értéket kaptam, akkor álljon le a program ? Válasz: Nem! a bekérést ciklikusan addig kell folytatni, amíg végre a helyes adatokat megkapjuk.

Elemezzük a feladatot:

Külön ciklussal fogjuk bekérni a magasságot, külön a testsúlyt. Így egyszerűbb a lefutási feltételek megfogalmazása és csak a hibásan megadott adat újra bevitelét kérjük.

A "bekérési" ciklus addig kell fusson, amíg hibás a kapott érték, tehát ezt kell megfogalmazni az **amíg** utáni kifejezésben.

Pszeudokód:

Program
Változók: magassag, suly (pozitív valós értékek)
Ciklus
Be: magasság [110-210 cm]
amíg (magassag < 110 vagy magassag > 210)
Ciklus vége
Ciklus
Be: suly [30-150 kg]
amíg (suly < 30 vagy suly > 150)
Ciklus vége
Ki: magassag, suly
Program vége

Kódoljuk a megoldást:

2.3.3.2. mintapélda, **hátultesztelő ciklus, ellenőrzött adat-bevitel**

```
#include <stdio.h>
#include <conio.h>
void main (void)
//testmagasság és testsúly adatok ellenőrzött bevitele
{
  float magassag, suly;
  do
  {
    printf ("\nKérem a testmagasságot cm-ben (110-210) :");
    scanf ("%f", &magassag); fflush (stdin);
  }   //do
  while ((magassag < 110) || (magassag > 210));

  do
  {
    printf ("\nKérem a testsúlyt kg-ban (30-150): ");
```

```
    scanf ("%f", &suly);  fflush (stdin);
  }   //do
  while ((suly < 30) || (suly > 150));

  printf("\nKöszönöm az adatokat: %5.1f, %5.1f",magassag, suly);
  getch ();
}    // program vége
```

Megjegyzés: Mi lehet a hiba, ha a while utáni feltételt biztosan jól megfogalmaztuk, a program futtatásakor mégsem áll le az adat bekérés, "végtelen" ciklus hoztunk létre ?
Ellenőrizze mégegyszer a feltételt, majd a scanf utasítást, ahol gyakran lemarad a cím operátor &.

A hátultesztelő ciklusnál a ciklusmag utasításait egyszer mindenképpen végrehajtja a rendszer. Ez nem mindig alkalmas szerkezet a feladataink megoldására.

Előltesztelő ciklus

Pszeudokódja: **(**Lefutási feltételt fogalmazunk meg:)

Ciklus amíg (*Lefutási feltétel)*
Cikusmag utasításai
Ciklus vége

Működése
A ciklusmag utasításait mindaddig kell végrehajtani. amíg a *Lefutási feltétel igaz.* Ha a *lefutási feltétel* hamissá válik, a végrehajtás a ciklus vége után folytatódik. Ha a *lefutási feltétel* már a ciklusba való belépéskor hamis, a ciklusmag utasításai egyszer sem hajtódnak végre.

A C nyelvi programszerkezet:

```
while (kifejezés)     //lefutási feltétel
{
  ciklusmag utasításai
}
```

A **while** ciklus mindaddig ismétli a hozzá tartozó utasításokat (a ciklusmag utasításait, más szóval a ciklus törzsét), amíg a vizsgált kifejezés (lefutási feltétel) értéke nem nulla (igaz). A vizsgálat mindig megelőzi a ciklusmag utasításainak végrehajtását, ezért van olyan eset, amikor a ciklusmag utasításai egyszer sem kerülnek végrehajtásra.

2.3.3.3. feladat
Állítsunk elő 0 és 50 közötti véletlenszámokat addig, amíg a 25-ös szám generálása is megtörtént. Írassuk ki a kapott számokat.

Megoldás:

Elemezzük a feladatot:
Állítsunk elő egy véletlen-számot, majd ismételjük az alábbi tevékenységeket:
Ha a szám nem egyenlő 25-tel, akkor:
írjuk ki
állítsunk elő egy újabb véletlen-számot.

Pszeudokód:

Program
Változók: szam (pozitív egész)
véletlenszám-generátor inicializálása
szam = random (51)
Ciklus amíg (szam <> 25)
Ki: szam
szam = random (51)
Ciklus vége
Ki : "Megvan a 25-ös !"
Program vége

Kódoljuk a megoldást:

2.3.3.3. mintapélda, előltesztelő ciklus

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main (void)
//előltesztelős ciklus: véletlen számok előállítása, amíg 25-öt kapunk
{
  int szam;           //az előállított váletlen-szám tárolására
  randomize ();       //a véletlenszám generátor inicializálása
  szam = random (51); //0 és 50 közötti véletlenszám előűllítása
  while (szam != 25)
  {
     printf ("\n/%d ",szam);
     szam = random (51);
  }
  printf ("\n...Megvan a 25-ös !");
  getch ();
}    // program vége
```

Futtassuk többször a programot ! Az esetek többségében azt tapasztaljuk, hogy a képernyőre írt adatok kifutnak, nem látjuk az adatsor elejét. Jó lenne azt is tudni, hogy hányadikként sikerült a 25-öt előállítani. A későbbiekben erre még visszatérünk.

Növekményes (számláló) ciklus

Gyakran előforduló feladat, hogy valamely tevékenység(ek)et előre megadott számszor kell végrehajtani. Erre a feladatra a programnyelvek egy külön ciklus-utasítást biztosítanak, a növekményes, számláló, vagy léptetéses ciklust.

Pszeudokódja:

Ciklus *ciklusváltozó = ...***kezdőértéktől .... végértékig .... lépésközzel**
Ciklusmag utasításai
Ciklus vége

Működése:
A ciklusmag utasításai a ciklusváltozó kezdő és végértékének, valamint a lépésszámnak megfelelő számszor kerülnek végrehajtásra.

A C nyelvi programszerkezet:

A C nyelvi megvalósítás valójában sokkal tágabb, mint amit a pszeudokódban feltüntettünk.

```
for (init_kif; feltétel_kif; léptető_kif)
{
    ciklusmag utasításai
}
```

A for ciklus végrehajtásának lépései:

- Megtörténik az init_kif kifejezés kiértékelése. Az init_kif típusára vonatkozóan semmilyen megkötés nincs, megadása nem kötelező. Általában arra használjuk, hogy a ciklusban használt objektumokat inicializáljuk.
- A következő lépésben a feltétel_kif (aritmetikai vagy mutató típusú) kifejezést dolgozza fel a rendszer (ha megadtuk). Ha a feltétel_kif nincs megadva, annak értékét igaznak tételezi fel a rendszer.
- Ha a feltétel_kif értéke nem nulla (igaz), akkor végrehajtódnak a ciklusmag utasításai, majd a léptető_kif kifejezés kiértékelése következik. Minden további iterációt a feltétel_kif kiértékelése nyit meg, s annak értékétől függ a folytatás.
- Ha a feltétel_kif értéke nulla (hamis), akkor a **for** utasítás befejezi a működését és a vezérlés a ciklusmag utasításait tartalmazó blokk után következő utasításra adódik.

A **for** ciklus a **while** ciklus egy speciális alkalmazásának is tekinthető, így a fenti ciklus átírható az alábbi szerkezetre:

```
init_kif;
while (feltétel_kif)
{
  ciklusmag utasításai
  léptető_kif
}
```

2.3.3.4. feladat

Készítsen programot, amely n darab * karaktert ír ki a képernyőre.

Megoldás:

Pontosítsuk a feladatot:
Az n értékét honnét veszem ? Válasz: be kell kérni a felhasználótól.

Pszeudokód:

Program
Változók: n - darabszám, pozitív egész
i - ciklusváltozó, pozitív egész
Ciklus
Be: n
amíg (n<= 0)
ciklus vége

Ciklus i= 1 **kezdőértéktől** n **végértékig** 1 **lépésközzel**
Ki: "*"
ciklus vége
Program vége

Kódoljuk a megoldást:

2.3.3.4. mintapélda, egyszerű for ciklus n db * karakter kiírására

```
#include <stdio.h>
#include <conio.h>
void main (void)
//for ciklus: n darab * karakter kiírása
{
  int i, n;       //i a ciklusváltozó, n a darabszám
  do
  {
    printf ("\nHány csillag legyen a képernyőn ?");
    scanf ("%d", &n); fflush (stdin);
  }  //do
  while (n <= 0);

  for (i=1; i<= n; i++)
  {
     printf ("*");
  }
  getch ();
}    // program vége
```

Figyeljünk a szintaxisra ! A for utasításban a csukó zárójel után általában nincs pontosvessző ! Ha mégis kitesszük, az azt jelenti a fordító számára, hogy a ciklusmaghoz csak egy üres utasítás tartozik.

Az alábbi, egyetlen **for** utasítással az első n természetes szám összegét képezzük:

```
for (i=1, osszeg=0; i <=n; i++);
```

Az init_kifejezés két tagból áll, őket vessző választja el egymástól. A módosító részben is meg lehet adni több kifejezést. A feltétel_kifejezésben is megenged a fordító több kifejezést, de csak az elsőt veszi figyelembe, ezért a továbbiak megadása megtévesztő lehet, kerüljük ezt a megoldást.

A ciklusokat egymásba is ágyazhatjuk, hiszen a ciklusmagban újabb ciklusutasítás is elhelyezhető. Leggyakrabban a **for** ciklusokat szokás egymásba ágyazni.

2.3.3.5. feladat
Készítsen szorzótáblát !

Megoldás:

Elemezzük a feladatot:
A képernyőre először kiírjuk a SZORZÓTÁBLA feliratot középre, majd egy külön sorban a számokat 1-től 9-ig. (szorzó1).
Ezután sorra vesszük a másik szorzótényezőt (1-től 9-ig), nevezzük ezt szorzó2-nek, és soronként kiírjuk: a következő szorzót, és a szorzatokat egymás mellé.
Két ciklust kell szerveznünk. A képernyőre úgy tudunk a legegyszerűbben egy táblázatot kiírni, hogy vesszük a táblázat sorait egymás után (ez lesz a külső ciklus, szorzo2-vel), és a soron belül vesszük az oszlopokat egymás után (ez lesz a belső ciklus szorzo1-gyel).

Pszeudokód:

Program
Változók : szorzo1, szorzo2: pozitív egész
Ki: "SZORZÓTÁBLA"
//Következik a fejléc-sor kiírása
Ciklus szorzo1 = 1 **kezdőértéktől** szorzó1 = 9 **végértékig** 1-es **lépésközzel**
Ki: szorzo1
Ciklus vége //szorzo1
// a szorzótábla elemei
Ciklus szorzo2 = 1 **kezdőértéktől** szorzó2 = 9 **végértékig** 1-es **lépésközzel**
Ciklus szorzo1 = 1 **kezdőértéktől** szorzó1 = 9 **végértékig** 1-es **lépésközzel**
Ki: szorzo1 * szorzo2
Ciklus vége *//szorzo1*
Ciklus vége *//szorzo2*
Program vége

Kódoljuk a megoldást:

2.3.3.5. mintapélda, **egymásba ágyazott ciklusok: szorzótábla**

```
#include <stdio.h>
```

```
#include <conio.h>
void main (void)
//egymásba ágyazott for ciklusok: szorzótábla kiiratás
{
  int szorzo1, szorzo2;
                //a szorzótényezők, egyben a ciklusváltozók is
  printf ("\n              SZORZÓTÁBLA\n\n     ");
  for (szorzo1=1; szorzo1<10; szorzo1++)
  {
     printf ("%3d", szorzo1);
  }  //for szorzo1
  printf ("\n");
  for (szorzo2=1; szorzo2<10; szorzo2++)   //Külső for ciklus
  {
     printf ("\n%5d",szorzo2);
     for (szorzo1=1; szorzo1<10; szorzo1++)  //Belső for ciklus
     {
     printf ("%3d",szorzo1 * szorzo2);
     }  //szorzo1
  }  //szorzo2
  getch ();
}    // program vége
```

A break és a continue utasítások

A már megismert **break** utasítás hatására az utasítást tartalmazó legközelebbi **switch**, **while**, **for** és **do-while** utasítások működése megszakad és a vezérlés a megszakított utasítás utáni első utasításra kerül. Felhívjuk a figyelmet arra, hogy egymásba ágyazott ciklusok, illetve a fenti utasítások egymásba ágyazása esetén a **break** utasítás mindig csak a legbelső utasításból való kilépést idézi elő.

2.3.3.6. feladat
Keresse meg programmal a 2001 szám legnagyobb osztóját !

Megoldás:

Elemezzük a feladatot:
Egy egész szám osztója legalább kétszer megvan a számban, tehát a legnagyobb osztó keresését elegendő a szám felénél kezdeni, majd egyesével haladni lefelé, amíg az első olyan számot megtaláljuk, amellyel osztva 2001-et, a maradék = 0-

Pszeudokód:

Program
Változók : oszto pozitív egész
Ciklus oszto = 2001/2 **kezdőértéktől** 2 **végértékig** -1-es **lépésközzel**
Ha (2001 mod oszto = 0)
Ki: "A legnagyobb osztó:", oszto

kiugrás
ciklus vége
Ki: "A vizsgálat befejeződött."
Program vége

Kódoljuk a megoldást:

2.3.3.6. mintapélda, **kiugrás a ciklusmagból: a 2001 egész szám legnagyobb osztója**

```
#include <stdio.h>
#include <conio.h>
void main (void)
//kiugrás a ciklusmagból: egész szám legnagyobb osztója
{
  int oszto;
  printf ("\n2001 legnagyobb osztóját keressük.");
  for (oszto = 2001/2; 1; oszto--)
  {
     if (2001%oszto == 0)
     {
       printf ("\n2001 legnagyobb osztója: %d", oszto);//megvan !
       break;
     }
  }
  printf ("\nA vizsgálat befejeződött.");
  getch ();
}    // program vége
```

A **continue** utasítás a **while**, **for** és a **do-while** ciklusok soron következő iterációját indítja el, a ciklustörzsben a **continue** utasítást követő utasítások átlépésével. A **for** ciklus esetében a feltétel kifejezés kiértékelését megelőzi a léptető kifejezés feldolgozása.

A **break** és a **continue** utasítások gyakori használata rossz programozási gyakorlatot jelent. Az egyszerűbb feladatok többnyire az áttekinthetően megoldhatók e két utasítás használata nélkül is. A **continue** utasításra példát a későbbiekben fogunk látni.

Ellenőrző kérdések

- Milyen programszerkezetek szolgálnak az utasítások ismételt végrehajtására ?
- Melyek az előltesztelő és a hátultesztelő utasítások ? Írja fel a pszeudokódot, a C nyelvi megvalósítást, ismertesse a működést.
- Lehet-e kiugrani ciklus utasításokból, ha igen, akkor hogyan.
- Mi a hatása a ciklusmagban elhelyezett continue utasításnak ?
- Mi lesz az i változó értéke a ciklus lefutása után ?
  - a./ for (i = 1; i < 5; i++);
  - b./ for (i = 1; i > 5; i--);
  - c./ for (i = 0; i < 3; i = i +2);

Gyakorló feladatok

233.1. feladat
Írassuk ki 1-től n-ig a számokat és a négyzetüket.

233.2. feladat
Írjuk teli Z betűvel a képernyő első 10 sorát. Minden sor elején a sor sorszáma jelenjen meg.

233.3. feladat
Tetszőleges betűvel tetszőleges sort (1-10) töltsünk fel a képernyőn.

233.4. feladat
Írassuk ki 100-tól lefelé csökkenő sorrendben az összes pozitív, 7-tel osztható egész számokat !

233.5. feladat
Írassuk ki az első 10, 5-tel osztható egész szám felét 2 tizedesjegy pontossággal!

233.6. feladat
Írassuk ki 200-tól 50-g csökkenő sorrendben a 7-tel osztható egész számokat és a négyzetüket.

233.7. feladat
Írassuk ki 200-ig az öttel osztható számok közül azokat, amelyek nem oszthatók 25-tel..

233.8. feladat
Írassuk ki programmal az első n páratlan szám köbét !

233.9. feladat
Készítsen kódtáblázatot ! Írassa ki 1-től 255-ig a kódot, és mellé a kódnak megfelelő karaktert. (A kód maga a szám, ha ezt a számot karakter formátummal íratjuk ki, akkor a képernyőn a kódnak megfelelő karakter jelenik meg.)

233.10. feladat
Készítsen függvény-táblázatot: A táblázatnak 3 oszlopa lesz, az elsőben a szög fokban, a másodikban a szög színusza, a harmadikban a szög koszinusza szerepel. A szög értéke 0-tól 360 fokig terjedjen, bekért lépésközzel.

233.11. feladat
Készítsen táblázatot az 1/(1-x) függvényről a (0,5) intervallumban. Kerülje el a nullával való osztást !

233.12. feladat
Készítsen programot, amely számtani sorozatot állít elő. Be kell kérni az első elem értékét, a lépésközt és az elemszámot.

233.13. feladat
Írjuk tele a képernyőt valamilyen karakterrel !
Írjuk tele a képernyő minden második (harmadik, stb.) sorát valamilyen karakterrel.

233.14. feladat

Készítsen programot, amely "kikérdezi " a szorzótáblát. Véletlenszerűen ad feladványt a szorzótáblából (pl. mennyi 5 * 7), a kapott választ elemzi és csak akkor áll le a kikérdezés, ha egymás után öt jó válasz érkezett.

233.15. feladat

Készítsen programot, amely megkeresi azokat a 10-es számrendszerbeli háromjegyű számokat, amelyeknek a számjegyeit köbre emelve és összeadva eredményül magát a számot kapjuk (Armstrong-féle számok).