Szabó Mária

Programozás – Strukturált programozás a gyakorlatban

NSZFI
NEMZETI SZAKKÉPZÉSI ÉS FELNŐTTKÉPZÉSI INTÉZET

A követelménymodul megnevezése:
Informatikai ismeretek

A követelménymodul száma: 1155-06 A tartalomelem azonosító száma és célcsoportja: SzT-018-50

Új Magyarország FEJLESZTÉSI TERV

PROGRAMOZÁS-STRUKTÚRÁLT PROGRAMOZÁS

ESETFELVETÉS – MUNKAHELYZET

Ön egy oktatási feladatokat ellátó cégnél dolgozik informatikusként. Azt a feladatot kapta, hogy tervezzen egy olyan kisebb strukturált programot, ami alkalmasan bemutatja, a strukturált programozás legfontososabb jellemzőit.

SZAKMAI INFORMÁCIÓTARTALOM

TÖRTÉNETI ÁTTEKINTÉS

1. Első generációs programozási nyelvek

Az első programozási nyelv a gépi kód volt. Ennek a nyelvnek az utasításait a számítógép képes volt közvetlenül, minden átalakítás nélkül végrehajtani, értelmezni. A problémákat nagyon precízen kellett megfogalmazni, lépésekre bontani. Emellett a nyelv erősen gépfüggő volt, hiszen minden gépen más és más utasításokat használt, az adott problémát minden géptípus esetén másképpen kellett leírni, megfogalmazni, alkalmazkodva a számolóegység sajátosságaihoz. Hátrányai mellett néhány előnyt is meg kell említenünk:

- A leggyorsabb programot eredményezi, mivel nem tartalmaz felesleges utasításokat.
- Az egyes utasítások egységes (egy vagy kétcímes) formában vannak leírva.

Példa egy gép kódú utasításra:

0B0A 0C0E 0D00

- Az első négy helyen az utasítás,
- majd az első memóriacím,
- és ezt követi a második memóriacím.

A gépi kódú lehetőségekhez képest komoly előrelépést jelentett az *Assembly nyelvek* megjelenése, amelyek tulajdonképpen segédeszközt jelentettek a gépi kódú utasítások egyszerűbb megfogalmazásában. Az egyes gépi kódú utasításokhoz egy-egy mneomnikus kódot rendeltek hozzá, a tárcímeket pedig a memória kezdetéhez viszonyított relatív címekkel számították. Szintén újdonságnak számított, hogy az egyes memóriacímeket egy-egy szimbolikus névvel lehetett helyettesíteni.

Egy összeadás Assembly nyelven:

```
ADD X, Y
```

Az ADD szócska az összeadásra (addition) utal, az X és Y pedig memóriacímeket szimbolizálnak.

Tehát az Assembly egy olyan alacsony szintű programozási nyelv, amelynél a művelet és a tárcím szimbolikusan megadható.

Természetesen, ha egy Assembly programot szeretnénk futtatni, azt mindig meg kell előznie egy átalakítási, fordítási műveletnek, amikor az Assembly utasításokat gépi utasításokká alakítjuk át. A fordítóprogramok „**compiler**" típusúak voltak. Az Assembly legtöbb utasítása egyetlen gépi utasítást reprezentált, azaz az Assembly 1:1 tipusú nyelv. Volt néhány utasítás, amely több gépi kódú utasítást jelentett, ezeket makróknak nevezték.

Az Assembly nyelv a gépi kódú programozás után igen nagy előrelépést jelentett, általa sokkal kényelmesebbé vált a programfejlesztési munka. Napjainkra az alacsony szintű programozási nyelvek háttérbe szorultak. Az Assembly-t ma már csak olyan feladatok esetén használják, ahol hardver szintű vezérlést kell megvalósítani.

2. Második generációs programozási nyelvek

A számítógépek alkalmazási területének bővülése szükségessé tette a programok fejlesztési idejének csökkentését, azaz felmerült az igény, hogy a programokat minél gyorsabban írják meg a programozók. A gépi kód és az assembly nehézkessége, géphez igazodása miatt nem volt erre alkalmas.

A 60-as évek elején jelentek meg az első magas szintű programozási nyelvek. *Az új nyelvek struktúrája már nem a számítógép sajátosságaihoz, hanem a problémához igazodott,* a programozóknak itt már nem kellett a gépi végrehajtással foglalkoznia.

A magas szintű programozási nyelvek 1:N típusú nyelvek, azaz egyetlen magas szintű programnyelvi utasítás több gépi kódú utasítást reprezentál. Természetesen a magas szintű nyelven írt programokat is le kell fordítani – az assembly-hez hasonlóan – de komoly előny, hogy ezeknél a nyelveknél már sokkal kisebb a hibalehetőség, a forrásprogram sokkal áttekinthetőbb.

Az első változatok utasításkészlete még nem tette lehetővé a struktúrált programozást, gyakran kellett alkalmazni az ugró utasításokat (GOTO).

3. Harmadik generációs programozási nyelvek

A második generáció programozási nyelvei már lényegesen függetlenebbek voltak a gépi logikától, mint az első generációs nyelvek, az elkészített programokat többféle gépen is használni lehetett. Azonban a nyelvek többsége egy-egy problémacsoport megoldására specializálódott, ami a programok fejlesztése, javítása során igen komoly akadályokat jelentett.

A 60-as évek derekán kíséreltek meg először olyan programozási nyelvet fejleszteni, amely rendelkezik az előző nyelvek jó tulajdonságaival, emellett többféle problémacsoport megoldására alkalmas, univerzális. Az új korszakot a procedurális programozási szemlélet és az adatok struktúrájának hangsúlyozása jellemezte.

A nyelveket két csoportra oszthatjuk:

- utasítás-orientált (BASIC, PASCAL…)
- kifejezés-orientált (C…)

Az általános célú programozási nyelvek családjába tartozó nyelvek széleskörűen alkalmazhatók a gazdasági számításoktól a rendszerprogramozási fejlesztésekig, széles a lehetőségek skálája.

A programok blokk-szerkezetű felépítése már a strukturáltság koncepcióját vetíti előre. A program tartalmaz egy főprogram részt, amely külön blokkokban több alprogramot foglalhat magába, az alprogramok egymásba ágyazhatók. A fordító a deklarációkat utólag helyezi el a program elején.

4. Negyedik generációs programozási nyelvek

Az utóbbi években kezdett széleskörűen elterjedni egy új szemléletű programozási technika, az objektum-orientált programozás, amely az elődeinél egyszerűbbé, hatékonyabbá, biztonságosabbá teszi a programkészítési munkát.

A XX. század második felében a számítástechnika forradalmi változáson ment keresztül. A Neumann elvek megjelenése és az első memóriában tárolt gépi nyelvű program lefuttatása után alig ötven évvel már információs rendszerek, központilag vezérelt számítógép hálózatok programozása, döntéshozást támogató rendszerek fejlesztése a célunk. Nyilvánvaló, hogy ilyen jellegű munkák elvégzésére a hagyományos programozási nyelvek által biztosított eszközök nem megfelelőek. A hardver eszközök rohamos fejlődésére támaszkodva kezdett kialakulni a programozás során egy új szemlélet, mely az eddigi alapelvekkel szöges ellentétben állt.

Az objektum-orientált programozási nyelvek absztrakt adattípusokkal és a hozzájuk tartozó előre definiált algoritmusokkal segítik a programozó munkáját. Az objektum-orientált nyelvek az adatokat és az őket kezelő eljárásokat, függvényeket egységbe foglaltan úgynevezett objektumként kezelik.

Az objektum-orientált programozás lényege, hogy a programozás során felhasznált elemeket a nyelv előre definiálja, és különböző tulajdonságokkal ruházza fel. Amikor a programozó használni szeretné az egyik elemet, csak a tulajdonságait kell egyéni ízlése-illetve a feladatnak megfelelő elvárások szerint beállítania.

Ez a programozási technika sokkal közelebb áll az emberi gondolkodáshoz, mint az eddigi programozási módszerek.

Meg kell említenünk a nyelvek nyilvánvaló előnyei mellett néhány rossz tulajdonságot is. A program hatékonyságának növekedésével egyenes arányban nő a program hardverigénye – elsősorban memória – igénye is, a lefordított program méretéről nem is beszélve.

5. Ötödik generációs programozási nyelvek

Jellemzőjük: természetes nyelvek, az eljárás orientált programozási nyelvek helyett ezek problémaorientált nyelvek.

A strukturált programozás alaptétele: Minden algoritmusnak létezik strukturált programja.

A STRUKTURÁLT PROGRAMOZÁS ÉPÍTŐKOCKÁI

Egy programot akkor nevezünk strukturáltnak, ha nem tartalmaz feltétel nélküli (GOTO) ugrást, és algoritmusa teljes mértékben megvalósítható véges mélységig „egymásba skatulyázott", egyszerű és összetett program struktúrákkal (szerkezetekkel).

Böhm és Jacopini tétele

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

A csak szekvenciákból, szelekciókból és iterációkból építkező programot strukturált programnak nevezzük. A strukturált programozásban a ciklusból való kiugrás fogalma ismeretlen. Ebből következik, hogy a program minden szekvenciájának – és így az egész programnak is – egyetlen belépési és egyetlen kilépési pontja van, ennélfogva a program lényegesen áttekinthetőbb.

Egy bonyolultabb algoritmust nem lehet fejben megtervezni, ahhoz eszközök kellenek. Olyan eszközre van szükség, mely általánosan elfogadott, és mások is ismerik, használják. Az algoritmusok áttekinthető formában való leírására számtalan eszköz létezik. Ilyen a mondatszerű leírás, a folyamatábra, vagy a struktogram. Ezeket részletesen megismerheti a "Programtervezés" tartalomelemből.

Programnyelvtől függetlenül minden program lebontható három alapvető műveletre:

Szekvencia: azt jelenti, hogy a kiadott parancsok egymás után következnek be. Mindennapi életből példa:

- Felkelek,
- Megmosakodom,
- Elindulok a munkába stb.

Iteráció: azaz ismétlődés. Mindennapi életből példa:

- Hétfőtől péntekig csináld meg a fenti műveleteket
- Addig vezetheted az autót, amíg van benne benzin

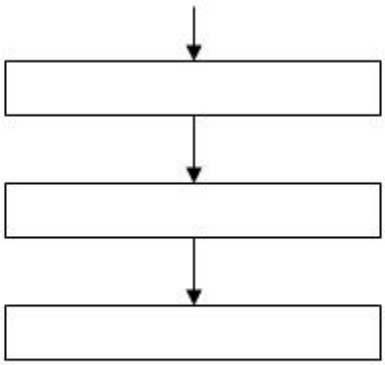
Döntés: Ha ... akkor ... egyébként. Mindennapi életből példa:

- Ha ma csütörtök van, akkor elmegyek edzésre (egyébként nem).

1. Összetett utasítás – szekvencia

A szekvencia egyszerűen az utasítások egymás után írása, az utasítások egy listája.

- Minden szekvencia elemnek egy belépési és egy kilépési pontja lehet. (tiszta szerkezet, részekre tagolható, egyik rész kimenete a másik rész bemenete)
- Minden szekvencia belülről tetszőlegesen strukturálható.



1. ábra. Szekvencia

2. Elágazás

Előfordulhat, hogy egy feladat nem egymás után megoldandó részfeladatokra, hanem alternatív módon megoldandó részfeladatokra tudunk csak felbontani. Ilyenkor elágazásra lesz szükségünk.

Felnőtt emberek esetén az elhízás mértékét az ún. testtömegindex segítségével szokták meghatározni. Ha az ember magassága **m** méter, tömege pedig **kg** kilogramm, akkor **bmi** testtömegindexét a **bmi** = **kg** / **m**^{2} összefüggésből számolhatjuk. Ha ez a szám 25 fölött van, akkor az illető felnőtt túlsúlyos, míg 30 fölött elhízott.

Foglaljuk össze a fenti eljárás algoritmusát!

Be: kg, m

bmi = kg/(m*m)

Ha bmi 25-nél kisebb

akkor Ki: Ön nincs elhízva.

Ha bmi 25 és 30 közé esik

akkor Ki: Ön túlsúlyos.

Ha bmi 30-nál több

akkor Ki: Ön elhízott.

Tehát az elágazás olyan vezérlési szerkezet, amely az utasítások egy adott csoportját attól függően hajtja végre, hogy egy adott logikai feltétel teljesül-e.

Az elágazásoknak alapvetően két fajtáját különböztetjük meg:

- kétirányú elágazás (if, if…else),
- többirányú elágazás (switch…case).

Kétirányú elágazás

Az elágazás legegyszerűbb változatát a következő algoritmus adja meg:

Ha *logikai feltétel* **akkor**

utasítások

Elágazás vége

Az utasítások rész akkor és csak akkor hajtódik végre, ha a logikai feltétel teljesül. A fenti szerkezet egy egységet alkot, ennek egy része csupán az utasítások blokk. Az utasítások blokk beljebb kezdésével ez külön is hangsúlyozható, és jobban áttekinthető maga a programkód is.

Az elágazás másik változatát a következő algoritmus adja meg:

Ha *logikai feltétel* **akkor**

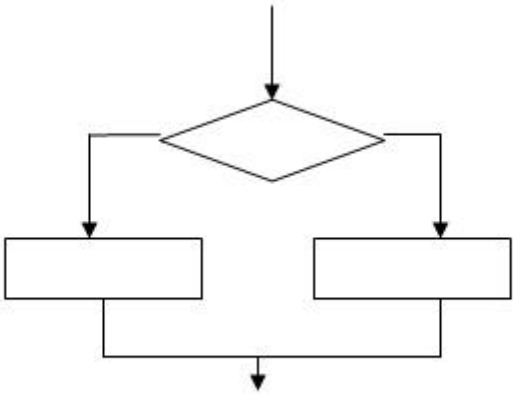
utasítások

Különben

utasítások

Ha vége

Az utasítások rész akkor és csak akkor hajtódik végre, ha a logikai feltétel teljesül, ellenkező esetben a "különben" ág hajtódik végre.



2. ábra. Kétirányú elágazás

Egymásba ágyazott elágazások

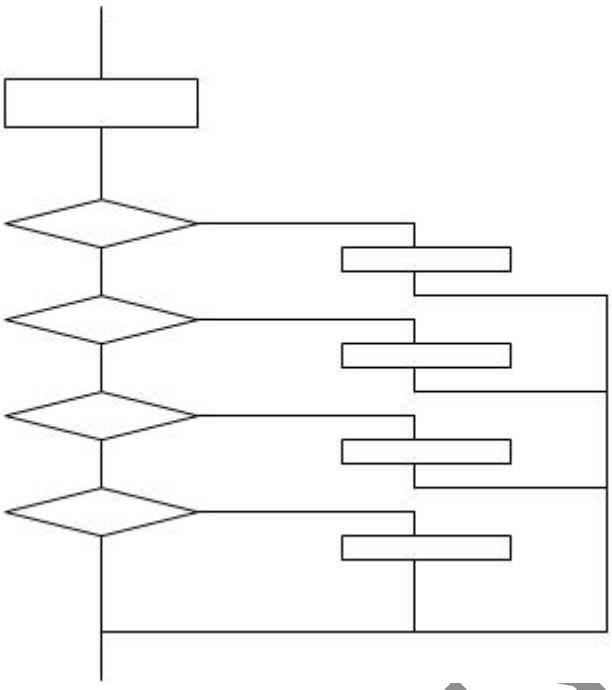
Egy elágazás utasításblokkjában szükségünk lehet újabb elágazásokra, ekkor nem kell mást tennünk, mint egy másik feltétellel megfogalmazott "Ha" utasítást elhelyezni az utasítások blokkjában.

Többirányú elágazás

A többirányú elágazások esetén egy kifejezés értéke alapján a programunk több irányban is folytatódhat. Ebben az elágazás típusban többszörös kétirányú elágazás egyszerre megvalósítható.

A szerkezet bemutatására egy egyszerű példát választunk. Gyakori feladat, hogy egy tulajdonság különféle értékeinek előfordulási számát kell meghatároznunk. A feladat, amelyet kiválasztottunk a következő:

Határozzuk meg, hogy egy adott karakter típusú változó értékei közül hányszor fordulnak elő az egyes magánhangzók! Ha nem magánhangzó karakter került a változóba, akkor annak számát is határozzuk meg! A feladat természetesen megoldható kétirányú elágazások egymásba ágyazásával, de lényegesen egyszerűbb a többirányú elágazás alkalmazása.



3. ábra. Többirányú elágazás

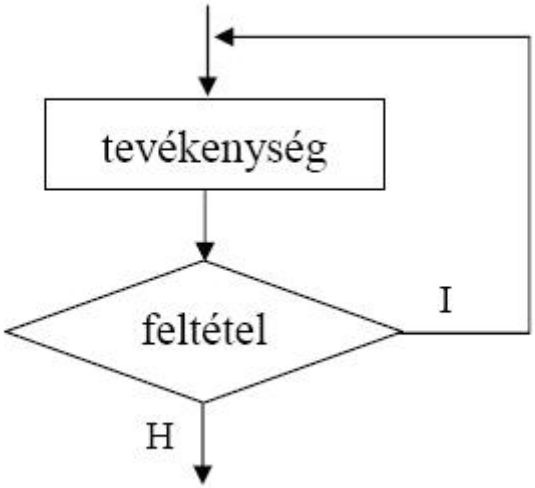
3. Ciklusok (iterációk)

A ciklusok olyan programrészletek, amelyek többször egymás után végrehajtódnak, és az, hogy hányszor, egy bizonyos feltételtől függ.

Típusai

- Hátultesztelős ciklus

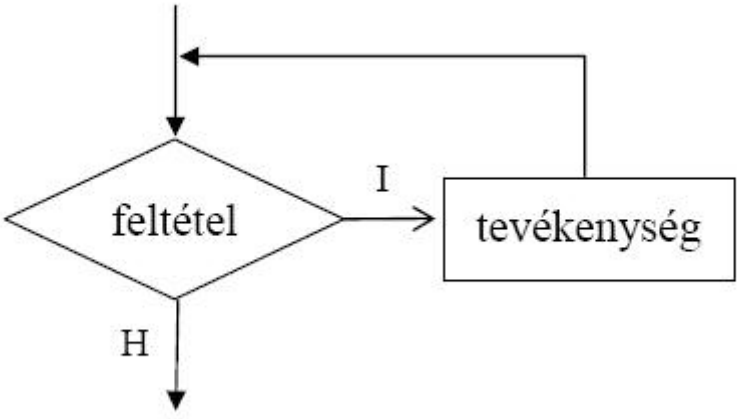
Működése: egyszer mindenképpen végrehajtódik az utasítás, majd ha a feltétel igaz, a ciklus végrehajtása folytatódik, ellenkező esetben, pedig leáll. Vannak feladatok, amiket nélküle csak körülményesen lehet megoldani. Alkalmazzuk például adatok ellenőrzéssel történő beolvasása esetén.



4. ábra. Hátultesztelős ciklus

Elöltesztelős ciklus

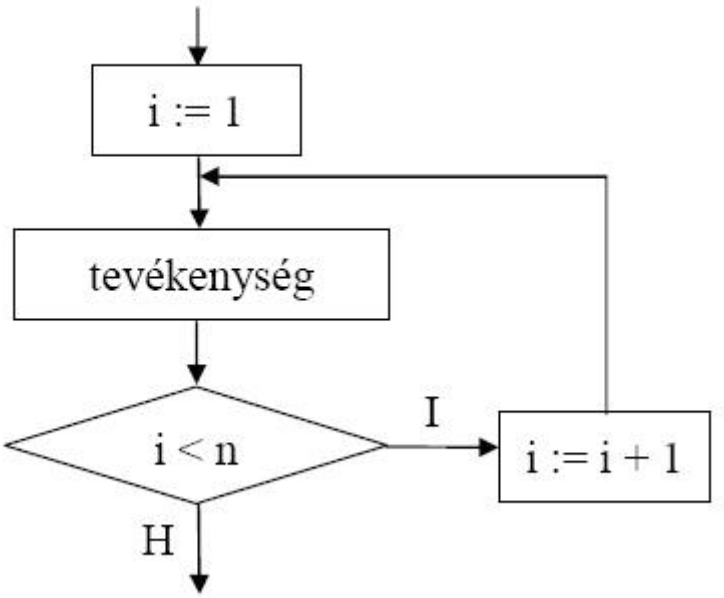
Az elöltesztelős ciklus tehát először megvizsgálja, hogy a feltétel fennáll-e. Ha igen, akkor lefuttatja a ciklusmagot, és az újból kezdődik. Ha nem, akkor a program a ciklus utáni ponton folytatódik tovább, azaz a ciklusmag kimarad. Lehetséges tehát, hogy az elöltesztelős ciklus egyszer sem fog lefutni. Az elöltesztelő ciklus alkalmazásának tipikus példája az adatállományok fájlból történő beolvasása. Előfordulhat ugyanis, hogy az állomány üres, és ilyenkor nincs mit beolvasni. Hasonló a helyzet a könyvtárak listázásakor is, hiszen a könyvtárban nem biztos, hogy van állomány.



5. ábra. Elöltesztelős ciklus

Növekményes ciklus

Az ismétlések száma előre ismert. Ebben az esetben a ciklus az előre megadott értékig fut. A megadott érték csak egész szám lehet, tört szám esetén hiba lép fel. A megadott szám minden esetben csak eggyel nőhet illetve csökkenhet, ez a feltételtől függ. Megadhatjuk a kezdő illetve a végértéket is. A programnyelvek többségében a növekményes ciklus a "**for**" kulcsszóval érhető el. A **for** ciklusban mindenképpen szükség van egy változóra, aminek az értéke vagy nő vagy csökken.



6. ábra. Növekményes ciklus

Összegezve:

A ciklus fajtája	**A futások száma**	**Legkevesebb hányszor fut le?**
Elöltesztelő	előre nem ismert lehet	lehet hogy egyszer sem
Hátultesztelő	előre nem ismert	legalább egyszer
Számlálós	előre ismert lehet	lehet hogy egyszer sem

4. Adattípusok

Programozás során a programok bizonyos adatokkal végeznek különböző műveleteket. Hogy mely adatokkal mit lehet kezdeni az az adat típusától függ. A különböző programnyelvek más-más adattípusokat definiálnak, melyek közül a legtöbb ismeri a számokat, (természetes és valós számok) karaktereket, logikai változókat.

Egy adattípus definiálása annyit jelent, hogy megadjuk hogy mi az értékkészlet (értelmezési tartomány) és milyen műveleteket végezhetünk velük, mely művelet mit jelent.

Az adattípusok segítségével definiálhatunk:

- **változókat** (program futása során változik az értéke) és
- **konstansokat** (program futása során az értéke változatlan, állandó.)

A változók, konstansok definiálása: adattípus változónév=kezdőérték alakban történik.

ADATTÍPUSOK JELLEMZŐI

1. Integer (egész szám)

Méret: 1 vagy 2 bájton ábrázoljuk (azaz 8 vagy 2*8biten) amiből az első bit előjelbit. Ha az első bit 1 akkor a szám negatív, különben pozitív (vagy nulla).

Intervallum:

1 bájt esetén: -128 és 127 közötti számok lehetnek,

2bájt esetén: -32768 és 32767 közötti számok lehetnek.

Műveletek: összeadás (+), kivonás (-), szorzás (*), egész osztás =maradékos osztás (DIV), maradékképzés =egész osztás utáni maradék(MOD), negálás (-1 -el való szorzás), hatványozás (^).

Relációs műveletek: kisebb (<), nagyobb (>), egyenlő (=), kisebb vagy egyenlő(<=) , nagyobb vagy egyenlő (>=), nem egyenlő (< >)

Negatív egész szám kettes komplemens alakjának meghatározása:

- felírjuk a szám abszolút értékét és átváltjuk kettes számrendszerbe
- a számot bitenként invertáljuk.
- majd hozzáadunk 1-et.

Természetesen a pozitív egész számok ábrázolása esetén egyszerűen csak átírjuk a számot kettes számrendszerbe.

Lássunk egy példát:

Ábrázoljuk a **-1**-t kettes komplemens kódban:

- abszolút érték: 00000001
- bitenként invertál: 11111110
- plusz 1: 11111111

2. REAL, FLOAT, DOUBLE… (valós számok)

Célja: a számábrázolási tartomány növelése (ezzel a pontosság bizonyos mértékben csökken).

Alapelve: az ún. normál (normalizált) alak.

Például: 10-es számrendszerben minden szám felírható $m*10^k$ alakban, ahol

- m a mantissza abszolút értéke,
- k a karakterisztika k=0;…(egész).

Ehhez a tizedes vesszőt (pontot) addig kell tolnunk a megfelelő irányba, hogy 1 és 10 közé eső számot kapjunk (⇒ lebegő tizedes pont). Az eltolás irányától és mértékétől függ a karakterisztika (kitevő) előjele illetve értéke.

Lebegőpontos alak kialakítása:

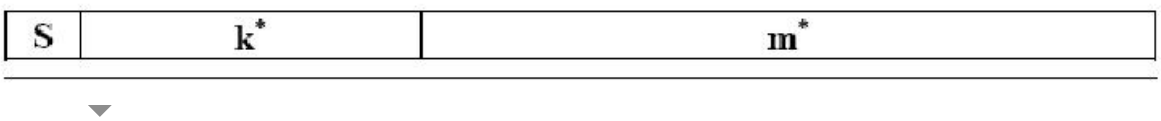
- a szám átalakítása kettes számrendszerbe
- normalizálás
  - nullára – az első értékes jegy 0 (pl.: 0,110011)
  - egyesre – az első értékes jegy 1 (pl.: 1,0011)
- karakterisztika kialakítása
  - a karakterisztika meghatározott értékéhez hozzáadjuk a táblázatból kiolvasható számot. Például 4 bájt esetén 127 (128).

az így kialakított alak: $m*2^k$

- A lebegőpontos alak felírása

Formátumok:

pontosság	hossz	Karakterisztika (k*)	Mantissza (m*)
egyszeres	4 byte (32 bit)	8 bit (eltolt 127)	23 bit
dupla	8 byte (64 bit)	11 bit (eltolt 1023)	52 bit
kiterjesztett	10 byte (80 bit)	15 bit (eltolt 16383)	64 bit



7. ábra. Lebegőpontos alak

S: a szám előjele

- 0: ha a szám pozitív
- 1: ha a szám negatív

Nézzünk egy példát a lebegőpontos alak kialakítására:

a=9.75 = 1001.11bin = 0.100111*2⁴, a karakterisztika: 4+127=131=10000011bin, tehát a 32 bitünk: 0_10000011_100111 00 0000 0000 0000 0000

-9.75 esetén: 1_10000011_100111 00 0000 0000 0000 0000 (a szóközök és az aláhúzások csak az olvashatóságot segítik)

Műveletek: összeadás(+), kivonás(-), szorzás(*), osztás(/), hatványozás(^), negálás(~)

Relációs műveletek: kisebb (<), nagyobb (>), egyenlő (=), kisebb vagy egyenlő(<=) , nagyobb vagy egyenlő (>=), nem egyenlő (< >)

3. BOOLEAN (logikai típus)

Méret: ábrázolásához elég lenne egyetlen bit. 0=hamis, 1=igaz. Gyakorlatilag mégis a legkisebb foglalási egység a bájt, ezért 1 logikai változó esetén is kénytelenek vagyunk minimum 1 bájtot lefoglalni.

Intervallum: igaz/hamis. Összesen két értéket vehet fel.

Műveletek: AND (és), OR (vagy), NOT(nem, negálás), XOR(kizáró vagy)

Relációs műveletek: kisebb (<), nagyobb (>), egyenlő (=), kisebb vagy egyenlő(<=) , nagyobb vagy egyenlő (>=), nem egyenlő (< >)

4. CHAR (karakter típus)

Ábrázolás (méret):

- ASCII kódtábla szerint 1 bájton 0-255 kódú jelek.
- UNICODE-16 szerint 2 bájt
- UNICODE-32 szerint 4 bájt
- UNICODE-64 szerint 8 bájt

Intervallum: ASCII 0-255 kódú jelek. Tartalmazzák az angol abc kis és nagybetűit, továbbá néhány ékezetes betűt. A tartalmazott betűk nem abc sorrendben követik egymást, mert külön tartományt képeznek a kis és a nagybetűk. Ezen kívül tartalmazzák az írásjeleket, és néhány speciális karaktert. (http://hu.wikipedia.org/wiki/ASCII címen meg lehet nézni az egyes karaktereket)

Műveletek: nincs karakter specifikus művelet.

Relációs műveletek: vannak, (kód alapján) de ez nem logikus csak egy résztartományon belül (pl. A<B<a).

ÖSSZETETT ADATTÍPUSOK

1. szöveges típus

- karakterek tárolására alkalmas típus
- a karakterek betűk, számok, vagy tetszőleges, a billentyűzetről (vagy máshonnan) bevihető jeleket jelentenek.
- egyes esetekben speciális jelentés tulajdonítható nekik
- egyes programnyelvek esetén a stringek hossza maximálisan 255 karakter
- Hasonlóan a többi változóhoz, a string-eket is deklarálni kell.
- A stringek kezelhetőek vektorként is. Hivatkozhatunk rájuk elemenként, mint a karakterekettartalmazó tömbökre

Műveletei:

- Sztringek hossza
- Sztringek összefűzése
- Részsztringek készítése
- Keresés sztringben
- Összehasonlítás
- Törlés
- Beszúrás…

2. tömb típus

- A tömb egy összetett homogén adatstruktúra
- Tetszőleges, de előre meghatározott számú eleme van
- Az elemek típusa azonos
- A tömb lehet egy vagy többdimenziós, a dimenzió száma nem korlátozott

A tömb elemeinek elérése

A tömb elemein végiglépkedhetünk egy egyszerű számláló ciklus alkalmazásával.

A tömböknek egy igen fontos típusa a dinamikus tömb. Mérete megváltoztatható futási időben. A dimenziók számára általában nincs korlát. Az egyes elemekre a [] operátorral hivatkozhatunk.

3. Rekord típus

Rugalmas szerkezet, tetszőleges számú, különböző típusú rész szerepelhet benne. Rekord egy elemére úgy hivatkozunk, hogy a rekord-változó neve után ponttal elválasztva írjuk az elem azonosítóját

ADATTÍPUSOK ALKALMAZÁSA:

Különböző adattípusokon más-más műveletek végezhetők el. Különösen kell ügyelni a számábrázolás mélységére. Rosszul megválasztott adattípus a program hibás működését eredményezheti.

TANULÁSIRÁNYÍTÓ

1. Állapítsuk meg egy billentyűzetről bekért számról, hogy prímszám-e! A prímszámok pozitív egészek, pontosan két osztójuk van, 1 és önmaga. Készítsen algoritmust és programot, az Ön által tanult programnyelven.

   Megoldási javaslat: ha 2 és a szám négyzetgyöke közt nem talál osztót, akkor a szám prím.

2. Állapítsuk meg két billentyűzetről bekért számról, hogy mi a legnagyobb közös osztójuk!

   Megoldási javaslat: készítsen léptető ciklust, ami 1-től a kisebbik számig vizsgálja az oszthatóságot. Ha közös osztót talál, az lesz a legnagyobb közös osztó.

3. Állapítsuk meg két billentyűzetről bekért számról, hogy relatív prímek-e! Akkor relatív prímek, ha a legnagyobb közös osztójuk az 1. Készítsük el a megoldás algoritmusát, illetve az Ön által tanult programnyelven készítsen programot.

Megoldási javaslat: az előző feladat alapján határozza meg a legnagyobb közös osztót. Ha az 1, akkor a számok relatív prímek, különben nem. A fogalmat pozitív egészek halmazán értelmezzük.

4. Állítsuk elő egy szám prímtényezős felbontását! Készítsük el a megoldás algoritmusát, illetve az Ön által tanult programnyelven készítsen programot.

Például 360=2*2*2*3*3*5!

Megoldási javaslat: 2-től indulva keressünk osztót. Ha találunk írassuk ki a képernyőre, és az osztást is végezzük el a számon. Az így megváltozott számmal ismételjük a műveletet mindaddig, amíg a szám 1 nem lesz.

5. Állapítsuk meg, hogy egy adott intervallumba eső számok közül melyik a legnagyobb prímszám! Az intervallum alsó és felső határának értékét kérjük be billentyűzetről! Próbáljunk keresni idő-hatékony megoldásokat! Készítsük el a megoldás algoritmusát, illetve az Ön által tanult programnyelven készítsen programot.

Megoldási javaslat: az előző feladatok megoldásait használjuk.

6. Írjon olyan programot az Ön által tanult programnyelven, amely egy ciklussal kiszámolja és kiírja az alábbi számtani sorozat első 20 elemének összegét! Készítsük el a megoldás algoritmusát is!

3,5,7,9,11,stb.!

Ellenőrizzük le az eredményt a számtani sorozat összegképlete segítségével!

Megoldási javaslat: a növekményes ciklus 3-mal kezdődik, 11-gyel végződik. A ciklusváltozó 2.

7. Írjunk olyan programot, amely kiszámolja és kiírja az alábbi változó növekményű sorozat első 20 elemének összegét! Készítsük el a megoldás algoritmusát is!

3,5,8,12,17,23,30,stb.

Megoldási javaslat: használjon elöltesztelős ciklust, amely nem csak az összeg előállítását végzi, hanem a ciklusváltozója is növekszik 1-gyel!

8. Írjunk olyan programot illetve algoritmust, amely bekéri egy tetszőleges számtani sorozat első elemét és a differenciát! Ezek után kiírja a képernyőre a számtani sorozat első 15 elemét, illetve ezek összegét és átlagát!

9. Írjunk olyan programot illetve algoritmust, amely bekéri egy tetszőleges mértani sorozat első elemét és a hányadosát! Ezek után kiírja a képernyőre a mértani sorozat első 20 elemét, illetve ezek összegét és átlagát! Ügyeljen a megfelelő adattípus kiválasztására!

10. Írjon olyan programot az Ön által tanult programnyelven, amely kiszámolja és kiírja 2^n első 20 elemének értékét! Készítsük el a megoldás algoritmusát is! N értékei: 1,2…

Megoldási javaslat: ha ismerjük (egy változó értékeként) pl. 2^3-t, akkor 2-vel való szorzással egyszerűen előállítható 2^4 értéke.

11. Írjon olyan programot az Ön által tanult programnyelven, amely egy ciklussal kiszámolja és kiírja az alábbi sorozat első 10 elemének összegét! Készítsük el a megoldás algoritmusát is! A sorozat: $a_n = a_n -5 + 2*n$. $a_1 = 3$.

Megoldási javaslat: használjon növekményes ciklust!

12. Írjon olyan programot az Ön által tanult programnyelven, amely kiszámolja az első N négyzetszám összegét! Billentyűzetről kérje be N értékét! Készítsük el a megoldás algoritmusát is!

13. Határozzuk meg egy [a,b] intervallum belsejébe eső négyzetszámokat, és azok összegét! Az a és b értékét kérjük be billentyűzetről!

Megoldási javaslat: ügyeljen arra, hogy a < b.

14. Írjon olyan programot az Ön által tanult programnyelven, amely egy ciklussal kiszámolja és kiírja a Fibonacci sorozat első 15 elemének összegét! Készítsük el a megoldás algoritmusát is!

A sorozatról tudjuk a következőket:

- $a_1 = 1$
- $a_2 = 1$
- $a_n = a_{n-1} + a_{n-2}$ ha $n>2$

15. Határozzuk meg két N dimenziós vektor skaláris szorzatát, ha adottak a két vektor koordinátái. Készítsük el a megoldás algoritmusát, illetve az Ön által tanult programnyelven készítsen programot.

Megoldási javaslat: a skaláris szorzatot meghatározhatjuk, ha az azonos indexű elemeket összeszorozzuk, majd ezeket a szorzatokat összeadjuk.

16. 5 km-enként megmértük a felszín tengerszint feletti magasságát. (összesen N mérést végeztünk) A méréseket szárazföld felett kezdtük és fejeztük be. Ott van tenger, ahol a mérés eredménye 0, másutt >0. Határozzuk meg, van-e ezen a tengerszakaszon sziget! Készítsük el a megoldás algoritmusát, illetve az Ön által tanult programnyelven készítsen programot.

Megoldási javaslat: a mérési eredményeket N dimenziós tömbben tároljuk, melynek elemei nemnegatív számok. Az első és utolsó szám pozitív. Sziget ott van, ahol a pozitív számokat nulla határolja.

17. Az előző feladatból mondjuk meg

- Hány szigeten jártunk?
- Adjuk meg azt a szigetet, ahol a legmagasabb csúcs van!

18. Határozzuk meg N (N>1) természetes számnál nem nagyobb számok közül a legnagyobb négyzetszámot!

19. Állítsunk elő 50 véletlen számot 1 és 500 között. Készítsük el a megoldás algoritmusát, illetve az Ön által tanult programnyelven készítsen programot.

- Válogassuk ki a 100-nál nagyobb páratlan számokat!
- Válogassuk ki a hárommal nem osztható páros számokat!
- Válogassuk ki a 200-nál nem nagyobb számok közül azokat, melyek számjegyeinek összege öttel osztható!
- Válogassuk ki a 100 és 300 közé eső, 3-as számjegyet tartalmazó számokat!
- Keressük meg a legnagyobb 3-mal nem osztható, de 7-el osztható számot!
- Keressük azt a legkisebb számot, mely tartalmaz 3-as számjegyet!

20. Készítsük el egy ötös lottósorsolás programját! Készítsük el a megoldás algoritmusát is. Ügyeljünk arra, hogy egy számot csak egyszer húzható ki!

Megoldási javaslat: két egymásba ágyazott elöltesztelős ciklust használjon. A külső ciklus állítja elő a számot, a belső ciklus pedig összehasonlítja a már kihúzott értékkel. Ha már előfordult a szám, újra generáljuk.

21. Határozzuk meg tmb[N,N] mátrix felső háromszög elemeinek összegét!

Megoldási javaslat:egy elem a felső háromszögmátrixban van, ha a sorindex kisebb mint az oszlopindex. Az első index a sorindex, a második az oszlopindex.

Oldja meg a feladatot az alsó háromszögmátrix elemeire is!

22. Adott a tmb[N,N] mátrix. A mátrix fődiagonálisába írjunk olyan értékeket, hogy a mátrix minden sorában az elemek összege 0 legyen!

Megoldási javaslat: fődiagonális esetén az oszlop- és sorindex azonos.

23. Egy osztály tesztlapokat töltött ki. Minden kérdésre 4 lehetséges válasz volt, és csak 1 jó. A válaszokat egy mátrixban tároljuk. Készítsünk programot, amely összehasonlítja a válaszokat a helyessel, majd megmondja, melyik tanuló hány jó választ adott. A helyes válaszokat tároljuk egy vektorban.

Megoldási javaslat: vegyen fel egy változót, amely értékét növelje 1-gyel, ha azonos indexű helyen azonos értéket talál.

24. Írj programot, mely kiírja két számokat tartalmazó tömb közös elemeit! Készítsük el a megoldás algoritmusát, illetve az Ön által tanult programnyelven készítsen programot.

25. Írja fel

(a) -87-t kettes komplemens alakban

(b) -103-t kettes komplemens alakban

26. Írja fel 4 bájton 12,125-t lebegőpontos alakban!

ÖNELLENŐRZŐ FELADATOK

1. feladat

Melyik típusú ciklusszervezés a következő algoritmus?

Ciklus ciklusváltozó = I kezdőértéktől N végértékig k lépésközzel

Ciklusmag utasításai

Ciklus vége

2. feladat

Melyik típusú ciklusszervezés a következő algoritmus?

Ciklus

Ciklusmag utasításai

mígnem Kilépési feltétel

Ciklus vége

3. feladat

Melyik típusú ciklus szervezés a következő algoritmus?

Ciklus amíg Lefutási feltétel

Ciklusmag utasításai

Ciklus vége

4. feladat

Mi a különbség a két szelekció között? Keressen példát alkalmazásukra!

(a)
```
Ha  Feltétel  akkor
        Utasítás(ok)
Elágazás vége
```

(b)
```
Ha Feltétel akkor
        Utasítás(ok)1
Egyébként
        Utasítás(ok)2
Elágazás vége
```

5. feladat

Keresse meg pl. az internet lehetőségeit kihasználva, hogy a strukturált programozás általános módszereinek kidolgozása kinek a nevéhez kötődik! Készítsen életrajzáról rövid esszét vagy prezentációt!

6. feladat

Igaz-e a következő állítás?

A feltétel nélküli ugrás nem szükséges elem a strukturált programozásban, gyakori használata áttekinthetetlenné teszi a programot.

7. feladat

Minek a definíciója a következő állítás?

- Valamely tevékenység sorozat ismételt végrehajtását jelenti.
- Az ismétlés feltételhez kötött.

8. feladat

Készítsen algoritmust, majd ehhez az Ön által tanult programozási nyelven készítsen programot, amely bekéri egy felnőtt testmagasságát (110–210 cm) és testsúlyát (30–150 kg), majd kiírja azokat. Csak a megadott határok közötti értékeket fogadhat el.

9. feladat

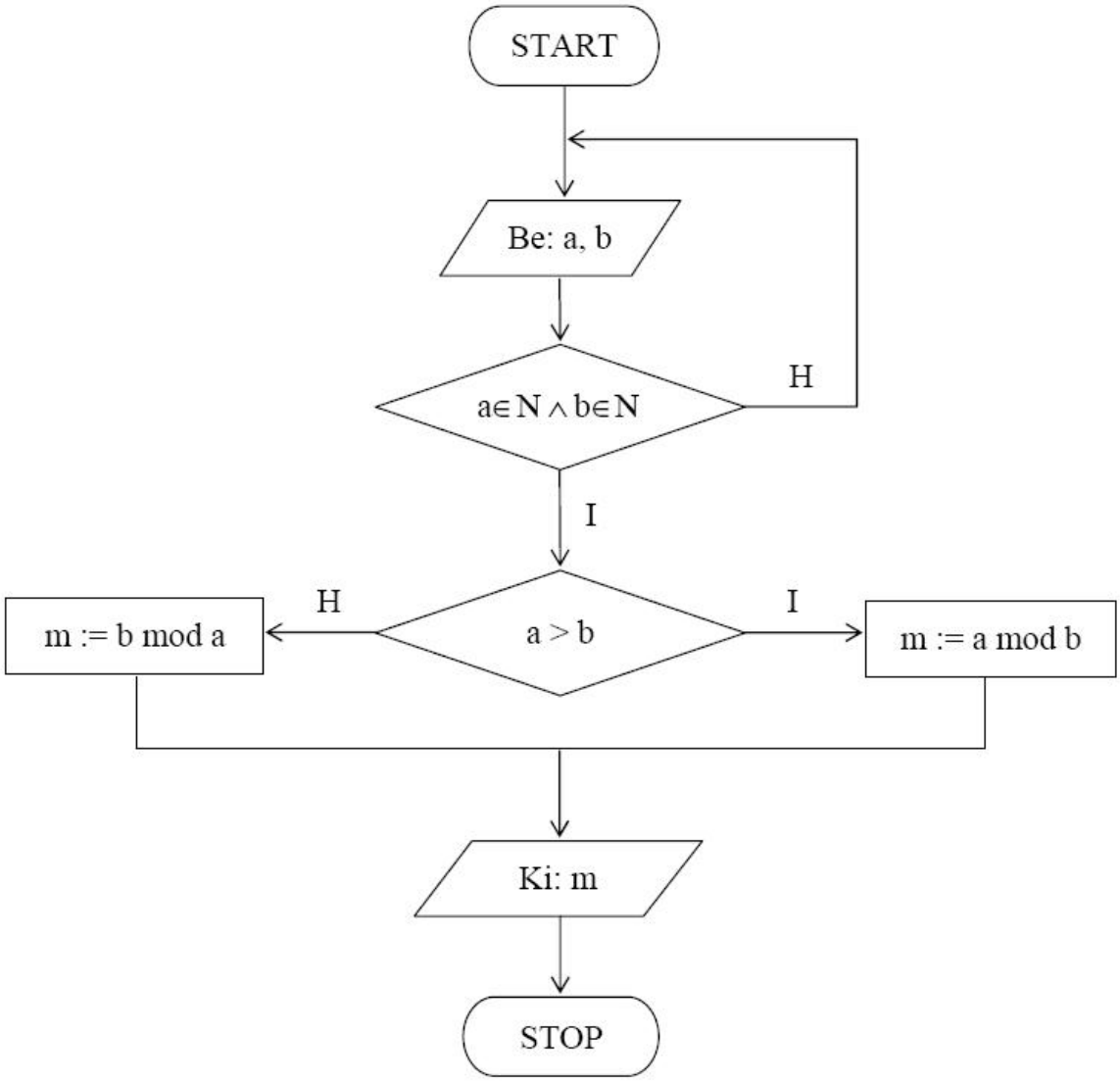
Készítsen algoritmust, majd ehhez az Ön által tanult programozási nyelven készítsen programot amely előállít 0 és 50 közötti véletlen-számokat addig, amíg a 25-ös szám generálása is megtörtént. Írassuk ki a kapott számokat.

10. feladat

Készítsen algoritmust, majd ehhez programot az Ön által tanult programozási nyelven, amely n darab * karaktert ír ki a képernyőre. N értékét a felhasználótól olvassa be!

11. feladat

Mit csinál a következő algoritmus?



8. ábra.

MEGOLDÁSOK

1. feladat

Növekményes (számláló).

2. feladat

Hátul tesztelős.

3. feladat

Elől tesztelős.

4. feladat

Az (a) feladat egyágú, a (b) pedig kétágú szelekció.

5. feladat

E. W. Dijkstra dolgozta ki.

6. feladat

Igaz.

7. feladat

Ciklus

8. feladat

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
```

9. feladat

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

10. feladat

```
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
```

IRODALOMJEGYZÉK

FELHASZNÁLT IRODALOM

Sipos Mariann: A visual C++ és az MFC; INNOVADIDACT Bt., 2004

Illés Zoltán: Programozás C# nyelven; Jedlik Oktatási Stúdió Budapest, 2008

Trey Nash: C#2008; Panem Kft, 2008

A(z) 1155-06 modul 018-as szakmai tankönyvi tartalomeleme felhasználható az alábbi szakképesítésekhez:

A szakképesítés OKJ azonosító száma:	A szakképesítés megnevezése
54 481 01 1000 00 00	CAD-CAM informatikus
54 481 04 0010 54 01	Gazdasági informatikus
54 481 04 0010 54 02	Infostruktúra menedzser
54 481 04 0010 54 03	Ipari informatikai technikus
54 481 04 0010 54 04	Műszaki informatikus
54 481 04 0010 54 05	Távközlési informatikus
54 481 04 0010 54 06	Telekommunikációs informatikus
54 481 04 0010 54 07	Térinformatikus

A szakmai tankönyvi tartalomelem feldolgozásához ajánlott óraszám:
20 óra

A kiadvány az Új Magyarország Fejlesztési Terv
TÁMOP 2.2.1 08/1-2008-0002 „A képzés minőségének és tartalmának fejlesztése” keretében készült.
A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Kiadja a Nemzeti Szakképzési és Felnőttképzési Intézet
1085 Budapest, Baross u. 52.
Telefon: (1) 210-1065, Fax: (1) 210-1063

Felelős kiadó:
Nagy László főigazgató