Problémaosztályok, algoritmusok

Bevezetés

Alapfeladatok és algoritmusok

Algoritmus készítés

Áttekintés:

- Algoritmus fogalma, tulajdonságai
- Algoritmus tervezés lépései, moduláris programozás
- Strukturált program fogalma, elemei
- Algoritmus leíró eszközök , pszeudokód

Probléma megoldás …

Nyaralás, tengerpart, vizesárok feltöltése

- Vedd a kezedbe a kisvödröt

- Ismételd az alábbiakat addig, amíg tele nem lesz a vizesárok:
- Merítsd tele a kisvödröt
- Vidd a vizesárokhoz és öntsd bele a vizet
- Ismétlés vége

- Tedd le a kisvödröt

Bár a fenti tevékenységek ismétlést tartalmaznak, véges számú lépésben a feladat megoldásához vezetnek (tele lesz a vizesárok)

Az algoritmus fogalma …

Szinonímák: eljárás, recept, módszer.

Eredet: IX. sz. Bagdad: dec. számokkal való számolási eljárások.

Számítógépes algoritmus = program ?
Több tízezer utasításból álló adatbáziskezelő program ??

Max. néhány száz soros C programban kódolható feladat.
A módszerek elemzése, hatékonyság vizsgálat

Nagyméretű feladatok: Szoftvertechnológia

Az algoritmus fogalma

Algoritmus: az egyértelmüen előírt módon és sorrendben végrehajtandó tevékenységek ***véges*** sorozata, melyek alapján egy feladattípus ***véges*** számú lépésben megoldható.

Az algoritmus leírásának egyértelműnek, pontosnak, lépésenként ***végrehajthatónak*** kell lennie.

Az algoritmus leglényegesebb tulajdonságai, és a vele szemben támasztott követelmények

- Lépésekből (elemi tevékenységekből) áll
- Minden lépésnek egyértelműen végrehajthatónak kell lennie
- Hivatkozhatunk összetett lépésekre is, részletezés később
- Mindig van tárgy, amit itt adatnak hívunk (absztrakció: csak a feladat végrehajtásához szükséges adatokat vesszük figyelembe)
- A végrehajtandó utasításoknak célja van, végrehajtás során változnak az adatok bizonyos tulajdonságai
- Az algoritmusnak általában vannak bemenő adatai
- Legalább egy kimenő adat, kommunikáció a külvilággal
- Véges számú lépésben el kell vezessen a megoldáshoz
- Legyen hatékony
- Legyen felhasználóbarát.

Algoritmus tervezés, moduláris programozás

- Feladat specifikáció
- Adatok tervezése
- Moduláris programozás

Feladat specifikáció

- Az algoritmus tervezésének első lépése a feladat körültekintő és alapos elemzése, a probléma megfogalmazása. A feladat leírását feladat specifikációnak is szokás nevezni, ezt írásban is rögzíteni kell.

- Az algoritmusok leírása:
  - mondatszerű leírás,
  - folyamatábra,
  - stuktogram,
  - Jackson jelölés, …

- A feladat megfogalmazása egyértelmű legyen, valamennyi lehetséges esetet lefedjen.

Adatok tervezése

Mivel az algoritmus adatokon dolgozik, meg kell tervezni a feladatban szereplő külső és belső adatokat, adat szerkezeteket. Csak ezután kezdhetünk hozzá az adatokat mozgató, manipuláló lépések megtervezéséhez.

Példa: tankönyv, **A4**-es méretben **L1** oldal, **K1** karakter/oldal

B5 méretben **K2** karakter/oldal

kérdés: **L2**

adatok: K1, L1, K2, L2 mértékegység, értéktartomány -> típusok

Moduláris programozás

- Az algoritmus (program) tervezésének alapja a **dekompozíció**. A bonyolultabb feladatokat részekre, modulokra kell bontani. A részekre ki kell dolgozni a megoldás menetét, majd a részeket újra össze kell állítani, hogy azok együttműködve, egymásra épülve a teljes feladat megoldását szolgáltassák.

- **A moduláris programozás olyan programozási mód, melyben a teljes program modulokból áll. Az egyes modulok jól meghatározott részfeladat megoldását végzik, kezelhető méretűek, egyértelmű céljuk van és jól definiáltan csatlakoznak a program többi moduljához.**

A moduláris programozás irányelvei

- **"Oszd meg és uralkodj" elv**:
  - A feladatokat egyértelműen le kell osztani modulokra.
  - A modulok belső működésébe más modul nem szól bele.
  - A modul a saját feladatának tökéletes elvégzéséért felelős.
  - Fontos, hogy a modulok lehetőleg egymástól függetlenül működjenek, mert így a hibák kiszűrése egyszerűbb, a feladat átláthatóbb, könnyebben módosítható.
- **Adatok (információ) elrejtésének elve**:
  - Az egyes modulok csak saját adataikon dolgozzanak, csak akkor használjanak közös adatokat, ha ez elkerülhetetlen.
- **Döntések elhalasztásának elve**:
  - Csak akkor hozzunk meg egy döntést, ha az elengedhetetlenül szükséges.
  - Mindazon döntéseket, amelyekhez még nincs elég ismeretünk, halasszuk későbbre - különben előfordulhat, hogy egy korán meghozott döntést a későbbiekben meg kell változtatnunk.
- **Döntések kimondásának elve:**
  - A feladat megoldása során, ha már meghoztunk egy döntést, azt le kell rögzíteni, mert különben erről később esetleg megfeledkezve, ellentmondásos döntéseket hozhatunk.

A modulokra bontás iránya

- **Felülről lefelé** (top-down) tervezés esetén a megoldást felülről lefelé, fokozatosan, lépésenként finomítjuk. A feladatot részfeladatokra, a részfeladatokat ismét kisebb feladatokra bontjuk mindaddig, amíg az úgynevezett elemi tevékenységekhez jutunk. Az elemi tevékenységek tovább nem bonthatók, egy lépésben végrehajthatók.

- **Az alulról felfelé** (bottom-up) tervezés lényege, hogy már meglévő, kész modulokból építkezünk. Erre akkor kerül sor, ha bizonyos részfeladatokat egy előző feladat kapcsán már megoldottunk (gondolva a későbbi felhasználásokra is), vagy amikor egy rutingyűjteményt (szoftvert) vásároltunk.

Algoritmus tervezés, példa

- **Feladat:** Készíts programot, amely egy hőmérsékletet átalakít Celsius-fokból Kelvin-fokba

- **Feladat pontosítása**: °K= °C+271

- **Lépések (modulok) megtervezése**:
  1. Készíts programot: *#include <stdio.h>, void main()*, stb...
  2. egy (felhasználó által megadott) hőmérséklet (beolvasni)
  3. átalakít °C=>°K
  4. eredmény kiír
- **Adatok**:
  - Celsius-fok, Kelvin-fok

Algoritmus tervezés, példa

- **Lépések**:
  1. Készíts programot
  2. Beolvas egy hőmérséklet
  3. Átalakít °C=>°K
  4. Eredmény kiír

- **Adatok**:
  - Celsius fok: fokCels
  - Kelvin fok: fokKelv

```
#include <stdio.h>
void main()
{
    int fokCels, fokKelvin;
    printf("Homerseklet °C: ");
    scanf("%d",&fokCels);
    fokKelvin=fokCels+271;
    printf("Az eredmeny: %d",fokKelvin);
}
```

Algoritmus tervezés, mondatszerű leírás

- Lépések:
  1. ~~Készíts programot~~
  2. Beolvas egy hőmérséklet
  3. Átalakít °C=>°K
  4. Eredmény kiír

Változó: fokCels, fokKelv

Be: fokCels

fokKelv=fokCels+271

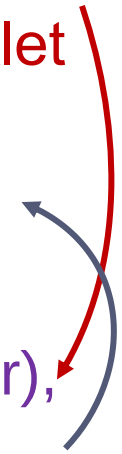
Ki: fokKelv

- Változók:
  - Celsius fok: fokCels
  - Kelvin fok: fokKelv

Algoritmus tervezés, példa

- **Feladat:** Készíts függvényt, amely egy hőmérsékletet átalakít Celsius-fokból Kelvin-fokba

- **Lépések megtervezése**:
  1. Készíts függvényt
  2. egy (paraméterekben megadott) hőmérséklet
  3. átalakít °C=>°K
  4. eredmény visszaad
- **Adatok**:
  - Celsius-fok (paraméter), Kelvin-fok (eredmény)



```
int atalakitC_K(int fokCels)
{
    int fokKelv;
    fokKelv=fokCels+271;
    return fokKelv;
}

int k = atalakitC_K(21);
```

Algoritmus tervezés: példa 2.

- Olvass be egy egész számokból álló tömböt, és keresd meg a legnagyobb elemét

- Lépések
  1. (Írj programot)
  2. Olvass be egy tömböt
  3. Keresd meg a legnagyobb elemét
  4. (Írd ki az eredményt)
- Változók
  - Tömb: *integer*, max 20 elem
    tömb hossza: n
  - Legnagyobb elem

```
#include <stdio.h>
void main()
{
   int tomb[20];
   int i,n;
   int max;
   printf("Hany elembol all a tomb?");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("Kerem a %d. elemet:",i);
      scanf("%d",&tomb[i]);
   }
   max=tomb[0];
   for(i=0;i<n;i++)
   {
      if(tomb[i]>max) max=tomb[i];
   }
   printf("A tomb legnagyobb eleme: %d",max);
}
```

Strukturált program, mondatszerű leírás

Tetszőleges program felépíthető az alábbi **három** elemből:

- **Szekvencia**: Egymás után végrehajtandó tevékenységek sorozata
- **Szelekció**: Választás megadott tevékenységek közül
- **Iteráció**: Megadott tevékenységek ismételt végrehajtása

- ***Feltétel nélküli ugrás****: Vezérlés átadása a program egy megadott pontjára*.

Böhm és Jacopini tétele

A szekvencia, szelekció és az iteráció segítségével minden olyan algoritmus felépíthetö, amelynek egy belépési és egy kilépési pontja van

Strukturált program, mondatszerű leírás

- **Strukturált program:** A csak szekvenciákból, szelekciókból és iterációkból építkezö programot strukturált programnak nevezzük. A strukturált programozásban **nem** megengedett a feltétel nélküli ugrás, így többek között a ciklusból sem ugorhatunk ki.

  Ebböl az is következik, hogy a program minden vezérlö szerkezetének (szekvencia, szelekció, iteráció) - és magának a teljes programnak is - egyetlen belépési és egyetlen kilépési pontja van, így a program lényegesen áttekinthetőbb.

  A strukturált programozás általános módszereit E. W. Dijkstra dolgozta ki

Strukturált program, mondatszerű leírás

- **Kísérleti tapasztalatok:**

Gyakorlatilag nem léteznek valós problémák megoldásán alapuló bizonyítékok a strukturált programozás mellett. (Kétszer kellene megíratni egy nagy programrendszert …)

Vizsgálatok folytak arra nézve, hogy gyakorlott programozók mennyire könnyen értenek meg egy forráskódot, ha az strukturált, vagy strukturálatlan. (Milyen mélységig látták át a forráskódot)

Strukturált program, mondatszerű leírás

- **Mondatszerű leírás**

  A mondatszerű leírás, vagy **pszeudokód** lényege, hogy az algoritmust mondatszerü elemekből építjük fel. Ez nagyon hasonlít a beszélt nyelvre, illetve annak írásos formájára, de be kell tartanunk bizonyos szabályokat, a struktúrák képzésére megállapodás szerinti formákat és szavakat használunk.

Mondatszerű leírás, alapelemek

Változók

Bevitel, kivitel

Szekvencia
- értékadó utasítás

Szelekciók
- egyágú
- két- és többágú
- többágú, kapcsoló típusú

Iteráció (Ciklusok)
- elől tesztelő
- hátul tesztelő
- növekményes

Program

Változók: szam: egész
Be: szam
Ha szam < 0
    Ki: ”Negatív”
Elágazás vége
Ha szam == 0
    Ki: ”Nulla”
Elágazás vége
Ha szam >0
    Ki: ”Pozitív”
Elágazás vége
Program vége

Mondatszerű leírás, alapelemek

Változók: *változók felsorolása , értéktartomány, mértékegység*

Bevitel, kivitel:

Be: ... *változók felsorolása ...[.....]*

az adatokkal szemben támasztott követelmények

Ki: *... kifejezések felsorolása... [.....]*

a kiírás formájára vonatkozó követelmények

Szekvencia: Az egymás után végrehajtandó tevékenységek sorozata.

Tevékenység1
Tevékenység2
Tevékenység3

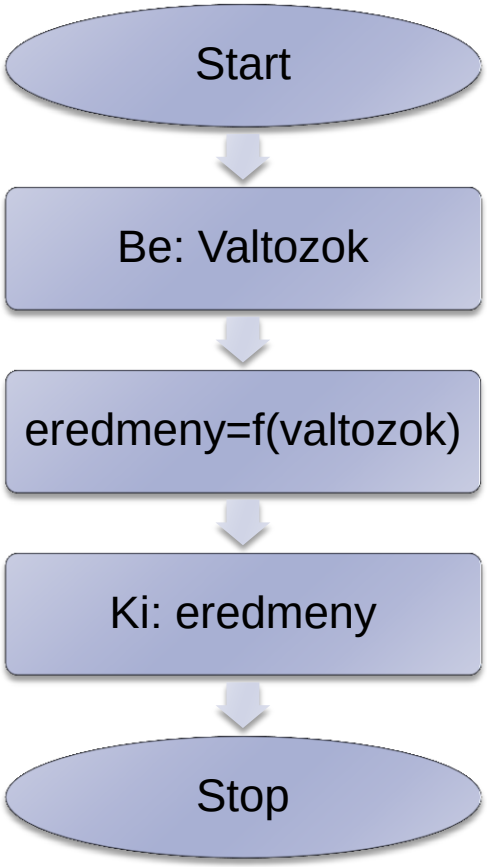
Az egy sorba írt tevékenységeket kettőspont választja el:

Tevékenység1 : Tevékenység2 : Tevékenység3

Értékadó utasítás: a szekvencia egy speciális fajtája, formája:

változó = kifejezés

Folyamatábra, struktogram



Példa: bevitel, szekvencia, kivitel

Egy raktárban ömlesztett árut tárolnak nagyobb zsákokban.

A kiszállításhoz kisebb dobozokba csomagolják az árut.

Adott mennyiség kiszállítása után mennyi marad raktáron és az hány nagy zsákban fér el.

Megoldás:

Pontosítás: mértékegység, mi az az adott mennyiség …?

Adatok: kiindulás: zsakdb, zsakliter,

kiszállítás: dobozdb, dobozliter

marad: zsakdb … ??

Be: zsakdb, zsakliter, dobozdb, dobozliter

marad = zsakdb – dobozdb*dobozliter/zsakliter

Ki: marad

Példa: bevitel, szekvencia, kivitel

A megoldás pszeudokódja:

Program

Változók:

zsakdb, zsakliter, (pozitív egész értékek 0..10000)

dobozdb, dobozliter (pozitív egész értékek 0..10000)

marad (poz. egész)

Be: zsakdb, zsakliter, dobozdb, dobozliter

marad = zsakdb – dobozdb*dobozliter/zsakliter

Ki: marad

Program vége

Mondatszerű leírás, Szelekciók

Szelekciók (elágazások, feltételes utasítások):

Programrészek közötti választás

Egyágú szelekció

Ha *Feltétel* **akkor**

Utasítás(ok)

Elágazás vége

Jelentése:

Ha *Feltétel* teljesül, akkor az *Utasítás(ok)* végrehajtásra kerülnek, egyébként nem. A program az *Elágazás vége* után folytatódik. (Az elágazás vége kiírás el is hagyható.)

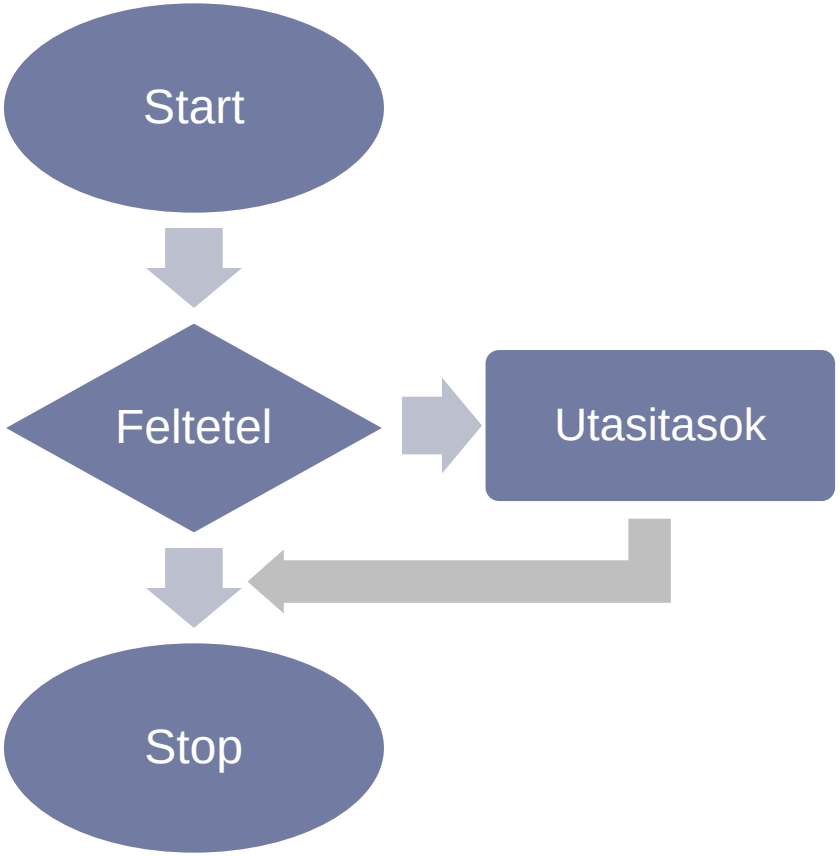
PÉLDA: Elindulás autóval:

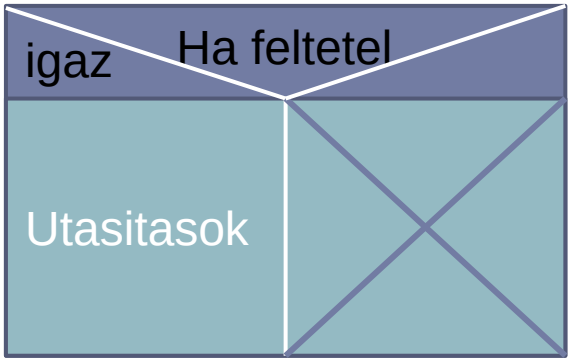
Indítsd be a motort, stb

Ha éjjel van, gyújtsd fel a lámpát

Rakd egyesbe és adjál gázt

Folyamatábra, struktogram





Egyágú szelekció kódolása a C-ben

az if utasítás:

```
if (kifejezés)
{
    utasítás szekvencia
}
```

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

Egyágú szelekció, példa

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

Megoldás:

Pontosítsuk a feladatot:

Legfeljebb mekkora lehet a szám ? Válasz: max. 3 jegyű, pozitív

Ha páros számot kaptunk, akkor ne csináljunk semmit ?

Válasz: páros szám esetén nincs semmi teendő !

Pszeudokód:

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

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

Egyágú szelekció, példa

Pszeudokód:

Program
Változók: szam (0 ... +999)
Be: szam
Ha (szam maradék 2 != 0)
Ki: "Páratlan !”
Elágazás vége
Program vége

C program:

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

Kétágú szelekció

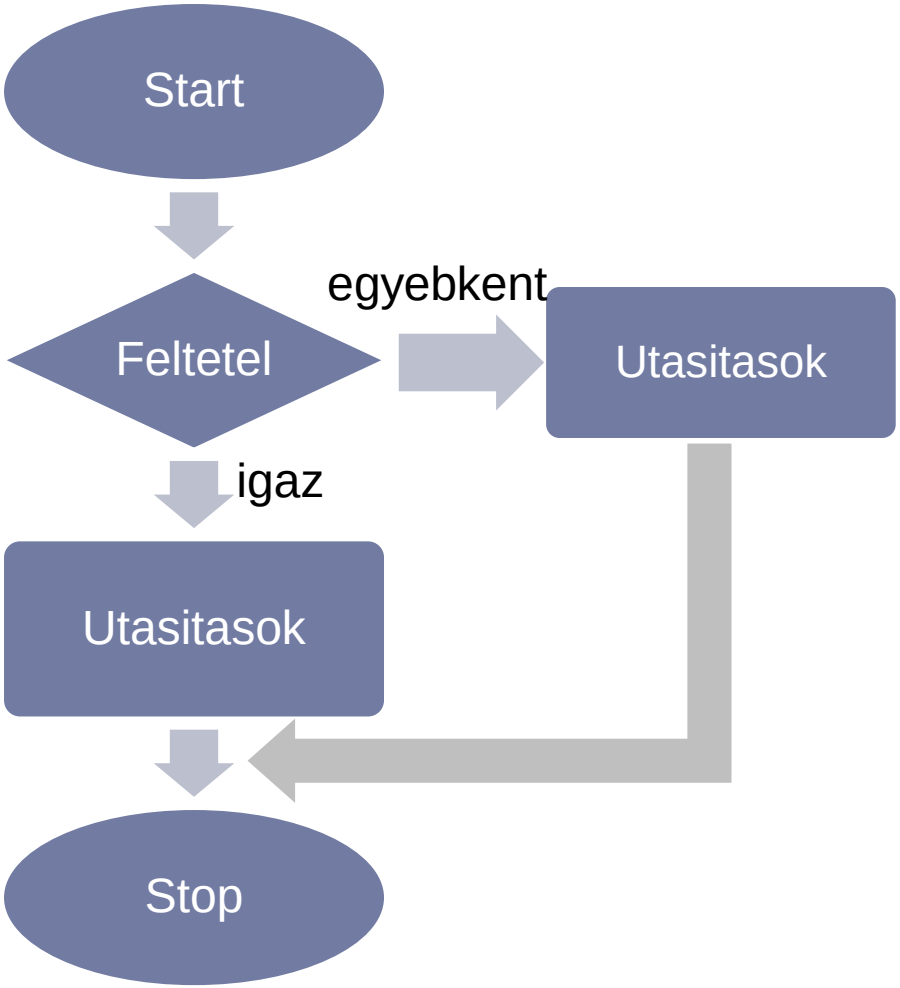
HA feltétel **akkor**
Utasítás(ok)1
Egyébként
Utasítás(ok)2
Elágazás vége

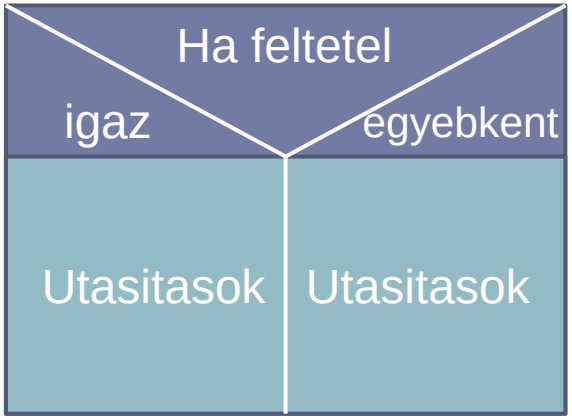
Jelentése: Ha *Feltétel* teljesül, akkor az *Utasítás(ok)1 kerül(nek)* végrehajtásra, egyébként Utasítás(ok)2. A program mindkét esetben az *Elágazás vége* után folytatódik

PÉLDA:

Ha *süt a nap,* **akkor**
kimegyek napozni
egyébként
internetezek a szobámban
Elágazás vége

Folyamatábra, struktogram





Kétágú szelekció kódolása C-ben

HA feltétel **akkor**

Utasítás(ok)1

Egyébként

Utasítás(ok)2

Elágazás vége

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.

Kétágú szelekció, 1. feladat

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

Megoldás:

Pszeudokód:

Program
Változók: szam (-999 ... +999)
Be: szam
Ha (szam maradék 2 != 0) **akkor**
Ki: "Páratlan !"
Egyébként
Ki: "Páros!”
Elágazás vége
Program vége *

Kétágú szelekció, 1. feladat, kódolás

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

```
#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                                    *
```

Kétágú szelekció, 2. 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
     Elágazás vége
Program vége
```

Kétágú szelekció, 2. feladat

```
#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                                                    *
```

Kétágú szelekció, 3. 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:
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 !"

- Pontosítsuk a feladatot:

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

Kétágú szelekció, 3. feladat megoldása: pszeudokód

Ha a magasság 100 cm fölötti, akkor megvizsgálja, hogy túlsúlyos-e:
ha igen, akkor
kiírja: "Túlsúlyos, fogynia kell!".
Ha a magasság 100 alatti érték,
akkor írja ki a program, hogy "Gyerekekkel nem foglalkozom !”

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 !”
Elágazás vége
Egyébként
Ki: "Gyerekekkel nem foglalkozom !”
Elágazás vége
Program vége

Kétágú szelekció, 3. feladat megoldása: C program

Ha (magassag > 100) **akkor** ***//1.***
  Ha (suly > magassag -100) ***//2.***
    Ki: "Ön túlsúlyos !„
  Elágazás vége ***//2.***
Egyébként ***//1.***
  Ki: "Gyerekekkel nem foglalkozom !”
Elágazás vége ***//1.***

```
void main (void)
{
 int magassag, suly;
 printf("\nKérem a testmagasságot cm-ben:");
 scanf("%d",&magassag);
 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 !");
}   // program vége
```

Egymásba ágyazott szelekciók

A szelekció IGAZ és HAMIS ágán is állhat újabb szelekció. Ezeket egymásba ágyazott szelekcióknak nevezzük.

4. 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-)...

Pontosítsuk a feladatot: Negatív életkort is megadhat a felhasználó ?

4. feladat megoldása: pszeudokód

Program
Változók: eletkor, (pozitív egész)
Be: eletkor
Ha (eletkor < 0) **akkor**
Ki: "Hibás adat !"
Egyébként ha (eletkor < 15) **akkor**
Ki: "Gyerek vagy még !"
Egyébként ha (eletkor < 24) **akkor**
Ki: "Kamaszkorban a legnehezebb !"
Egyébként ha (eletkor < 63) **akkor**
Ki: "A felnőtteknek sem könnyű ..."
Egyébként
Ki: "Idősebbek bölcsebbek is ?…"
Elágazás vége
Program vége

Hogyan kódoljuk az Egyébként ha szerkezetet ?

4. feladat megoldása: C program

```
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... !");
```

A program többirányú elágaztatása

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
```

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 akárhány lehet.

5. feladat: többirányú elágazás

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

Megoldás:

Pontosítsuk a feladatot:

Ha nem műveleti jelet kaptunk …
Mi történjen, ha nullával való osztás a kijelölt művelet ?.

5. feladat megoldása: pszeudokód

Program
Változók: x, y, eredmeny
op: a műveleti jel tárolására
siker: egész t. segédv., értéke:
1: a művelet elvégezhető
0: a művelet nem végezhető el
-1: hibás műveleti jel
Be: x, op, y [szóközök nélkül !]
Ha (op == '+') **akkor**
eredmeny = x + y
Egyébként ha (op == '-') **akkor**
eredmeny = x - y
Egyébként ha (op == '*') **akkor**
eredmeny = x * y
Egyébként ha (op == '/') **akkor**
Ha (y <>0) **akkor**
eredmeny = x / y
Egyébként
siker = 0
Egyébként
siker = -1
Elágazás vége

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 "
Elágazás vége

Program vége

Kódolás: else if szerkezettel…

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 …

5. feladat megoldása: pszeudokód

Elágazás (op) **szerint**
'+' **esetén :** eredmeny = x + y **kiugrás**
'-' **esetén :** eredmeny = x - y **kiugrás**
'*' **esetén :** eredmeny = x * y **kiugrás**
'/' **esetén :**
Ha (y <>0) **akkor**
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

Többágú kapcsoló

Elágazás kifejezés szerint

értékcsoport1 **esetén** *Utasítás(ok)1*

értékcsoport2 **esetén** *Utasítás(ok)2*

.....

értékcsoportn **esetén** *Utasítás(ok)2*

Más esetben *Utasítás(ok)n+1*

Elágazás vége

Jelentése:

Ha a kifejezés értékeaz i-dik (i=1, 2, ...n) értékcsoportba esik, akkor a neki megfelelő utasítás(oka)t kell végrehajtani, majd a feladat megoldása az **Elágazás vége** után folytatódik.

Az egyes értékcsoportok véges, diszkrét halmazokat alkotnak.

FONTOS: az elágaztatás nem logikai feltétel szerint történik !

Többágú kapcsoló: C program

- Pszeudokód:

Elágazás kifejezés szerint

érték1 **esetén**
Utasítás(ok)1

érték2 **esetén**
Utasítás(ok)2
.....

Más esetben
Utasítás(ok)n+1

Elágazás vége

- C nyelvben:

```
switch (egesz tipusu változo)
{
        case ertek1:
                utasitas-blokk1
                break;
        case ertek2:
                utasitas-blokk2
                break;
        default:
                utasitas-blokkN
                break;
}
```

Többágú szelekció

- PÉLDA:

Elágazás a lámpa színe szerint

Piros lámpa esetén Megállok : Kézifék be.

Sárga lámpa esetén Megállok : Kézifék be

Zöld lámpa esetén Továbbhaladok

Más esetben: az elsőbbségi szabályoknak megfelően járok el ...

Elágazás vége