Programozás TURBO PASCAL nyelven

BBS-E Bt. 1999, 2001.

Szerzők: 1. fejezet: Sági Gábor
2-11 fejezet: Molnár Csaba

Szerkesztő: Bártfai Barnabás

ISSN 1418-8791
ISBN 963 03 7152 9

Kiadja a BBS-E Betéti Társaság
1630 Budapest, Pf. 21.
Felelős kiadó: a BBS-E Betéti Társaság ügyvezetője

Nyomdai munkák: Bíró Family Kft.
Felelős vezető: Bíró Endre

Tartalomjegyzék

Előszó

A könyv két jól elkülöníthető részből áll. Az első fejezet a programozás elméletével foglalkozik, így elsősorban azoknak szól, akik még nem próbálkoztak programírással. Könyvünk második, gyakorlati részében a Turbo Pascal programozási nyelvet ismerheti meg a kedves olvasó.

Ha még nem programozott, de szeretne, és nincs lehetősége tanórákon vagy tanfolyamokon elsajátítani egy programozási nyelvet, akkor ez a könyv e célra kiválóan alkalmas. Ez a kezdők könyve. A szerző igyekezett úgy felépíteni, hogy bárki, aki kedvet érez magában programok írására, e könyvecske segítségével megtanulhassa az alapokat, és talán még azon túl is eljuthasson egy kicsit. Szeretnénk, ha mindenki megszeretné a programozást, hiszen az egy nagyon jó és rendkívül kreatív játék! De ha már belekóstolt a programozásba Turbo Pascal nyelven, de elege van az olyan könyvekből, amelyek inkább egy szótárra hasonlítanak, vagyis ABC rendben felsorolják a létező összes utasítást és eljárást, akkor erre a könyvre van szüksége. (Bár természetesen az ilyen típusú könyvekre is szükség van!) Ha már programozott, de nagyrészt a saját erődből jöttél rá a dolgokra, ezzel a könyvvel rendszerezheti a tudását.

Ha már ismeri a Turbo Pascal nyelvet, és tudása elmélyítésére vágyik, arra, hogy új dolgokat tanulj meg a nyelvről, akkor valószínűleg ez nem az Ön által keresett könyv. Persze sohasem lehet tudni...

De ha olyan könyvet keres, amelyet oktató és diák egyaránt haszonnal forgathat, akkor ez a könyv az.

Összefoglalva: ez a könyv a kezdetektől indulva vezeti be olvasóját a programozásba, illetve a Turbo Pascal programozási nyelvbe. Minden új ismeretet példákon keresztül magyaráz, és sok feladatot tartalmaz, amelyekkel gyakorolhatók ezek az újdonságok. Egyaránt alkalmas lehet oktatói segédletnek, tankönyvnek, illetve az önálló tanulást is támogatja.

Sokszor feltették nekem azt a kérdést, hogy manapság, amikor szinte minden problémára létezik szoftver, kell-e programozási ismeret azok-

nak, akik nem profi programozók. A válaszom az, hogy nem kell azoknak, akik a számítógépet csak munkaeszköznek tekintik, de akit már egy kicsit is érdekel a masinája, annak melegen ajánlható. A számítógépes ismeretekhez, a számítógép kultúrájához mindenképpen hozzátartozik a programozás, és segít abban, hogy egy kicsit megismerhessük és megérthessük gépünk „lelkivilágát". Azonfelül a programok írása megtanít bennünket sok olyasmire – logikus gondolkodásra, pontos tervezésre, a kis részletek el nem hanyagolására, stb. –, amit az élet egyéb területein is alkalmazhatunk. Főleg gyerekeknél nagyon hasznosak ezek a képességek; nem is kell magyarázni, hogy mekkora hasznát vehetik ezeknek az iskolában, például a matematika (vagy bármely más) órán. Jobb is, ha a figyelmüket ezzel kötjük le: ha már „számítógépezik" egész nap, mindenképpen jobb, ha programokat ír, mintha harci játékokkal játszana. A programozás van olyan hasznos és kreatív, mint bármelyik építőjáték (egy kicsit az is: építőköveink az egyes utasítások, ezekből kell valami olyat alkotni, ami „megáll a saját lábán"). És végezetül még egy érv: a legkedveltebb szoftverek, a szövegszerkesztők és táblázatkezelők világába is belopóztak mára a programozás alapfogalmai. A szövegszerkesztők körlevél szolgáltatása, a Word-ben a mezők kezelése, az Excelben néhány függvény (és persze még sokáig sorolhatnám) megtanulása sokkal egyszerűbbé válhat a programozás alapfogalmainak ismeretével. Adatbázis-kezelő programok használatánál pedig szinte nélkülözhetetlenek a programozási ismeretek (sokszor nagyon is magas szinten!). Mondhatnánk erre azt persze, hogy akkor halljuk azokat az alapfogalmakat, aztán fejezzük is be! Lehetne így is. Azonban sokkal jobban megtanulható mindez, ha a gyakorlatban is kipróbáljuk egy programozási nyelv segítségével.

A könyv használatáról annyit, hogy ha önállóan halad vele, akkor szigorúan sorban kell haladni, semmit sem kihagyva. Az egyes fejezetekben példákon keresztül igyekszünk bemutatni az új ismereteket. A példaprogramokat célszerű begépelni, és a javasolt módokon kipróbálni. Minden fejezet végén feladatok találhatók, amelyekkel gyakorolható az adott témakör. Minden feladatnak több megoldása lehetséges, bár ezek eltérhetnek egymástól hosszban, hatékonyságban, bonyolultságban ...stb.

Az első programozáselméleti fejezet után megismerkedhet a programozás, és ezzel együtt a Turbo Pascal alapjaival is, vagyis bemutatjuk azokat az alapfogalmakat és „építőköveket" amelyekből egy program összerakható. Ezen ismeretek birtokában már komoly programok is

írhatók, de a programkészítés csak „ösztönösen" megy. A továbblépés a strukturált programozás lehet, ennek alapelemeivel, a programkészítés lépéseivel, illetve az algoritmusok készítésével foglalkozik a következő fejezet.

A további két nagyobb témakörben az alapismereteken túlmenő, de hasznos nyelvi elemeket igyekszünk bemutatni, amellyel a tudásunk biztosabbá és gazdagabbá válhat. Ezek részben a Turbo Pascal lehetőségei, részben olyan alkalmazási példák, amelyek hasznos építőkövekként beépülhetnek majd saját programjainkba is.

Az utolsó fejezetben egy összetettebb program elkészítését mutatjuk be lépésről lépésre, ezáltal is a programozási gyakorlatot szeretnénk erősíteni.

A könyv nem törekszik sem a Turbo Pascal, sem a programozás teljes ismeretkörének bemutatására. Nem szerepel tehát e lapokon a nyelv összes eljárása és függvénye, és hiába keresi a kedves olvasó benne a magas szintű programozási ismereteket. Nem része továbbá a könyvnek a Turbo Pascal integrált fejlesztő környezetének ismertetése sem.

Még egy megjegyzés: a könyv használatához megkívántatik némi gépkezelői alapismeret (programok indítása, kész anyagok mentése, betöltése, ...stb.). Ez nem szerepel a könyvben, de a könyv végén található néhány olyan könyv ajánlása, amely hasznos segítséget nyújthat ezek megismeréséhez is.

1. A programozás elmélete

1.1. Bevezetés

A számítástechnikában, hasonlóan az élet más területeihez, folyamatosan valamilyen problémát kell megoldanunk. A probléma megoldása során az általunk ismerttől - megfelelően összekapcsolt - műveletekkel jutunk el az ismeretlenig. Ezt a megoldási menetet nevezzük algoritmusnak.

Az algoritmus tehát nem más, mint műveletek megfelelő módon való összekapcsolása. Az ismert adatokat INPUT-nak, az ismeretlen adatok OUTPUT-nak nevezzük.

Gyakran előfordul, hogy egy feladat első ránézésre egyszerűnek tűnik, azonban egy kis gondolkodás után rájöhetünk, hogy a feladat nem is olyan egyszerű. Érdemes elgondolkodnunk azon, milyen bonyolult algoritmust kell végrehajtanunk ahhoz, hogy telefonáljunk. Első pillanatokban nem is gondolunk arra, hogy mit kell tennünk, ha foglalt a vonal vagy ha nem veszik fel a telefont stb. A mindennapok, a gyakran ismétlődő feladatmegoldás során ezek a felmerülő problémák számunkra már nem okoznak gondot, hiszen nagyon sokszor megoldottuk már, automatikusan cselekszünk ilyen helyzetekben. Másik példa, talán mindenki emlékszik arra, mikor először beült egy kocsiba vezetőként. Először komoly gondot okozhatott már a pedálok használata is, de később már a sebességváltás is könnyedén ment és mindez néhány kilométer vezetés után.

A számítástechnikában a programozó készíti el a probléma megoldásának eszközét. Ez az eszköz a program. A program egy adott feladat megoldására készített – számítógép által megértett – algoritmus alapján felépített utasítások összessége. Az adott programot fel kell készíteni arra, hogy az összes lehetőséget figyelembe vegye. Ne következzen be

az az eset – az előző példa alapján - , hogy a program ne tudjon mit csinálni akkor, ha netalántán az üzenetrögzítő kapcsol be. A programban nagyon részletesen meg kell adnunk, hogy a program mit is csináljon.

1.2. A program készítésének menete

Egy program elkészítése általában nem egy lépésben történik. Nézzük végig mi történik akkor, ha a főnök azt mondja a titkárnőjének, hogy hívja fel telefonon az egyik beosztottját. A titkárnő először megérti a problémát , majd eltervezi és végül végrehajtja a megoldási lépéssorozatot. Az algoritmus a következő fő lépésekből áll:

1. kinyitja a telefonregisztert
2. felemeli a telefonkagylót
3. tárcsáz, amíg a beosztott fel nem veszi a telefont
4. ha felvette a beosztott a telefont, bekapcsolja a főnökhöz
5. a telefonregisztert bezárja

1.2.1. A program elkészítésének lépései

Ahhoz, hogy egy programot megfelelő minőségben el tudjunk készíteni mindenképpen be kell tartani a program elkészítésének lépéseit, melyek a következők:

- elemzés
- tervezés
- végrehajtás
- ellenőrzés

1.2.2. Az algoritmus és szerkezete

A feladat megoldása során nagyon fontos a program algoritmusának elkészítése. A kérdés csak az, hogy ezt milyen részletességgel tegyük meg. Minden feladat további részfeladatokra bontható. Érdemes elgondolkodni azon, kell-e olyan problémákkal foglakozni, mint:

Hol van a telefonregiszter?

Melyik oldalon van a keresett személy?

Elérhető-e telefonon?

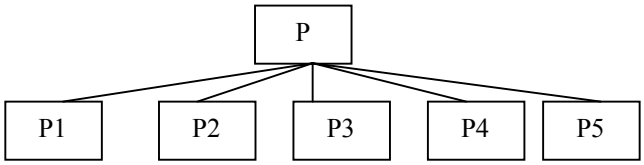
1.2.2.1. Szekvencia

Nézzük meg az előző feladat 5 lépését (a főnök a beosztottal szeretne beszélni):

P1: a titkárnő kinyitja a telefonregisztert
P2: a titkárnő felemeli a telefonkagylót
P3: tárcsáz, amíg a valaki fel nem veszi a telefont
P4: ha felvette valaki a telefont, kéri a beosztottat, és bekapcsol a főnökhöz
P5: a telefonregisztert bezárja

A probléma megoldása a részfeladatok egymás utáni megoldása.

A probléma megoldásának hierarchikus ábrája:



Ebben a feladatban az egyes részfeladatokat egymás után kell végrehajtani. A P1..P5 részprogramok szekvenciát (sorrendet) alkotnak.

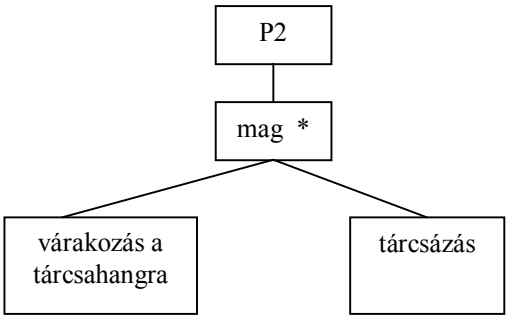
1.2.2.2. Iteráció

Gyakran előfordul, hogy bizonyos cselekménysorozatot többször kell végrehajtanunk.

Erre a példa a P3-as részfeladatot, ebben a feladatban egész addig kell tárcsáznunk, míg valaki fel nem veszi a telefont, ha senki sem veszi fel, később újra megpróbálhatjuk:

1. a kagyló felvétele után várjuk a tárcsahangot
2. ha megjött, tárcsázunk

Ezt a végrehajtási módot iterációnak nevezzük. Az iteráció magját az ismétlődő műveletek adják.

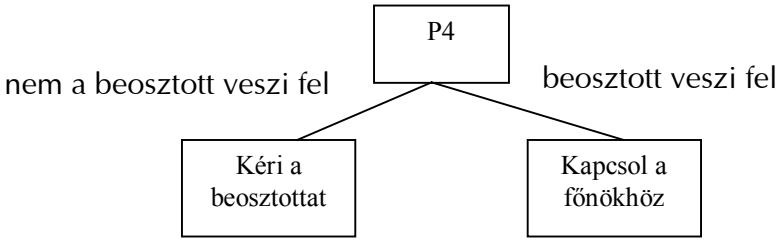


1.2.2.3. Szelekció

A mindennapi életben a döntés az egyik legfontosabb tevékenység. A döntés nem más, mint a dolgok közötti választás. A programozásban és az élet sok területén a választást szelekciónak nevezik.

Példaként nézzük meg a P4-es részfeladatot részletesen:
1. ha valaki felvette a telefont, meg kell kérdezni ki az,
2. ha a beosztott, kapcsol a főnökhöz
3. ha nem a beosztott, akkor kérni kell a beosztottat.

Az a folyamatot, mikor kiválasztjuk, hogy a beosztott vette fel a telefont vagy sem, döntésnek nevezzük.



Az eddig bemutatott ábrákat program-szerkezeti vagy hierarchia-ábrának nevezzük, mivel a program hierarchikus szerkezetét tükrözi.

1.2.3. Az algoritmussal szemben támasztott követelmények

Teljesség: minden információt tartalmaznia kell, ami a feladat megoldásához szükség.

Egyértelműség: az algoritmus leírása legyen egyértelmű, minden lépés csak egyféleképpen legyen értelmezhető.

Befejeződés: az algoritmusnak véges számú lépésben be kell fejeződni.

Az algoritmus minden egyes lépése egy műveletet valósít meg. Az emberi olvasásra szánt algoritmusok tartalmazhatnak meglehetősen összetett lépéseket is (pl.: lemegyek telefonálni), de a számítógépes algoritmusnak mindenképpen rendkívül részletesnek kell lennie.

A programozás során alapvetően 4 féle algoritmus lépést alkalmazunk:

számítás: egyszerű numerikus, logikai vagy karakterművelet

döntés: számok, karakterek összehasonlítása, és ennek eredménye alapján egy vagy több alternatív lépés kiválasztása

input: számításra adatok bevitele

output: a kiszámított eredmények kiadása

1.2.4. Folyamatábrák

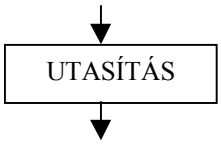
A hierarchia-ábra a probléma, illetve a megoldás szerkezetének ábrázolására alkalmas. Célszerű azonban egy másik ábrázolási módot alkalmazni, amely a folyamatot helyezi előtérbe és közérthető formájú, mégis félreérthetetlenül leírhatjuk vele az algoritmust. A végleges program-kód elkészítése előtt célszerű lehet egy korlátozott formájú magyar nyelvű leírást használni. Ezt a nyelvet pszeudonyelvnek nevezzük. A magyar nyelv sokszínűsége miatt ez a leírási mód nem biztos, hogy mindenki számára ugyanazt fogja jelenteni. Ezért az algoritmusok leírásához egy, a hétköznapi nyelvi formánál precízebb, ámde „emberi fogyasztásra alkalmas" formára is szükség van. Ezt valósíthatja meg az algoritmust kifejező, képi ábrázolás, a folyamatábra. Ez az ábrázolási módszer képi megjelenítési formája miatt áttekinthető és érthető.

Az algoritmus egy-egy lépésének leírásához a folyamatábrán egy-egy szimbólum felel meg. A lépések időben egymáshoz való viszonyát a szimbólumok síkbeli elhelyezkedésével, illetve összekapcsolásával ábrázolhatjuk.

Kezdő felhasználók számára mindenképpen célszerű a folyamatábra elkészítése és a folyamatábra alapján a program kódjának megírása, hiszen minden egyes szimbólum egy-egy utasításnak felel meg az adott programozási nyelven, így egy jól elkészített folyamatábra megadja a program kódját.

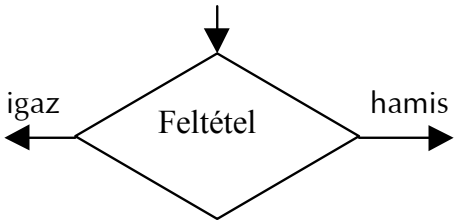
A folyamatábra szimbólumai:

Azt az algoritmus lépést, amelyet minden feltétel nélkül, egyszer hajtunk végre így jelöljük:



A téglalapba írjuk be az algoritmus lépést. A nyilak az algoritmusok időben lefolyását mutatják. Az ábrába bevezető nyíl előtt az időben korábban zajlódó folyamatok vannak, az ábrából kifele mutató nyíl pedig a következő utasításokat mutatja.

A választás lehetőségét a hegyére állított rombusszal ábrázoljuk. Ebben az esetben a választási lehetőségek száma kettő: igaz vagy hamis. Több egymás után következő rombusz esetében már bonyolultabb feltételrendszert is ki tudunk alakítani.



A rombusz belsejébe írhatjuk a választható feltételeket. Abban az esetben, ha az állítás hamis, akkor a „hamis nyíl" irányába fog az algoritmus folytatódni, ha igaz, értelemszerűen az „igaz nyíl" irányába.

1.2.5. A strukturált problémamegoldás

Egy adott feladatra a problémát megoldó algoritmust általában nem kapjuk meg készen, ezt nekünk kell elkészíteni, ki kell fejlesztenünk azokat. A probléma megoldása sikeresen elvégezhető strukturált módszerekkel.

A strukturált problémamegoldás három részből áll:

1. a probléma megfogalmazása és megértése
2. az algoritmus megtervezése
3. a program elkészítése

1. A probléma megfogalmazása és megértése: ez a rész segít megérteni az adott problémát, félreértéseket tisztázni és a nyitott kérdésekre válaszolni. Ebben a részben pontosan le kell írnunk:

- a modell elemeit
- az inputok és az outputok tartalmát és formáját
- a feldolgozás fő elemeit, az adatok közötti összefüggéseket

2. Az algoritmus megtervezése: az első pontban leírtak alapján fejlesztjük ki az algoritmust, definiáljuk a modell változóit, leírjuk az adatok szerkezetét és kapcsolatát. A változók közötti kapcsolatok meghatározása alapján összeállítjuk a megoldás elemeit és azokat logikus rendbe fűzzük.

Az algoritmus elkészítése lépésenként finomítva történik: először általában valamilyen pszeudonyelven írjuk le az algoritmust, utalva az egyes algoritmus részek kapcsolatára., majd a feladatot részekre bontjuk. Ezt követei a részfeladatok pontosítása addig, míg az egyes elemek egyértelművé nem válnak, vagyis az adott programnyelven közvetlenül leírhatók nem lesznek.

3. A második lépés elkészülte után következik a program elkészítése, vagyis a kódolás. Ez nem más, mint az adott algoritmust megvalósító program elkészítése.

A strukturált problémamegoldás előnye, hogy az algoritmusok részekre bontásával bármilyen bonyolultságú probléma megoldható, ha megfelelő sorrendben hajtjuk végre a lépéseket. Előnye még, hogy az első és második lépés megfelelő minőségű elvégzése után a programozónak már csak a kódolással kell foglalkoznia és nem kell menetköz-

ben a program kódján módosítani. Ezen előnyök jelentősen csökkentik a hibalehetőségeket, leegyszerűsítik a programozást, valamint egy esetleges későbbi programmódosítás során a javítások végrehajtását.

Nézzünk egy feladatot a strukturált feladat megoldási módszerre!

Két általunk megadott numerikus érték (szám) közül azt a számot kell kiírni a képernyőre amelyik nagyobb. Tervezzük meg azt az algoritmust amelyik bekéri tőlünk a két számot és eldönti, hogy az első szám vagy a második szám a nagyobb!

Jelöljük a két számot így: SZAM1, SZAM2. A nagyobbat tegyük a NAGYOBB változóba.

Amelyik szám nagyobb, írjuk ki a képernyőre.

Az algoritmus első közelítésben az alábbi lépésekből fog állni:

- SZAM1 és SZAM2 bekérése
- a nagyobb szám kiválasztása
- az eredmény kinyomtatása.

Az első és a harmadik lépés egyszerű, de a második már bonyolultabb, ezért ez további finomításra szorul:

ha SZAM1 > SZAM2, akkor NAGYOBB = SZAM1

ha SZAM1< SZAM2, akkor NAGYOBB = SZAM2

Ezek után már bármilyen számunkra ismert programnyelven kódolhatjuk a feladatot.

1.2.6. A program készítésének folyamata

A probléma megfogalmazása, az algoritmus elkészítése és a kódolás után még további munkafázisok vannak:

- formális ellenőrzés
- program tesztelés
- éles használat

A formális ellenőrzés során az alkalmazott fordítóprogram segítségével az kódolás szintaktikai (nyelvhelyességi) ellenőrzése történik meg. Ha a kód szintaktikailag helyes, akkor a fordítóprogram a gép számára használható kódot készít. Abban az esetben, ha szintaktikai hibát talál a fordítóprogram, jelzi azt és a kódot nem hagyja gépi nyelvre lefordítani. A hiba javítása után újbóli ellenőrzéskor - ha minden hibát kijavítottunk - a fordítóprogram gépi nyelvre fordítja a programot.

A következő lépés a tesztelés. Ebben a fázisban olyan próbaadatokkal töltjük fel a gépet, amelyek lehetőséget biztosítanak a hibák feltárására (pl.: az előző feladat esetén a program nem fog helyesen működni, ha a két szám azonos). Itt még célszerű kevés adattal dolgozni, hiszen

az itt végzett munka jó program esetén hiábavaló. Ettől függetlenül ezt a lépést nem célszerű kihagyni, mert egy esetlegesen rosszul működő program hibáira fényt derít, mielőtt még élesben használnánk a programot. Ha ebben a fázisban hibásan működik a program a probléma megfogalmazásától újra kell kezdeni a munkát. Ez első ránézésre időrablónak tűnhet. Általában nem az egész feladatmegoldás rossz, hanem annak egy kis hányada. A hibák felderítése érdekében mindenképpen célszerű a korábban készített dokumentáció áttekintése és javítása.

Ha a teszteléssel végeztünk, jöhet az éles használat, amikor valódi adatokkal töltjük fel a programot. Ha a tesztelésünk lelkiismeretes volt és jó eredménnyel zárult, az esetek többségében az éles üzemeltetés sem fog gondot okozni. Bonyolultabb programok esetében azonban előfordulhat, hogy a tesztelés során van olyan programrész, amit nem teszteltük le és éles futtatásnál derül csak ki a hiba. Ekkor is vissza kell menni a feladat megoldásának elejéhez és ki kell javítani a programot.

A program írója, illetve tesztelője, ha mód van rá, két különböző személy legyen.

1.3. Programozási alapfogalmak

A programokat karakterenként írjuk. A programnyelvek általában rögzített karaktereket használhatnak.
Ezek a jelek:

az angol ABC betűi és az aláhúzásjel: A...Z, a...z, _

a számok: 0...9

speciális karakterek: +, -, *, /, = , <, >, (,), [,], ., ,: ,; ,, ', #, $

A programok legkisebb egységei az illető nyelv szimbólumai. Ezek az elemek, melyekből egy programot fel lehet építeni:

- a programozási nyelv által előírt kulcsszavak
- a programozó által képzett nyelvi elemek (változók, azonosítók, konstansok)
- különböző műveleti és elhatároló jelek.

E szimbólumokból meghatározott szabályok szerint utasításokat lehet összerakni, amelyek valamilyen műveletet végeznek el a gépen. Az utasítások sorozata adja a megoldandó programot.

A fejezetben a példaprogramok TURBO PASCAL-ban és BASIC-ben íródtak, bemutatva ezáltal, mennyire mások és egyben mennyire ha-

sonlóak a különböző programnyelvek. A példaprogramok szemléltetik az adott probléma megoldását a különböző programnyelveken.

1.3.1. A programírás konvenciói

A nyelv szintaxisát azok a szabályok alkotják, amelyek a programozási nyelv elemeinek helyes használatát biztosítják. A megengedett nyelvi szerkezetek jelentéstartama a szemantika.

A programozási nyelvekben vannak bizonyos „kulcsszavak", amelyeknek speciális jelentése van, ilyen szavak például az utasítások, függvények is. Ezek az úgynevezett fenntartott szavak, melyek csak igen korlátozott módon használhatók bizonyos feladatokra.

1.3.2. Az azonosítók

Programozás során gyakran szükséges olyan karaktersorozatot használni, amit a programozó definiál változónévként vagy egyéb azonosítóként. Programnyelvektől függetlenül ezek nem kezdődhetnek számjeggyel, nem tartalmazhatnak speciális karaktereket, valamint szóközt. Az azonosító maximális hossza programnyelvtől függő, de általában 8 karaktert használnak.

1.3.3. A konstans

Konstans olyan adat, amely a műveletek során nem változik meg, értéke állandó marad. A legtöbb programnyelvben többféle típusú konstans lehet:

- Egész konstans: pozitív, negatív számok, valamint a nulla: (pl.: 1, -134, 0)
- Valós konstans: minden olyan szám, amelyben tizedespont szerepel (pl.: 1,2 vagy –0,13)
- Karakter konstans: egyetlen karakter, ami lehet szám, betű, valamint speciális karakter
- Szó konstans: több karakterből álló karaktersorozat. Néhány karakter kivételével minden karakter használható.
- Boole-féle konstans: két lehetséges értéke van: igaz vagy hamis

1.3.4. A változók

A változó olyan objektum, amelynek értéke a program futása során változik. A változókra azonosítókkal hivatkozunk. A változó típusai hasonlóan programnyelvtől függően változhatnak, de megegyeznek a

konstansok típusaival. Egy adott típusú változó csak a neki megfelelő értékeket tartalmazhatja (nem lehet egy egész változóba egy szót tárolni), illetve művelet csak azonos típusú változók között lehetséges. Ha nem ilyen műveletet próbálunk végrehajtani, az típushibát fog okozni.

Minden programozási nyelvben vannak olyan függvények, amelyek a változók közötti típusváltást elvégzik, abban az esetben, ha ez lehetséges, ilyen átalakítás lehet például:

- számból karakter
- karakterből szám, ha a karakter csak számokat tartalmaz

Ezek programnyelvtől függenek.

1.3.5. kifejezések

A kifejezések konstansokból, változókból, műveleti jelekből illetve zárójelekből és függvényekből állhatnak. A zárójelek a műveletek végrehajtásának sorrendjét határozzák meg.
Példa kifejezésre:

```
(a+2)-(cos(30°)/b)*d
```

a leggyakrabban használt műveleti jelek:

+	összeadás
–	kivonás
*	szorzás
/	osztás

1.3.6. A műveletek végrehajtásának sorrendje (prioritása)

A műveleteket mindig a matematika szabályai szerint kell végrehajtani.

Először mindig a zárójelben lévő műveletet kell végrehajtani.

A szorzás és az osztás magasabb prioritású művelet, mint az összeadás és a kivonás, tehát azokat előbb kell végrehajtani. A szorzás és az osztás, valamint az összeadás és a kivonás azonos prioritású. Azonos prioritású műveletek esetén a végrehajtási sorrend balról-jobbra történik. Az aritmetikai kifejezésekben a függvények egy számított értékkel térnek vissza, ezt az értéket fogja a kifejezésbe behelyettesíteni (a függvényekről később részletesebben lesz szó).

1.3.7. Boole-féle algebra

A programozás során nagyon gyakori feladat, hogy valamilyen adatok összehasonlításának eredményétől függően kell valamilyen utasítássorozatot végrehajtani. Egy összehasonlítás eredménye vagy igaz vagy hamis lehet, több választási lehetőség nincs. Sokszor akad olyan feladati is, hogy több adatot kell valamilyen logikai rendszer szerint összekapcsolni, ehhez nyújt segítséget a Boole-féle algebra ismerete.

A Boole-féle algebra három elemet tartalmazhat az összehasonlítások kombinálására:

AND (és): Akkor és csak akkor igaz, ha A és B is igaz,
OR (vagy): Akkor igaz, ha A vagy B igaz,
NOT (nem): Akkor igaz, ha A hamis,
A és B logikai értékek. Értékük vagy igaz (true) vagy hamis (false)

A logikai értékek ábrázolására igazságtáblázatot szoktak használni, amely tartalmazza a lehetséges alternatívákat.

Az AND (és) művelet igazságtáblája:

A	B	A és B
0	0	0
1	0	0
0	1	0
1	1	1

Az OR (vagy) művelet igazságtáblája:

A	B	A vagy B
0	0	0
1	0	1
0	1	1
1	1	1

A NOT (nem) művelet igazságtáblája:

A	Nem A
1	0
0	1

A műveletek prioritása:
Zárójelben lévő műveletek
NOT művelet
AND, OR művelet

Ahhoz, hogy két vagy több adatot össze tudjunk hasonlítani, meg kell határozni, milyen összehasonlító (reláció) műveletet alkalmazzunk:

= egyenlő
> nagyobb, mint
< kisebb, mint
>= nagyobb vagy egyenlő
<= kisebb vagy egyenlő
<> nem egyenlő

A reláció jelekkel aritmetikai kifejezéseket hasonlíthatunk össze, az összehasonlítás általános alakja:

```
„kifejezés1” „relációs jel”  „kifejezés2”
```

pl.: a következő aritmetikai művelet csak akkor igaz, ha A nagyobb, mint B, ellenkező esetben hamis.

```
A > B
```

Az egyes kifejezésekben használhatók aritmetikai műveletek, ezeket célszerű zárójelbe tenni.

Pl.: `(A-5) > (B*3)`

Logikai műveleti jeleket csak logikai kifejezésekre lehet alkalmazni. Ha A, B és C numerikus változók, és azt akarjuk hogy a logikai művelet csak abban az esetben legyen igaz, ha A nagyobb B-nél és C-nél is azt így kell felírnunk:

```
(A > B) AND (A > C)
```

Nem így :

```
A > (B AND C),
```

mivel B és C aritmetikai értékek és nem logikai értékek.

1.3.8. Az értékadó utasítás

A számítások végrehajtásához a számítógépnek meg kell adni milyen műveletet hajtson végre, milyen adatokkal, és melyik változóba kerüljön. Az értékadás formája a következő:

Változó = kifejezés,

Pl.: `A =12 vagy`
`C = 3*12*(R+39)*sin(30°)`

Ahol az egyenlet bal oldalán lévő változóba kerül az egyenlet jobb oldalán lévő kifejezés értéke. Látható, hogy a kifejezésben szerepelhet

szám, függvény, változó és konstans is. Természetesen az értékadáskor a változó típusától függően megadhatunk karaktereket, szavakat, illetve logikai értékeket is.

```
BETU = 'P'
SZO = 'De jó!'
LOGIKA = true (igaz)
```

Az értékadás művelete programnyelvenként eltérő lehet.

1.3.9. Input és output

A fejezet elején már volt szó arról, hogy a probléma megoldása során az ismert (input) adatoktól jutunk el az ismeretlenig (output), megfelelő algoritmussal. Az input/output utasítások a környezettel való kapcsolattartást biztosítják. A kapcsolattartás billentyűzeten, mágneslemez egységen, mágnesszalag egységen, illetve a képernyőn keresztül történhet.

1.3.9.1. Az input

Az input célja, hogy a program számára adatokat vigyünk be a gép memóriájába, amit a program futása során feldolgoz.

1.3.9.2. Az output

A program az input feldolgozása után valamilyen eredményre jut, ezt az eredményt nevezzük outputnak. Az output megjelenése a képernyőn vagy a nyomtatón, gyakran mágneses tárolóegységen történhet. Az output megjelenítése az esetek többségében formázottan történik, ez azt jelenti, hogy meg kell adni a megjelenített adat megjelenítési környezetét, milyen hosszú lehet a kiírandó karaktersorozat, milyen legyen a színe stb.

1.3.10. A soros programozás

Minden programnyelvben megvannak azok a szabályok, amelyek meghatározzák a program szerkezetét. Ilyen szabályok az utasítások szerkezete, megadásának sorrendje. A soros programozás nem biztosít lehetőséget döntések (szelekció), illetve ciklusok (iteráció) kezelésére.

Ettől függetlenül fontos ismernünk a soros programozás lehetőségét, hiszen az utasítások nagy része sorosan követi egymást, például egy cikluson belül is.

Az utasítok sorrendjére a programozás során nagyon oda kell figyelnünk, ha két utasítást összecserélünk, az az egész program eredményére hatással lehet, nagyon valószínű, hogy a program nem azt fogja csinálni amit elvárunk tőle.

1.3.11. A programok szerkezete

A program írásának első lépése, hogy deklaráljuk a program számára a számítógépbe bekerülő és a végrehajtás során használandó változók helyét. Ez a deklaráció néhány nyelvben nem kötelező, de a programnyelvek többségében elkerülhetetlen (BASIC-ben nem szükséges). A deklarációs rész tartalmazza a konstansokat, valamint a program futási környezetének beállításait. Ezt a részt a program deklarációs részének nevezzük. Ezek az utasítások a nem végrehajtandó utasítások körébe tartozik. Magasszintű programozási nyelveknél a fordítóprogram dönti el, hogy az adott változóhoz mekkora tárterületet foglal le, a változó típusa alapján. A tárolási helyekre az azonosítókkal hivatkozhatunk.

A deklarációs részt követik a végrehajtandó utasítások. Az utasításokon kívül itt célszerű elhelyezni a megjegyzéseket (commenteket), melyek a program működésének leírására szolgálnak.

Példák soros programozásra:
1, Egy általunk megadott szöveg kiíratása a képernyő középső sorba.

A probléma megoldásának menete:

1.Változó deklarálása
2.Képernyőtörlés
3.Adat bekérése
4.Pozicionálás középső sorba
5.Az adat kiíratása

A feladat megoldása BASIC nyelven:

```
CLS
INPUT  „Kérem a megadott szöveget:”, szoveg$
CLS
LOCATE 12,1
PRINT szoveg$
```

A feladat megoldása PASCAL nyelven:

```
PROGRAM P1;
USES CRT;
VAR
Szoveg: string[80]
```

```
BEGIN
CLRSCR;
READ(szoveg);
CLSSCR;
WRITELN(szoveg);
END.
```

2, Másodfokú egyenlet megoldása. A program megoldása során nem foglalkozunk azzal, hogy az egyenletnek esetleg nincs megoldása.

A megoldáshoz szükséges képlet:

$$GYOK_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Megoldás:

A, B, C bekérése
GYOK1 és GYOK2 kiszámítása
GYOK1 és GYOK2 kiíratása

Megoldás BASIC-ben:

```
CLS
INPUT „Kérem, adja meg az A-t:”; A
INPUT „Kérem, adja meg az B-t:”; B
INPUT „Kérem, adja meg az C-t:”; C
CLS
D = SQR ( B * B – 4*A*C)
GYOK1 = ( - B + D ) / ( 2 * A )
GYOK2 = ( - B - D ) / ( 2 * A )
LOCATE 5, 5
PRINT „ Az első gyök:”, GYOK1
LOCATE 7, 5
PRINT „A második gyök:”, GYOK2
```

Megoldás PASCAL-ban:

```
ROGRAM masodfoku;

{ A program nem foglalkozik azzal, ha az egyenletnek
nincs megoldása.
Nem megfelelő értékek esetén a program hibaüzenettel
leáll! }

USES crt;

VAR            {Változók deklarálása}
a, b, c : integer;
d, gyok1, gyok2: real;
```

```
BEGIN
CLRSCR;
WRITE('Kérem, adja meg az A értékét:');
READLN(a);
WRITE('Kérem, adja meg a B értékét:');
READLN(b);
WRITE('Kérem, adja meg a C értékét:');
READLN(c);

d := sqrt( b * b - 4 * a * c );
gyok1 :=( - b + d ) / ( 2 * a );
gyok2 :=( - b - d ) / ( 2 * a );

WRITELN('Az egyenlet első gyöke: ', gyok1);
WRITELN('Az egyenlet második gyöke: ', gyok2);
READLN;
END.
```

1.4. Alternatívák és ciklusok kezelése a programban

A mindennapi életben az egyik leggyakoribb művelet a döntés. A cselekvések döntő többsége függ valamilyen környezeti elemtől.
Pl.: Milyen az idő, mennyi ismeretünk van a döntéshez stb. Hasonló módon a programozás során is használni kell döntési mechanizmusokat. Ezek használata bizonyos probléma megoldásoknál elkerülhetetlen. A programozás során egy vagy több változó összehasonlításából jön ki egy logikai eredmény (igaz vagy hamis), a logikai eredmény alapján fut az A vagy a B algoritmus felé a program.

A logikai eredmény aszerint igaz vagy hamis, hogy a két vagy több érték közötti összehasonlítás során milyen eredményre jutunk.

Pl.:

ha A=3 és B=6, akkor

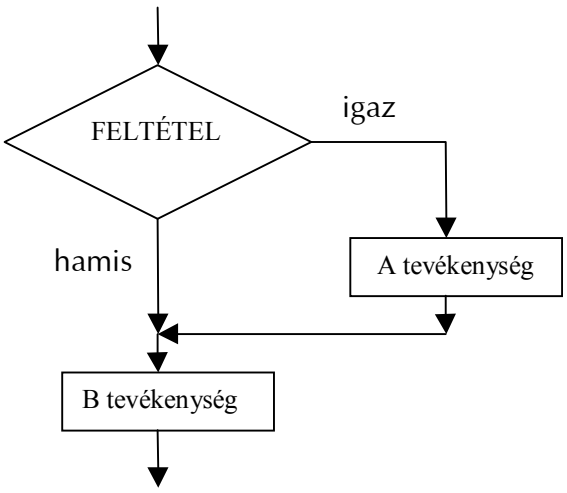
A > B eredménye HAMIS

A < B eredménye IGAZ

Az összehasonlítások között a Boole-féle algebra elemei használhatók.

1.4.1. Egyágú döntések

Az döntések lehetnek egyágúak abban az esetben, ha azt akarjuk, hogy az A algoritmus csak akkor hajtódjon végre, ha a feltétel teljesül, abban az esetben, ha a feltétel nem teljesül, fusson tovább a program, de az A algoritmust ne végezze el.



Feladat: Kérjen be a program egy számot, és csak akkor írja ki az üzenetet („A szám nagyobb tíznél”), ha a szám nagyobb, mint tíz.

A program megoldása BASIC-ben:

```
CLS
INPUT „Kérem, adjon meg egy számot:”, szam
IF szam > 10 THEN
   PRINT „A szám nagyobb tíznél”
END IF
```

A program megoldása PASCAL-ban:

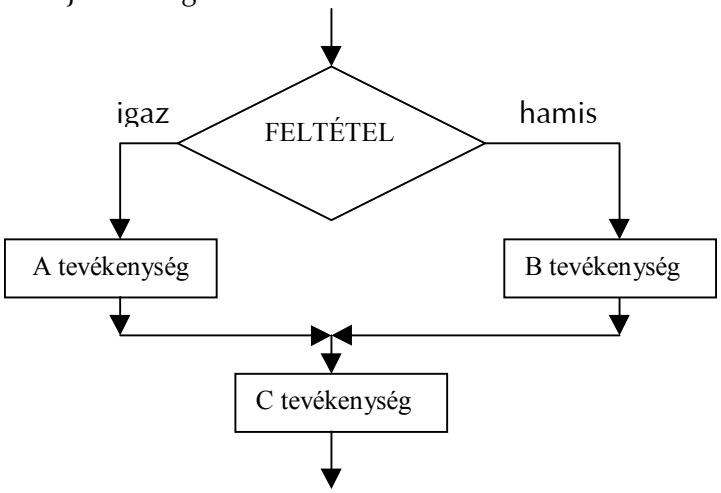
```
PROGRAM nagyobb10;
{ Ha 10-nél nagyobb számot adunk meg a program kiírja
azt }
USES crt;

VAR
szam : integer;

BEGIN
CLRSCR;
WRITE('Kérem, adjon meg egy 10-n,l nagyobb számot:');
READLN(szam);
IF szam > 10 THEN
WRITELN('A szám nagyobb, mint 10!');
READLN;
END.
```

1.4.2. Kétágú döntés

Abban az esetben beszélünk kétágú döntésről, ha a feltétel IGAZ voltakor az A algoritmus hajtódik vége, a feltétel HAMIS voltakor a B algoritmus hajtódik végre.



Feladat: Kérjük be egy személy életkorát. Ha több mint 18 éves, írjuk ki, hogy a személy nagykorú, ha nincs még 18 éves, írjuk ki, hogy nem nagykorú.

A program megoldása BASIC-ben:

```
CLS
INPUT „Kérem adja meg az életkorát:”, kor
IF kor > 18 THEN
 PRINT „Nagykorú”
ELSE
 PRINT „Nem nagykorú”
ENDIF
```

A program megoldása PASCAL-ban:

```
PROGRAM nagykoru;
USES crt;

VAR
kor : integer;

BEGIN
CLRSCR;
WRITE('Kérem adja meg a személy életkorát:');
READLN(kor);
IF kor > 18 THEN
WRITE('A személy nagykoru.')
```