C++ nyelvű algoritmusok

KENDE MÁRIA[♣], NAGY ISTVÁN[♣]

2005. szeptember 17.

TARTALOM

♣ Budapesti Műszaki Főiskola, Neumann János Informatikai Kar

BEVEZETÉS

E jegyzet célja segítséget nyújtani a hallgatói számára a C++ nyelv alapjainak elsajátításához.

A jegyzet négy részből áll.

Az első rész egy hierarchikus fogalomgyűjtemény, melynek elsődleges célja az anyag szócikkeinek tematikus rendbe állítása. Az egyes fogalmak kidolgozottsága vázlatos, a hozzájuk tartozó leírás mellett legalább ilyen fontos a helyzetük a fogalmak rendjében.

A második rész a programtervezésbe kíván bevezetni egy konkrét feladat megoldásán keresztül.

A harmadik rész egy programozási feladatgyűjtemény.

Végül az anyag legfontosabb része a Példatárban található C++ nyelvű mintaprogramok gyűjteménye. Ezek a programok az egyszerűbbektől az összetettebbekig, a kis apró jelenség, vagy eszközbemutató programtól a komolyabb algoritmikus gondolkodást is igénylőig egyetlen célt szolgálnak; az olvasó eljuttatását a C++ nyelven való programozás alapjainak elsajátításához. Ennek érdekében az egyes programokat áttekinthetően struktúráltuk, valamint sűrűn elláttuk megjegyzésekkel, melyek az egyes részek megértésén túl sokszor a hasonló feladatok megoldására is felkészítenek. E programokat a Borland cég Turbo C++ 3.0 fejlesztő környezetében teszteltük, de tapasztalataink szerint más környezetekben újrafordítva is megfelelően működnek.

A felkészüléshez javasolt módszer a következő:

1. A hallgató az anyag egyes fogalmait a hierarchikus fogalomrendszerbeli bemutatás sorrendjében egymásután gondolja végig; biztonságosan érti-e, tudja-e alkalmazni.
2. Ha a leírás az éppen tanulmányozott fogalom használatához hivatkozik egy Példatárbeli mintaprogramra, akkor vegye elő annak listáját (ha számítógépnél ül, akkor töltse be az általa használt C++ fejlesztőrendszert e programmal), és gondolja át a működését. Számítógép előtti tanulás során ajánlatos a programot lefuttatni, esetleg egyes, a megértés szempontjából kritikus változók értékének a futás során való folyamatos vizsgálatát elvégezni a nyomkövető (a debugger) segítségével. Különösen hasznos a mintaprogram egyes apróbb-nagyobb változtatásainak hatását megfigyelni (persze az eredeti program megfelelő elmentését követően).

Nem célja tehát ennek az anyagnak (mint ahogy a tantárgynak sem) a C++ nyelv teljes ismertetése, azonban biztosak vagyunk abban, hogy a fenti módon felkészülő hallgatók megfelelő tájékozottságot és jártasságot szereznek a C++ nyelven való programozáshoz anélkül, hogy drága könyvek beszerzésére volna szükségük. Természetesen a későbbiekben (akár a tanulmányaik folytatásához is) hasznosak lehetnek a megfelelő szakkönyvek. Jelen anyag végén, ajánlott irodalomként a magyar nyelven elérhetők közül is megadtunk néhányat.

A leírásban felhasználunk néhány szokásos metanyelvi határolójelet. Ezek az alábbiak:

< fogalom megnevezés>, [<opcionális nyelvi elem>], "<a nyelv szimbóluma>"

A programrészleteket, kulcsszavakat `Courier` betűtípussal írtuk, így ott metanyelvi jelölések nem szerepelnek. Az utasítások leírásánál használt tagolások nem kötelezőek, de a programozói gyakorlat szerint célszerűek.

I. rész Hierarchikus fogalomgyűjtemény

A C++ programok felépítése

deklarációs file-ok

könyvtárhasználat a C++ programokban

- A C++ nyelv sajátossága, hogy eszközeinek jelentős részét nem tartalmazza a nyelv definíciója, és csak a fordítóprogramhoz mellékelt függvénykönyvtárak segítségével lehet alkalmazni. Mivel e könyvtárak nélkül a nyelv gyakorlatilag használhatatlan lenne, ezért ezeket is szabványosították, és a különböző operációs rendszer környezetekben (pontosan definiált módon) általában egyaránt elérhetők.
- A deklarációs file-ok a programban hivatkozott függvények prototípusait (lásd később a "Függvények" címszónál!) tartalmazzák, annak érdekében, hogy a fordítóprogram (compiler) a hivatkozott függvényeket a függvénykönyvtárakból a programhoz tudja szerkeszteni (linkelni).

példa a deklarációs file-ok beszerkesztésére

(itt a <> zárójelek nem metanyelvi jelek, hanem az utasítások kötelező részei)

```
#include <iostream.h>
#include <math.h>
```

globális azonosítók definiálása (elmaradhat)

(lásd az "azonosító használat"-nál!)

prototípus modell (függvénystruktúra) (elmaradhat)

a program függvényeinek prototípus-listája

(lásd a "Függvények"-nél!)

függvény definíciók

a program függvényeinek leírása (C++ nyelvű forráskódok)

- Egy C++ program mindig tartalmaz egy (és csak egy) `main` nevű függvényt, melyben a program elkezdődik, és ahonnan indulva hajtódik végre a program által megvalósított algoritmus.
- Részletesen lásd a "Függvények"-nél!

A C++ nyelv elemei

terminátor-jel használat ("`;`")

- A C++ nyelvben a definíciókat, deklarációkat és utasításokat a "`;`" terminátor jel zárja le.

azonosító használat

egy azonosító (azaz név) az angol ábécé betűiből, számokból és az aláhúzás ("_") jelből épülhet föl, és betűvel kezdődik (a kis- és nagybetűk különbözőnek számítanak)

azonosítók hatásköre (vagy másképpen: érvényességi köre)

globális azonosítók

a `main` és a többi függvényen kívül definiált (deklarált) azonosítók

- Globális változót nem célszerű használni.
- A program egésze szempontjából fontos konstansokat és típus-leírásokat célszerű globális azonosítóként bevezetni.

lokális azonosítók

a `main`, vagy bármely más függvényben definiált (deklarált) azonosítók

hatáskör feloldás (lásd az "Objektumok"-nál!)

sorszámozott elemi típusok

`int, long int, char, unsigned char`

memóriabeli méretük, értékkészletük

lebegőpontos elemi típusok

`float, double`

memóriabeli méretük, értékkészletük

konstans definíció

`const` <elemi típus neve> <azonosító> "=" <literál>[<típusjelölő>]`;`

ahol a

<literál> egy adott elemi típusú érték, de a helyén állhat egy, fordítási időben kiértékelhető konstans kifejezés is (például egy `sizeof()` függvény, lásd később!),

<típusjelölő> általában egy karakter, mely a compilernek jelzi, hogy a megadott konstanst milyen típusú adatnak kell tekintenie (nem minden típushoz létezik).

- A `const` minősítő-szóval ellátott változó a program futása során nem változtathatja meg az értékét (erről az úgynevezett run-time rutin gondoskodik, melyet a compiler szerkeszt a programhoz, és többek között különböző futási idejű ellenőrzéseket végez.
- A `const` minősítő-szó bármely változót inicializáló deklaráció elé kitehető (lásd még az "Összetett adattípusok" inicializálásainál!).
- konstans literál használata kifejezésben

<literál>[<típusjelölő>]";"

példák elemi konstansokra (TC039.cpp)

```
const int             MAX = 5000;
const long int        MIN = -287000L;
const char            UTOLSO = 'Z';
const char            UJSOR_JEL = '\n';
  // ahol '\n' a sorvég jelzésére szolgáló vezérlő karakter
const unsigned char   UMLAUT = 'ä';
const float           EGY = 1.0F;
const double          ALSO_HATAR = -2.3e-3;
  // ahol -2.3e-3 = -0.0023
const long double     PI = 3.141592653589793238 5L;
```

típus definíció

`typedef` <típusleírás> <típusnév>";"

ahol a <típusleírás> lehet elemi típusnév, vagy összetett típus definíció, melyben már szerepelhetnek az előzetesen definiált konstansok

- Az összetett típusok definícióit lásd az "Összetett adattípusok"-nál!
- Egy azonosítóval definiált típusleírás csak az azonosító hatáskörében használható.

változó deklaráció

<típusnév> <változónév lista>";"

ahol

<változónév lista> : egy változónév, vagy változónevek vesszővel elválasztott sorozata

<típusnév> : elemi, vagy összetett típus neve

- Egy változó csak az azonosítójának hatáskörében használható.

változó inicializálása

<elemi típus neve> <változónév> "=" <literál>[<típusjelölő>]";"

- Ha egy változó inicializálása elé kitesszük ki a `const` minősítő-szót, akkor a változó értéke a program-futás során nem változtatható meg (lásd a "konstans definíció"-nál!).
- Az összetett típusú változók inicializálását lásd az "Összetett adattípusok"-nál.
- A nem konstanst definiáló változó-inicializálás rontja a program áttekinthetőségét, ezért csak különösen indokolt esetben alkalmazzuk!

változó helyfoglalása a memóriában (a `sizeof()` függvény)

- Egy változó memóriabeli helyfoglalását a típusa határozza meg.
- A `sizeof()` függvény byte-ban megadja a paraméterében szereplő
  - » adattípus definiált hosszát, illetve
  - » változó deklarált hosszát.

~ jelentősége

• A `sizeof()` kiértékelése fordítási időben történik, így definíciókban és deklarációkban is felhasználható konstans kifejezésként (lásd lejjebb az "egydimenziós tömb-változó" deklarálásánál!).
• A C++ nyelvben a tömb- és stringkezelő függvények nem ismerik a paraméterükként megadott tömb-, illetve stringváltozó deklarált hosszát (mivel e függvények nem részei a nyelvnek). A szükséges információt ezért (általában második paraméterként) nekünk kell megadni. Erre a `sizeof()` függvényt szoktuk használni.

~ használata (TC032.cpp, TC041.cpp, TC043.cpp)

összeadás (+), kivonás (-), szorzás (*), osztás (/), maradék képzés (%)

• Ezek típusőrző műveletek, vagyis a velük végzett kifejezés értéke az első operandusuk típusát veszi fel. Ez esetenként értékvesztést eredményezhet (lásd a "típuskonvertáló operátor"-nál adott példát!).

adatábrázolási anomáliák

fixpontos túlcsordulási anomália és elkerülése

~ bemutatása (TC001.cpp, TC002.cpp, TC004.cpp)

lebegőpontos alulcsordulási anomália és elkerülése

~ bemutatása (TC003.cpp, TC005.cpp)

típuskonverziók

automatikus típuskonverziók

az értékadási anomália

• A különböző típusú változók közötti értékadás következtében

példa:

```
int  a;   float  b;
b = 4.5;  a = b;
// Értékvesztés: a értéke 4!
```

típusőrző műveletek számítási anomáliái

fixpontos túlcsordulás (a *, +, − műveletek esetén)

fixpontos operandusok osztása (a / művelet esetén)

típuskonvertálási anomáliák függvényeknél

példa:

```
float  a,b;
b = -4.5;  a = abs(b);
// Értékvesztés: a értéke 4!
```

~ elkerülése (TC011.cpp)

• Az automatikus típuskonverzióból adódó értékvesztés elkerülhető a típuskonvertáló operátor használatával

típuskonvertáló operátor

"(""<elemi típus neve>"")""<kifejezés>

példa:

```
int  a;   float  b,c,d;
a = 3;  b = 4.5;  c = a + b;  d = (float)a + b;
// Értékvesztés: a  c értéke 7.0! A d értéke már jó: 7.5
```

~ használata (TC011.cpp, TC033-1.cpp, TC033-2.cpp)

Függvények

alapelvek

- A C++ nyelvben az egyetlen programszervezési forma a függvény.
- Itt a függvények egyszintű struktúrát alkotnak, azaz nem ágyazhatók egymásba (vagyis egy függvény nem tartalmazhatja egy másik függvény definícióját).
- Egy C++ program mindig tartalmaz egy (és csak egy) `main` nevű függvényt, melyben a program elkezdődik, és ahonnan indulva hajtódik végre a program által megvalósított algoritmus.

függvény definíciója

<függvény típusa> <függvény neve> "`(`"<paraméter lista>"`)`"
"`{`"
[<lokális konstans és típusdefiníciók>]
[<lokális változó deklarációk>]
[<utasítások>]
[`return` [<visszatérési kifejezés>]]
"`}`"

ahol

<függvény típusa>

típusnélküli függvény esetén: `void`

- Ekkor a függvény eljárásként viselkedik, vagyis környezetével csak a tevékenységével (pl. beolvasás billentyűzetről, kiírás képernyőre, stb.), illetve a paraméterlistán keresztül tart kapcsolatot.

típusos függvény esetén

- A függvény típusa bármely elemi típus lehet.
- A függvény értékét a `return` utasítást követő visszatérési kifejezés határozza meg, melynek típusa a függvény típusával egyező kell legyen.

<paraméter lista>

- A paraméter lista elemi paraméterdeklarációk vesszővel elválasztott sorozata, ahol a paraméterek lehetnek érték- és címparaméterek.

értékparaméter

deklarációja:
<típusnév> <paraméternév>

használata:
a függvény bemeneti paramétere

címparaméter

deklarációja:
<típusnév>"`&`" <paraméternév>

használata:
a függvény be- és kimeneti paramétere

- Az összetett típusú paraméterek használatát lásd az "Összetett adattípusok"-nál!

<visszatérési kifejezés>

- Ez a függvény visszatérési értéke.
- A `return` utasítás szolgál a függvényből való kilépésre, valamint a függvény nevének a visszatérési kifejezés értékével való ellátására.

visszatérési érték típusos függvény esetén a
`return` <visszatérési kifejezés> utasítással
ahol a <visszatérési kifejezés> típusneve azonos a függvény típusnevével

visszatérés típusnélküli (`void`) függvény esetén a
» `return` utasítással történik
ekkor a függvény azonnal visszatér, vagy
» a függvény végén `return` nélkül

- Egy függvényben a `return` utasítás többször is szerepelhet.

függvény hívása

- Egy függvény hívása a nevével és az esetleges argumentumlistájával történik:
  <függvény neve> "`(`"<argumentumlista>"`)`"
  ahol az argumentumlista a függvény deklarációjában szereplő paramétereknek balról jobbra való megfeleltetése alapján az argumentumok vesszővel elválasztott sorozata, melyben egy argumentum a megfelelő paraméter típusnevével egyező típusnevű
  » változó a hívó programban (címparaméter esetén csak ez lehet), vagy
  » kifejezés (értékparaméter esetén lehet ez is).
- Ha a függvény definíciója nem tartalmazott paraméterlistát, akkor hívása:
  <függvény neve> "`()`"
- Egy függvényt függvényváltozóként és utasításként (eljárásként) is meghívhatunk
  » függvényváltozóként
  - A neve értéket hordoz, kifejezésben használjuk, értékadó utasítás jobboldalán áll.
  - Csak akkor használható ilyen módon, ha a függvény típusos.

  példa: (Egy karakter leütésére vár, és a leütött karakter kódját beteszi az `a` változóba)
  ```
  a = getch();
  ```
  » utasításként
  - Ekkor a nevével esetlegesen hozott értéket figyelmen kívül hagyjuk.
  - Ekkor is kommunikálhat a hívó programmal, de csak a paraméterlistáján keresztül.
  - Ha a függvény `void` típusú, akkor csak utasításként használhatjuk.

  példa: (Egy karakter leütésére vár, de azt nem használja fel)
  ```
  getch();
  ```

függvény deklarációja (prototípusa)

- Egy függvény deklarációjára (prototípusára) elsősorban akkor van szükség, ha a függvényre való hivatkozás (a függvény hívása) megelőzi a függvény definícióját (ezt nevezzük *előre-hivatkozásnak*). Ebben az esetben a deklaráció tájékoztatja a compilert
  » egyrészt arról, hogy a hivatkozás jogos volt (azaz nem egy definiálatlan függvényre történt hivatkozás),
  » másrészt arról, hogy a hivatkozás során az egyes argumentumok típusai összhangban vannak-e a definiált paraméterek típusaival.
- A program függvényeinek protípusait a program szerkezetének bemutatása érdekében az úgynevezett prototípus modellben általában akkor is megadjuk (bár ez nem kötelező), ha a fenti előre-hivatkozási feltétel nem áll fenn. (A prototípus modell tulajdonképpen a program függvénystruktúráját, azaz a hívási gráfját szemlélteti.)
- Egy függvény protípusa az alábbi alakú
  <függvény típusa> <függvény neve> "`(`"<a paraméterek típusainak listája>"`);`"
  ahol
  <a paraméterek típusainak listája> a függvénydefinícióban szereplő paraméterek típusneveit tartalmazza az előfordulás sorrendjében vesszővel elválasztva
  - A címparaméterek típusnevei mellett itt is kell szerepelnie az "`&`" jelnek.

függvények alapértelmezett argumentumai

- Egy függvénynek lehetnek alapértelmezett argumentumai (TC035-1.cpp). Ekkor a függvény hiányos argumentumlistával is hívható. A hiányzó argumentumok helyére az alapértelmezett értékek kerülnek, de argumentumok mindig csak az argumentumlista jobboldaláról hiányozhatnak, kihagyások nélkül. Végső soron az argumentumlista akár üres is lehet.
- Az alapértelmezett argumentumok beállíthatóak a függvény prototípusában is, a paramétertípusok nevei mellett (TC035-2.cpp).

függvénynév túlterhelése (overload)

- Egy függvénynévhez tartozhat több definíció is, ha a különböző definíciókhoz különböző prototípusok tartoznak. A program futtatásakor az a definíció lesz aktív, amelyik prototípusának megfelelő argumentumlistával történt a függvényhívás (TC036.cpp).

operátorok, mint speciális függvények

• Az operátorok (a műveletek, a "`,`" listajel, a relációk és az értékadó utasítás) speciális függvényeknek tekintendők a C++ nyelvben, ám ezeknél a visszatérési érték mellékhatásként jelentkezik, így azt nem célszerű felhasználni az áttekinthető programozás érdekében.

függvények használata

a függvényhasználat alapjai

TC030.cpp, TC031.cpp, TC034.cpp, TC035-1.cpp, TC035-2.cpp, TC036.cpp

függvények használata összetett algoritmusok megvalósításaiban

TC013-1.cpp, TC013-2.cpp, TC014.cpp, TC016.cpp, TC017.cpp, TC018.cpp, TC019.cpp, TC020.cpp, TC022.cpp

speciális függvények használata (lásd az "Objektumok"-nál!)

Utasítás szerkezetek

értékadó utasítás ("=")

Az értékadási szabály:

Az azonos típusnévvel deklarált változók egyetlen értékadó utasítással értéket adhatnak egymásnak.

Megjegyzések az értékadási szabályhoz:

• Tömbökre az értékadási szabály csak akkor érvényes, ha struktúrába ágyazzuk őket (lásd TC046.cpp), tehát tömbtípusú változók közvetlenül nem adhatnak egymásnak értéket egyetlen értékadó utasítással.

• Az értékadási szabály ebben a formában csak a C++ nyelvben teljesül (C-ben csak az alaptípusokra igaz).

• Különböző típusú változók közötti értékadás értékvesztést eredményezhet (lásd a "típuskonverziók"-nál), ezért általában kerülendő.

~ mellékhatása

• Az értékadó utasítás függvényként is viselkedhet, azaz visszadhat értéket.

• E mellékhatás az oka annak, hogy ha az "==" logikai relációjel helyett az "=" értékadó jelet használjuk, akkor nem jelez hibát a compiler, bár a program hibásan fut.

összetett utasítás

"`{`" <utasítás$_1$>"`;`" <utasítás$_2$>"`;`" ... "`;`"<utasítás$_n$>"`}`"

azaz egy összetett utasítás az "`{`" és "`}`" utasítás-zárójelek közötti utasítások sorozata.

~ mindenütt használható, ahol <utasítás> szerepel

üres utasítás ("`;`")

feltételes elágazás (if-else)

`if` "`(`"<logikai kifejezés>"`)`" <utasítás$_1$>"`;`" [`else` <utasítás$_2$>"`;`"]

ahol a <logikai kifejezés> valamilyen logikai reláció teljesülését vizsgálja, tartalmazhat logikai operátorokat is, és értéke igaz, vagy hamis lehet.

logikai operátorok

logikai műveletek (`&&`, `||`, `!`)

logikai relációk (==, `!=`, <=, >=, <, >)

zárójelhasználat

• Az áttekinthetőség érdekében akkor is célszerű zárójelet használni, ha a műveletek precedencia viszonyai ezt nem tennék kötelezővé.

~ használata

(TC013-1.cpp, TC013-2.cpp, TC015.cpp, TC019.cpp, stb.)

ciklusutasítások

• A ciklusutasítások segítségével egy, vagy több utasítás (az úgynevezett ciklusmag) bizonyos feltételtől függően többször végrehajtható.

ciklusutasítás típusok

elöltesztelő ~

<ciklusfej> <ciklusmag>";"

ahol

a ciklusfej alakja

`while` "("<logikai kifejezés>")"

a <ciklusmag> egy utasítás, vagy utasítások sorozata utasítás-zárójelek között, azaz egy összetett utasítás

• Speciális elöltesztelő ciklus a számláló ciklus, melynek ciklusfeje:

`for` "("<ciklusváltozó kezdőértékadása>";" <végérték vizsgálat>";" <ciklusváltozó növelése>")"

melyben a ciklusváltozó csak sorszámozott típusú lehet.

• Az elöltesztelő ciklusok esetén a ciklusfejben megfogalmazott feltétel teljesülése esetén hajtódik végre a ciklusmag (ez a `while` esetén a logikai kifejezés, a `for` esetén a végérték vizsgálat teljesülését jelenti).

hátultesztelő ~

`do` <ciklusmag>`;` `while` "("<logikai kifejezés>")";"

• A hátultesztelő ciklus esetén a ciklusfejet a `do` kulcsszó képviseli.

• A hátultesztelő ciklus esetén is a `while` utáni logikai kifejezés teljesülése esetén ismétlődik meg a ciklusmag végrehajtása.

• Fontos megjegyeznünk, hogy a hátultesztelő ciklusok ciklusmagja egyszer mindenképpen végrehajtásra kerül.

kiegészítő utasítások

ciklusutasítás megszakítása (`break` utasítás)

a vezérlés visszaadása a ciklusfejre (`continue` utasítás)

~ használatuk

(TC010.cpp, TC011.cpp, TC012.cpp, stb.)

többirányú elágazás (switch)

`switch` "("<sorszámozott típusú kifejezés>")"

"{" `case` <konstans kifejezés$_1$> ":" <utasítás$_1$>";" [`break`";"]

`case` <konstans kifejezés$_2$> ":" <utasítás$_2$>";" [`break`";"]

.....

[`default` ":" <utasítás$_n$>";" [`break`";"]]

"}";

~ mellékhatása (a vezérlés "továbbcsorgása" `break` hiányában)

Megjegyzés:

• A feltételes elágazások, a ciklus utasítások, valamint a többirányú elágazó utasítások tetszőleges mértékben egymásba ágyazhatók!

Összetett adattípusok

• Az összetett adattípusok (az implicit konstansdefiníciót tartalmazó felsorolt típuson kívül) valamilyen elemi, vagy már definiált adattípus(ok)ra épülnek. Ezek az alaptípusok maguk is lehetnek akár összetett adattípusok is.

• Az összetett adattípusokra definiált változókat (az elemi típusú változókhoz hasonlóan) szintén lehet inicializálni, továbbá az inicializálás elé kitéve a `const` minősítő-szót összetett típusú konstansok is létrehozhatók.

• Az összetett típusú konstansok közül kiemelkedik a string-konstans gyakorlati jelentősége, mivel a stringet a mindennapi felhasználásban a könyvtári függvények révén (és a legújabb C++ compilerekben már a nyelv elemeként is) önálló adattípusként használunk.

• Összetett típusú paramétereket általában célszerű címszerintiként használni, mivel ekkor a függvény meghívásakor csak a címet kell átadni (vagyis gyorsabb a függvényhívás), és nincs szükség az összetett típusú paraméter minden egyes elemének értékátadására (mely egy nagyméretű tömb átadásakor igen lassú lehet).
Példaként lásd a TC013-1.cpp, ellenpéldaként pedig a TC013-2.cpp programokat!
• Bemeneti stringparamétereket esetenként értékparaméterként adunk át, ha például a függvényhívásakor stringliterált akarunk megadni (TC035-1.cpp, TC035-2.cpp).

a felsorolt (enumerated) típusú változó

deklarálása közvetlenül (változó deklarációban)

`enum` <változónév> `"{"`<konstansnevek listája>`"};"`

deklarálása közvetve (típusdefinícióval) - AJÁNLOTT!

`typedef enum "{"`<konstansnevek listája>`"}"` <enum típus neve>`";"`

<enum típus neve> <változónév lista>`";"`

ahol <konstansnevek listája> : konstansnevek vesszővel elválasztott sorozata

• Az itt megadott konstansnevek implicit konstans definíciók!

• A listabeli konstansnevekhez a felsorolásuk sorrendjében a 0,1,2,... egészeket rendeli a belső reprezentáció.

• A felsorolt típus egy sorszámozott típus, tehát mindenütt állhat, ahol egész szám szerepelhet (például tömbindex helyén, vagy a `switch` utasításban).

~ használata (TC040.cpp)

a tömbtípus

• A tömb <méret> számú, azonos alaptípusú elemből áll, ahol
» az alaptípus bármely elemi, vagy összetett (pl. tömb-) típus lehet
» az indextípus bármely sorszámozott (pl. int, vagy enum) típus lehet
• Az első tömbelem indexe: 0, a másodiké: 1, ..., az utolsóé pedig <méret> - 1.

egydimenziós tömb-változó

deklarálása

közvetlenül (változó deklarációban)

<alaptípus> <változónév>`"["`<méret>`"];"`

közvetve (típusdefinícióval) - AJÁNLOTT!

`typedef` <alaptípus> <tömbtípus neve>`"["`<méret>`"];"`

<tömbtípus neve> <változónév lista>`";"`

ahol a <méret> megadása történhet

» konstans-literállal

példa:

```
typedef int     vtipus[5];
```

» konstans-azonosítóval

példa:

```
const int       MAX = 5;
typedef int     vtipus[MAX];
```

» konstans-kifejezéssel

példa:

```
typedef int     utipus[sizeof(vtipus)];
```

~ egy elemére való hivatkozás

a `vekt` nevű tömb `index`-edik elemére: `vekt[index]`

~ inicializálása (kezdő érték adása)

• Példaként tekintsük a következő vektort: vekt = (2.4, 3.9, -0.23, -5, 121)

~ közvetlen deklaráció esetén

```
float   vekt[5] = {2.4, 3.9, -0.23, -5, 121};
```

~ közvetett (típusdefiníciós) deklaráció esetén

```
typedef float   vekt_tipus[5];
vekt_tipus      vekt = {2.4, 3.9, -0.23, -5, 121};
```

~ implicit méret deklarációval

```
float   vekt[] = {2.4, 3.9, -0.23, -5, 121};
```

• Ebben az esetben a fordítóprogram automatikusan lefoglalja a szükséges memóriaterületet, vagyis a méret deklaráció az inicializálás révén történik.

méretének lekérdezése a `sizeof()` függvény segítségével

például a `sizeof(vekt)` függvény értéke 20 (mivel a float típus 4 byte-os)

~ használata

(TC012.cpp, TC013-1.cpp, TC013-2.cpp, TC014.cpp, TC020.cpp, TC022.cpp, TC046.cpp)

többdimenziós tömb-változó

deklarálása

közvetlenül (változó deklarációban)

<alaptípus> <változónév>"["<méret$_1$>"]"..."["<méret$_n$>"];"

ahol n implementációfüggő érték

közvetve (típusdefinícióval) - AJÁNLOTT!

`typedef` <alaptípus> <tömbtípus neve>"["<méret$_1$>"]"..."["<méret$_n$>"];"

<tömbtípus neve> <változónév lista>";"

egy elemére való hivatkozás

a `multi` nevű tömb (`i, j, k`) -adik elemére: `multi[i][j][k]`

inicializálása (kezdő érték adása)

• Példaként tekintsük a következő, `matrix` nevű 2-dimenziós tömböt:

$$\begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \end{bmatrix}$$

A *sorfolytonos kifejtés szabálya*:

A mátrixok kifejtése a belső reprezentációban sorfolytonos, azaz a fenti mátrix egy olyan 2 elemű tömbnek tekinthető, melynek az elemei 3 elemű tömbök. (A deklaráció természetesen ilyen módon is végezhető.)

~ közvetlen deklaráció esetén

```
int   matrix[2][3] = {{10,20,30}, {40,50,60}};
```

vagy a sorfolytonos kifejtés miatt:

```
int   matrix[2][3] = {10, 20, 30, 40, 50, 60};
```

~ közvetett (típusdefiníciós) deklaráció esetén

```
typedef  int    matrix_tipus[2][3];
matrix_tipus    matrix = {{10,20,30}, {40,50,60}};
```

vagy a sorfolytonos kifejtés miatt:

```
matrix_tipus    matrix = {10, 20, 30, 40, 50, 60};
```

~ implicit méret deklarációval

```
int   matrix[][3] = {{10,20,30}, {40,50,60}};
```

• Ebben az esetben a fordítóprogram automatikusan lefoglalja a szükséges memóriaterületet, vagyis a méret deklaráció az inicializálás révén történik. Természetesen - a sorfolytonos kifejtés szabálya miatt - ez csak az első dimenzió szerint történhet.

méretének lekérdezése a `sizeof()` függvény segítségével

például a `sizeof(matrix)` értéke 12, a `sizeof(matrix[1])` értéke pedig 6.

• Lásd a `sizeof` függvény használatát (TC032.cpp)!

a string típus

• A string egy karaktertömb, melynek a tényleges feltöltöttségét egy lezáró `NULL` (`'\0'`) karakter jelzi (ez nem más, mint az ASCII kódtábla 0 kódú karaktere).
Maga a C++ nyelv nem is ismeri a string típust (azaz egyszerűen tömbként kezeli), csak a `string.h` deklarációs file által elérhető könyvtári függvények biztosítják a kezelését.

• A tipikus stringkezelési tevékenységeket az alábbi könyvtári függvények végzik:
a string értékadást az `strcpy` és az `strncpy`, az összefűzést az `strcat` és az

`strncat`, a feltöltöttségi hosszúság lekérdezését az `strlen`, két string összehasonlítását az `strcmp` és az `strncmp`, végül pedig az alstring keresést az `strstr`.

~ használata

(TC035-1.cpp, TC035-2.cpp, TC041.cpp, TC046.cpp, TC060.cpp, TC061.cpp, TC062.cpp).

string-változó

deklarálása

közvetlenül (változó deklarációban)

`char` <változónév>"`[`"<méret>"`];`"

közvetve (típusdefinícióval) - AJÁNLOTT!

`typedef char` <stringtípus neve>"`[`"<méret>"`];`"

<stringtípus neve> <változónév lista>"`;`"

string-változóra való hivatkozás

a `szoveg` nevű stringre: `szoveg`

inicializálása (kezdő érték adása)

• Példaként tekintsünk egy 10 hosszúságúnak deklarált string változót, melybe a lezáró `NULL` karakter miatt tehát legfeljebb 9 karakteres szöveg tehető.

~ közvetlen deklaráció esetén

```
char   szoveg[10] = {'A', 'L', 'M', 'A'};
```

vagy stringliterállal végezve az inicializálást:

```
char   szoveg[10] = "ALMA";
```

~ közvetett (típusdefiníciós) deklaráció esetén

```
typedef  char   szoveg_tipus[10];
szoveg_tipus    szoveg = {'A', 'L', 'M', 'A'};
```

vagy stringliterállal végezve az inicializálást:

```
szoveg_tipus     szoveg = "ALMA";
```

~ implicit méret deklarációval (stringliterált használva) pedig:

```
char    szoveg[] = "ALMA";
```

• Ebben az esetben a fordítóprogram automatikusan lefoglalja a szükséges memóriaterületet, vagyis a méret deklaráció az inicializálás révén történik.

méretének lekérdezése a `sizeof()` függvény segítségével

például a `sizeof(szoveg)` függvény

értéke 10 a közvetlen, vagy közvetett deklaráció esetén, hiszen ennyi a deklarált hossza byte-ban, és

értéke 4 az implicit méret deklarációval, hiszen ennyi az `"ALMA"` literál hossza byte-ban.

a feltöltöttségi méretének lekérdezése az `strlen()` függvény segítségével

például az `strlen(szoveg)` értéke 4 mindhárom fenti esetben, hiszen 4 karaktert tartalmaz.

• Láthatóan a `NULL` karaktert nem számítja be, de persze nyilván érzékeli.

a string-tömb típus

• A string-tömb egy olyan tömb, melynek elemei azonos deklarált hosszúságú stringek.

• Egyéb tulajdonságai a tömbök és a stringek tulajdonságaiból következnek.

~ használata (TC042.cpp)

string-tömb típusú változó

~ deklarálása

közvetlenül (változó deklarációban)

`char` <változónév>"`[`"<méret1>"`]`" "`[`"<méret2>"`];`"

közvetve (típusdefinícióval) - AJÁNLOTT!

`typedef char` <stringtípus neve>"`[`"<méret2>"`];`"

`typedef` <stringtípus neve> <string-tömb típus neve>"`[`"<méret1>"`];`"

<string-tömb típus neve> <változónév lista>"`;`"

• Tehát egy string-tömb valójában egy kétdimenziós tömb.

~ egy elemére való hivatkozás

a `szoveg` nevű string-tömb `i`-edik elemére való hivatkozás: `szoveg[i]`

~ inicializálása (kezdő érték adása)

• Példaként tekintsünk egy olyan 3 elemű string-tömböt, melyben az egyes elemek 10 hosszúságúnak deklarált string változók (melybe tehát a lezáró `NULL` karakter miatt legfeljebb 9 karakteres szöveg tehető).

~ közvetlen deklaráció esetén (stringliterálokat használva)

```
char  tavasz[3][10] = {"Március","Április","Május"};
```

~ közvetett (típusdefiníciós) deklaráció esetén

```
typedef char   h_tip[10];
typedef h_tip  evsz_tipus[3];
evsz_tipus     tavasz={"Március","Április","Május"};
```

~ implicit méret deklarációval és stringliterállal pedig:

```
char  tavasz[][10] = {"Március","Április","Május"};
```

• Ebben az esetben a fordítóprogram automatikusan lefoglalja a szükséges memóriaterületet, vagyis a méret deklaráció az inicializálás révén történik.

~ méretének lekérdezése a `sizeof()` függvény segítségével

például a `sizeof(tavasz)` értéke 30, amennyi a tömb deklarált hossza byte-ban.

a stuktúra típus

• A struktúra tagjai (elemei) különböző típusokhoz is tartozhatnak, ahol

» az egyes tagok típusa bármely elemi, vagy összetett (pl. tömb-, vagy másik struktúra) típus lehet, és

» az egyes tagokra a hozzájuk rendelt nevükkel lehet hivatkozni.

• A struktúra típus definiálásakor az egyes tagok neveihez rendelt típusok megadásakor implicit változó deklaráció történik.

~ használata (TC042.cpp)

struktúra típusú változó

~ deklarálása

» közvetlenül (változó deklarációban)

Nem javasolható!

» közvetve (típusdefinícióval) - AJÁNLOTT!

`typedef struct` <struktúra-típus neve> `"{"`<tagdeklarációs lista>`"};"`

<struktúra-típus neve> <változónév lista>`";"`

ahol <tagdeklarációs lista> : tagdeklarációk pontosvesszővel elválasztott sorozata, az egyes tagdeklarációk pedig az egyes tagok típusainak megfelelő deklarációkat jelentik.

~ egy tagjára való hivatkozás

a `konyv` nevű struktúra változó `cime` nevű tagjára való hivatkozás: `konyv.cime`

~ inicializálása (kezdő érték adása)

példa:

```
typedef  struct   konyv_tipus
         {  char  szerzoje[20];
            char  cime[40];
            int   kiadas_eve;
            float ara;    // USD-ben megadva
         };
konyv_tipus   konyv = {"Herbert Schildt",
                  "C/C++ Programmer's Reference.",
                  1997, 21.40};
```

~ méretének lekérdezése a `sizeof()` függvény segítségével

például a `sizeof(konyv_tipus)` értéke 66,

amennyi a tömb deklarált hossza byte-ban

Megjegyzések az összetett adattípusokhoz:

- Az összetett adattípusok tetszőleges mértékben egymásba ágyazhatók.
- Értékadásukra az értékadási szabály vonatkozik (lásd feljebb az "értékadó utasítás"-nál!).
- Az összetett adattípusok esetén a típusdefiníció azért ajánlott, mert lehetővé teszi
  - » egyrészt a közvetlen értékadást az értékadási szabály szerint,
  - » másrészt az összetett típusú paraméterek használatát a függvényeknél.

Objektumok

- Az objektumok objektumtípusú "változók", melyeknek lehetnek
  - » adattagjai (ezek típusdefiníciók, vagy változók), és
  - » metódustagjai (ezek az objektum adattagjain transzformációkat végrehajtó függvények).

objektumtípusok

- » osztály (`class`)
- » struktúra (`struct`)
- Az osztály és a struktúra objektumtípusok lényegében ekvivalensek.
- Az osztályt használjuk jellemzően az objektumok létrehozására, vagyis olyan esetben, amikor metódustagra is szükség van (TC050.cpp, TC051.cpp).
- A struktúrát általában akkor használjuk, ha csak adattagok vannak (TC052.cpp).
- Megjegyezzük, hogy a korábban ismertetett struktúratípus definíció mellett létezik az alábbi is, de ez utóbbi használható a "hagyományos" struktúrák definiálására is.
- A továbbiakban általában csak az osztálytípussal foglalkozunk (a struktúratípusnál minden hasonló).

hatáskör kijelölés (láthatóság)

- » private (privát)
  - A "private:" címkével megjelölt (adat-, vagy metódus-) tagok csak az adott osztályban (illetve a származtatott osztályban) láthatók.
  - Az osztály objektumtípusban ez alapértelmezés, ezért ezt a címkét ott nem kell kitenni.
- » public (nyilvános)
  - A "public:" címkével megjelölt (adat-, vagy metódus-) tagok az adott objektum teljes érvényességi körében láthatók.
  - A struktúra objektumtípusban ez az alapértelmezés, ezért e címkét ott nem kell kitenni.

osztálytípus

~ definíciója

```
class <osztálytípus neve>
"{"private":"
     // A privát tagok definíciói, deklarációi
   public":"
     // A nyilvános tagok definíciói, deklarációi
"};"
```

~ metódustagja (osztály függvény)

beillesztett (beágyazott) metódustag

- Egy beillesztett metódus definíciója az osztálytípus definíciójában foglal helyet, és formailag egy függvény-definíció.
- Csak rövid függvény-definíciót célszerű belülre tenni, mert különben nem lesz áttekinthető az osztálytípus definíciója (TC050.cpp).

külső metódustag

- Egy külső metódusnak csak a prototípusa (deklarációja) van az osztálytípus definíciójában, maga a függvénydefiníció az osztálytípus definícióján kívül van.
- A külső metódus-definíció formailag annyiban tér el egy függvény definíciójától, hogy a metódus nevét az alábbi módon kell írni:

  <osztálytípus neve> "`::`" <metódus neve>

  ahol a "`::`" a hatáskör-kijelölés jele (TC051.cpp).

• A külső metódus-definícióban az adattagok nevére közvetlenül lehet hivatkozni.

konstruktor

• A konstruktor egy speciális metódus, melynek segítségével inicializálni lehet egy objektum adattagjait, mivel minden objektum-létrehozáskor automatikusan lefut. (Ennek megfelelően lehet egy konstruktor is belső, vagy külső.)

• A konstruktor tevékenysége általában nem más, mint hogy a függvényparamétereihez hozzárendeli az objektum adattagjait (esetleg csak egy részét) egy-egy értékadással.

• Amikor egy konstruktort tartalmazó objektumtípusnak létrehozzuk egy példányát, akkor az objektum neve mellé, függvényzárójelek között meg kell adni a hivatkozott adattagokat inicializáló literálokat /ekkor stringliterálok is használhatók a macskaköröm (") jelek között/.

• Ha egy objektum típusleírásában definiáltunk egy konstruktort, akkor azt objektum-létrehozáskor használni is kell.

• A konstruktornak is lehetnek alapértelmezett argumentumai (mint minden függvénynek), de üres argumentumlistát nem használhatunk. Az alapértelmezett argumentumok itt is beállíthatóak az objektumtípus definíciójában lévő konstruktor-prototípusban, a paramétertípusok nevei mellett (TC054.cpp).

• A konstruktorral inicializált objektum létrehozása az objektumot függvényhez teszi hasonlóvá, de persze az objektum nem függvény (lásd a TC054.cpp mintaprogramot, melyben a `sizeof` függvénnyel meghatároztuk az objektum adattagjainak összméretét byte-ban).

objektumok létrehozása (deklarációja)

• Egy objektum létrehozása nem más, mint egy osztály- (vagy struktúra-) típusú "változó" deklarálása (TC050.cpp).

<osztálytípus neve> <objektumlista>";"

ahol <objektumlista> az objektumok neveinek vesszővel elválasztott sorozata.

objektumok egy tagjára való hivatkozás

<objektum neve>"."<tag neve>

ahol a tag egyaránt lehet adattag, vagy metódustag (TC050.cpp)

objektumok értékadása

• Az objektumra is vonatkozik az értékadási szabály, vagyis az azonos objektumtípus különböző objektumpéldányai egyetlen értékadó utasítással adhatnak egymás adattagjainak értéket (TC050.cpp).

származtatott objektumok (öröklődés)

egy származtatott osztálytípus definíciója

```
class <osztálytípus neve>":" <láthatóság> <ős osztálytípus neve>
"{"
      // Osztálytípus definíciója
"};"
```

• Ha például a származtatott objektum az eredeti (az ős) objektumról "public" láthatósággal öröklődik, akkor az eredeti objektum adat- és metódustagjaira ugyanúgy lehet hivatkozni, mint a saját definícióbeliekre (TC055.cpp).

objektumok használata

(TC050.cpp, TC051.cpp, TC052.cpp, TC053.cpp, TC054.cpp, TC055.cpp)

Szabványos I/O kezelés

• A C++ a szabványos I/O kezelést az iostream objektum-könyvtár segítségével valósítja meg. Az ennek használatához szükséges deklarációs állomány az `iostream.h`, melyet a programjainkba az

```
#include <iostream.h>
```

precompiler utasítással szerkesztünk be.

• Az iostream könyvtár lehetővé teszi
  » a file-kezelést is (melyhez még az `fstream.h` deklarációs állományra is szükség van), és
  » a tömb-kezelést is (ehhez az `strstrea.h` deklarációs állomány kell még), valamint
  » a formázott I/O-kezelést is (ehhez az `iomanip.h` deklarációs állomány is kell).
• Az iostream könyvtárnak a programozók számára legfontosabb osztálya az ios, melyből származnak a leggyakrabban használt osztályok (melyekkel a legtöbb feladat megoldható):

Osztály	*Felhasználási kör*
istream	Általános bevitel
ostream	Általános kivitel
iostream	Általános be- és kivitel
ifstream	File bevitel
ofstream	File kivitel
fstream	File be- és kivitel
istrstream	Tömb alapú bevitel
ostrstream	Tömb alapú kivitel
strstream	Tömb alapú be- és kivitel

• A C++ predefinit csatornái
  » a cin, mely a szabványos bemenetet (alapértelmezésben a billentyűzetet) kezelő objektum, és az istream objektumból származik,
  » a cout, mely a szabványos kimenetet (alapértelmezésben a képernyőt) kezelő objektum, és az ostream objektumból származik.
  » a cerr, mely a szabványos hibakimenetet (alapértelmezésben a képernyőt) kezelő objektum, és az ostream objektumból származik.

II. rész Programtervezés

(Egy konkrét feladat megoldásán keresztül bemutatva)

BEVEZETÉS

Egy programozási feladat megoldását a kezdő programozók hajlamosak a program kódolásával indítani. Természetesen valamilyen előzetes modell ilyenkor is felépül bennük, azonban ez általában igen vázlatos, és legtöbbször nagyon intuitív. (Ez alatt azt értve, hogy sokszor a megoldás lényegét "a majd csak lesz valahogy" elv alapján a szőnyeg alá söprik, ami miatt sokszor egy, vagy több oldalnyi programozás után derül ki, hogy az egészet teljesen másképpen kellett volna csinálni...)

Az alábbiakban azt mutatjuk be egy konkrét feladat megoldásán keresztül, hogy egy program elkészítésének hogyan célszerű nekiállni. Ez nemcsak akkor hasznos, ha egy nagyobb lélegzetű problémát (például egy féléves feladatot) kell megoldani, hanem egy zárthelyi dolgozat esetén is. Az az idő ugyanis, amit "elvesztegetünk" a tervezéssel, bőven megtérül a program kódolási munkájánál.

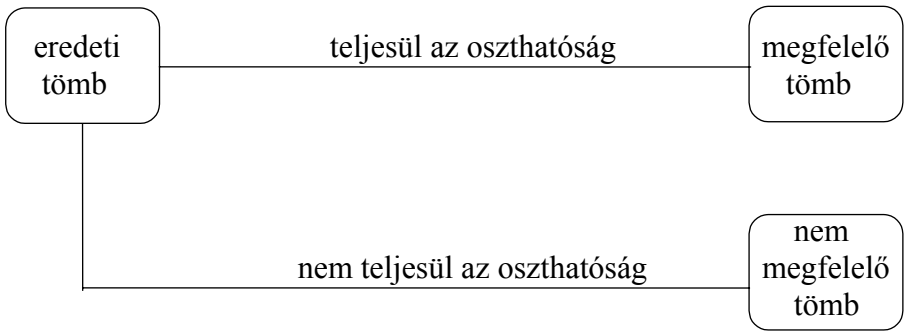
Végül megjegyezzük, az igazán nagy feladatok elkészítése során az alábbiakban bemutatott lépéseken túlmenően még további teendők is vannak (a megoldás modulokra bontása, memóriaigényvizsgálat, objektumok hatáskör- és élettartam elemzése, tesztelési tervek, valamint tesztfutási jegyzőkönyvek készítése, a felhasznált szoftvereszközök dokumentálása, stb.), melyekkel itt most nem volt célunk foglalkozni.

1. lépés: A feladat rövid, szöveges megfogalmazása

FELADAT: Tömbelemek szétválogatása két másik tömbbe adott T tulajdonság szerint, ahol T: a felhasználó által megadott számmal való oszthatóság.

2. lépés: A feladat elvi modelljének elkészítése

ELVI MODELL: (Itt kell megadni a feladat természete szerinti modellt. Például egy fizikai, vagy egy biológiai probléma esetén a fizikai, illetve a biológiai modellt.)



3. lépés: A feladat matematikai modelljének elkészítése

MATEMATIKAI MODELL: (Itt kell megadni az elvi modell matematikai leírását. Ez általában a feladat megoldásának algoritmus vázlatát is tartalmazza.
A matematikai modell leírását egy jól olvasható, különleges szimbólumokat lehetőleg nem tartalmazó ún. leíró nyelven elkészíteni. Ez a leíró nyelv általában leginkább a Pascalhoz hasonló.
Az alábbiakban a megjegyzéseket a // jel után helyeztük el.)

// MATEMATIKAI MODELL:
megfelelő_tömb := ∅;
nem_megfelelő_tömb := ∅;
minden index (index ∈ eredeti_tömb indexhalmaza) esetén:
    ha (eredeti_tömb[index] mod osztó) = 0
    akkor
        megfelelő_tömb := megfelelő_tömb ∪ eredeti_tömb[index]
    egyébként
        nem_megfelelő_tömb := nem_megfelelő_tömb ∪ eredeti_tömb[index];
// ahol
// ∅ jelölje az üres tömböt,
// ∈ jelölje a halmaz eleme relációt,
// ∪ jelölje a tömb-bővítés műveletét.
// Megjegyzés: Az eredmény tömbök tényleges méretét a fenti algoritmus végrehajtása állítja be!

4. lépés: A feladat adatmodelljének elkészítése

ADATMODELL: (Itt kell megadni a matematikai modell megvalósításához /a program implementáláshoz/ szükséges konstansokat, típusleírásokat, valamint változó deklarációkat.
Ez természetesen nem tartalmazza a program összes adatának leírását.
Az itt megadott konstansok és típusleírások általában globálisak, míg a változók általában csak lokálisak.
Az adatmodellt már célszerű az implementációs nyelven /ezúttal C++ nyelven/ elkészíteni.
A konkrét feladat esetén célszerű felfigyelni az adattömbök tényleges feltöltöttségét jelző méret jelentőségére.)

konstansok:
```
  const int   MAXTOMB = 20;  // A maximális tömbméret definiálása
```
típusok:
```
  typedef  struct { int  tomb[MAXTOMB]; // A tömb tagja, és
                    int  meret;         // a tényleges méret tagja
                  } tombtipus;          // a struktúra típusnak
```
változók:
```
  tombtipus   eredeti,          // A feladat
              megfelelt,        //   három
              nem_felelt_meg;   //   tömb-struktúrája
  int         oszto;            // A tömbelemek szétválogatásához
```

5. lépés: A feladat funkcionális modelljének elkészítése

FUNKCIONÁLIS MODELL: (Itt kell megadni a feladat részfeladatokra bontását. Ez tartalmazza az egyes alprogramok /függvények/ rövid szöveges leírását, a bemenő és kimenő paraméterek megadását, valamint a függvények formális specifikációit /azaz C++ nyelvben: a prototípusukat/.
Végül a funkcionális modell tartalmazza az úgynevezett hívási gráfot, mely a bevezetett alprogramok kapcsolatát mutatja be.)

`meret:`
leírás: Az eredeti_tömb szükséges méretét kéri be a felhasználótól
hívó függvény: main
bemeneti paraméter: nincs
kimeneti paraméter: az eredeti_tömb mérete
függvény érték: nincs
prototípus: `void meret(tombtipus&);`

`beolvas:`
leírás: A tömb tag felhasználó által történő feltöltése e struktúrában megadott méretben
hívó függvény: `main`
bemeneti paraméter: a feltöltendő struktúra méret tagja
kimeneti paraméter: a feltöltendő struktúra tömb tagja
függvény érték: nincs
prototípus: `void beolvas(tombtipus&);`

`oszto_bekeres:`
leírás: A T szétválogatási tulajdonság, vagyis az `oszto` bekérése
hívó függvény: `main`
bemeneti paraméter: nincs
kimeneti paraméter: a bekért osztó értéke
függvény érték: nincs
prototípus: `void oszto_bekeres(int&);`

`init:`
leírás: A paraméter-struktúra méret tagjának inicializálása
hívó függvény: `main`
bemeneti paraméter: nincs
kimeneti paraméter: a paraméter-struktúra méret tagja
függvény érték: nincs
prototípus: `void init(tombtipus&);`

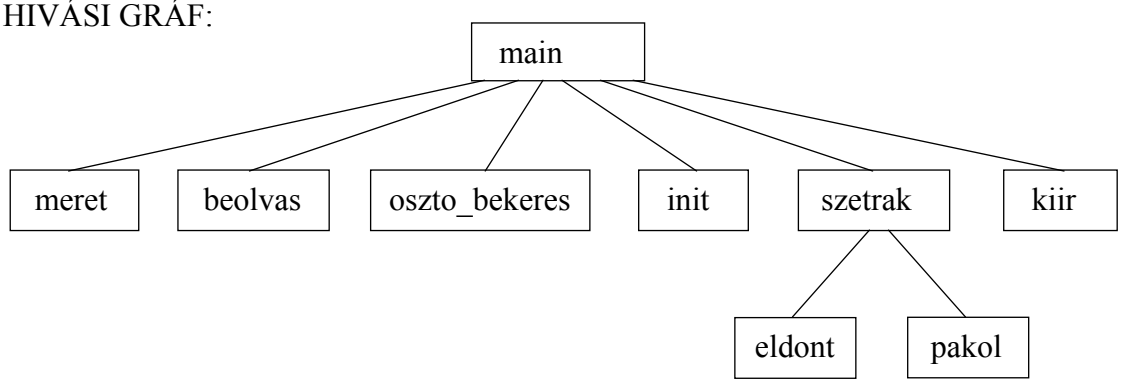
`szetrak:`
leírás: Szétválogatás a T tulajdonság, vagyis az `oszto`-val való oszthatóság szerint
hívó függvény: `main`
bemeneti paraméter: `oszto`
kimeneti paraméter: az `eredeti`, a `megfelelt`, és a `nem_felelt_meg` struktúrák
függvény érték: nincs
prototípus: `void szetrak(int, tombtipus&, tombtipus&, tombtipus&);`

`eldont:`
leírás: Az első paraméternek a második paraméterrel való oszthatóságát vizsgálja
hívó függvény: `szetrak`
bemeneti paraméter: az eredeti struktúra-tömb egy eleme, és az `oszto`
kimeneti paraméter: nincs
függvény érték: ha az oszthatóság teljesül, akkor a függvényérték `'I'`, egyébként `'N'`
prototípus: `char eldont(int, int);`

`pakol:`
leírás: A paraméter-struktúra tömb tagjához hozzáfűzi a másik paraméterként megadott elemet, és a struktúra méret tagját aktualizálja
hívó függvény: `szetrak`
bemeneti paraméter: az eredeti struktúra-tömb kijelölt eleme
kimeneti paraméter: valamelyik tulajdonság-struktúra tömb tagja
függvény érték: nincs
prototípus: `void pakol(int, tombtipus&);`

`kiir:`
leírás: A paraméter-struktúra tömbelemeinek kiíratása
hívó függvény: `main`
bemeneti paraméter: valamelyik struktúra tömb tagja
kimeneti paraméter: nincs
függvény érték: nincs
prototípus: `void kiir(tombtipus&);`

HIVÁSI GRÁF:



6. lépés: Az implementáció (a kódolás)

Az implementáció során az egyes függvények definícióinak (program kódjainak) elhelyezésénél célszerű a hívási gráf által mutatott hívási sorrendet betartani (ez persze nem mindig lehetséges). Ennek értelmében egy hívott függvény definíciója lehetőség szerint mindig közvetlenül előzze meg az őt hívó függvény definícióját. Bár a prototípus modell (a függvények prototípusainak felsorolása) lehetővé teszi az egyes függvény-definíciók tetszőleges elhelyezését, e módszerrel jobban áttekinthető (olvasható), logikusabb kódszerkezetet kapunk.

Az alábbiakban látható maga a program, valamint futtatásának képe valamely konkrét bemenő adatsorozatra.

```
// Program TC022.cpp
//-----------------------------------------------------------------
// Tömbelemek szétválogatása két másik tömbbe adott T tulajdonság
// szerint, ahol T: a felhasználó által megadott számmal való
// oszthatóság.
//-----------------------------------------------------------------
// Az utolsó javítás dátuma: 2000.04.15.
//-----------------------------------------------------------------

//-----------------------------------------------------------------
// A hivatkozott deklarációs file-ok
//.................................................................
#include <iostream.h>   // cin, cout
#include <iomanip.h>    // setw, setiosflag
//-----------------------------------------------------------------

//---------------------------------------
// Globális azonosítók
//.......................................
const    int      MAXTOMB = 20;          // A maximális tömbméret
typedef  struct   {int  tomb[MAXTOMB];  // A tömb tagja, és
                   int  meret;          // a tényleges méret tagja
                  } tombtipus;          // a tömbtípusnak
//---------------------------------------

//-----------------------------------------------------
// E modul függvénystruktúrája (prototipusok modell)
//.....................................................
//void main();
    void meret(tombtipus&);
    void beolvas(tombtipus&);
    void oszto_bekeres(int&);
    void init(tombtipus&);
```

```
    void szetrak(int, tombtipus&, tombtipus&, tombtipus&);
      char eldont(int, int);
      void pakol(int, tombtipus&);
    void kiir(tombtipus&);
//-------------------------------------------------------

//--------------------------------
//     A függvény definíciók
//--------------------------------

void meret(tombtipus&   adat)
// Az eredeti_tomb szükséges méretét kéri be a felhasználótól
{
  int   egesz;
  cout << "\nAdatbevitel a felhasználói tömbbe: \n\n";
  do {cout << "Az elemek száma (legfeljebb " << MAXTOMB << ") : ";
      cin >> egesz;
     } while (egesz <= 0  ||  egesz > MAXTOMB);
  adat.meret = egesz;
} //meret

void beolvas(tombtipus&   adat)
// A paraméter-struktúra tömb tagjának felhasználó által
// történő feltöltése e struktúrában megadott méretben
{
  int   i;
  cout << "Adja meg a tömb elemeit! \n";
  for(i = 0; i <= (adat.meret-1); i++)
    { cout << "A tömb " << i << "-edik eleme : ";
      cin >> adat.tomb[i];
    }
  cout << "\n";
} //beolvas

void oszto_bekeres(int&   egesz)
// A T szétválogatási tulajdonság, vagyis az oszto bekérése
{
  cout << "A T szétválogatási tulajdonság ";
  cout << "az Ön által adott számmal való oszthatóság, \n";
  cout << "ahol az osztó: ";
  cin >> egesz;
} //oszto_bekeres

void init(tombtipus&   egyik)
// A paraméter-struktúra méret tagjának inicializálása
{
  egyik.meret = 0;
} //init

char eldont(int    egy_elem,
            int    szam)
// Az első paraméternek a második paraméterrel való oszthatóságát
// vizsgálja
{
  if ((egy_elem % szam) == 0)
    return 'I';
```

```
  else
    return 'N';
} //eldont

void pakol(int          egy_elem,
           tombtipus&   egyik)
// A paraméter-struktúra tömb tagjához hozzáfűzi a másik paramé-
// terként megadott elemet, és a struktúra méret tagját aktuali-
// zálja
{
  egyik.tomb[egyik.meret] = egy_elem;
  egyik.meret++;
} //pakol

void szetrak(int          oszto,   // Az osztó
             tombtipus&   eredeti, // Az eredeti,
             tombtipus&   jo,      //  az oszthatóságot teljesítő,
             tombtipus&   nem_jo)  //  és a nem teljesítő tömb
// Szétválogatás a T tulajdonság, vagyis az oszto-val való osztha-
// tóság szerint
{
  int     futo_index;       // Az eredeti tömbhöz
  for (futo_index = 0; futo_index < eredeti.meret; futo_index++)
    if (eldont(eredeti.tomb[futo_index], oszto) == 'I')
      pakol(eredeti.tomb[futo_index], jo);
    else
      pakol(eredeti.tomb[futo_index], nem_jo);
}

void kiir(tombtipus&   egyik)
// A paraméter-struktúra tömbelemeinek kiíratása
{
  int   i;
  if (egyik.meret == 0)
    { // Ekkor nincs kiírandó elem
      cout << " --- nincs elem --- \n\n";
      // Kiugrás a függvényböl
      return;
    };
  for (i = 0; i < egyik.meret; i++)
     // A kiíratás mezöszélessége: 5,
     // az adatelhelyezés a mezöben jobbra tömörített.
     cout << setw(5) << setiosflags(ios::right) << egyik.tomb[i];
  cout << "\n\n";
} //kiir

/**********/
void main()
/**********/
{
  tombtipus   eredeti,          // A feladat
              megfelelt,        //   három
              nem_felelt_meg;   //   tömb-struktúrája
  int         oszto;            // A tömbelemek szétválogatásához
```

```
  // Az eredeti tömb tényleges méretének bekérése
  meret(eredeti);

  // Beolvasás az eredeti tömbbe
  beolvas(eredeti);

  // A T szétválogatási tulajdonság, vagyis az oszto bekérése
  oszto_bekeres(oszto);

  // A tulajdonság tömbök méreteinek inicializálása
  init(megfelelt);
  init(nem_felelt_meg);

  // Szétválogatás a T tulajdonság, vagyis
  // az oszto-val való oszthatóság szerint
  szetrak(oszto, eredeti, megfelelt, nem_felelt_meg);

  cout << "Az eredeti tömb elemei :\n";
  kiir(eredeti);

  cout << "A T tulajdoságnak megfelelö tomb elemei :\n ";
  kiir(megfelelt);

  cout << "A T tulajdonságnak nem megfelelö tömb elemei :\n";
  kiir(nem_felelt_meg);
} //main

/*
A program futtatása:
--------------------

Adatbevitel a felhasználói tömbbe:

Az elemek száma (legfeljebb 20) : 5
Adja meg a tömb elemeit!
A tömb 0-edik eleme : 65
A tömb 1-edik eleme : 173
A tömb 2-edik eleme : -42
A tömb 3-edik eleme : 0
A tömb 4-edik eleme : -5

A T szétválogatási tulajdonság az Ön által adott számmal való oszt-
hatóság,
ahol az osztó: 5
Az eredeti tömb elemei :
   65  173  -42    0   -5

A T tulajdoságnak megfelelö tomb elemei :
   65    0   -5

A T tulajdonságnak nem megfelelö tömb elemei :
  173  -42
*/
```

III. rész Feladatgyűjtemény

SZÁMÁBRÁZOLÁSI ANOMÁLIÁK

1. Ismertesse a túlcsordulási anomáliát. Milyen adattípus használatánál fordulhat ez elő, mi az oka, és mi a kiküszöbölésének módja? Mutasson egy algoritmust a problémára és a megoldására.
2. Ismertesse az alulcsordulási anomáliát. Milyen adattípus használatánál fordulhat ez elő, mi az oka, és mi a kiküszöbölésének módja? Mutasson egy algoritmust a problémára és a megoldására.

PROGRAMOZÁSELMÉLETI ALAPFOGALMAK

1. Ismertesse az alábbi fogalmakat: algoritmus, folyamatábra, program, utasítás. Definiálja a konstans, a típus, és a változó fogalmát.
2. Ismertesse a lokális azonosító, és az értékparaméter fogalmát, valamint adjon rájuk példát. Mikor használunk lokális azonosítókat, és értékparamétert?
3. Ismertesse a globális azonosító, és a címparaméter fogalmát, valamint adjon rájuk példát. Mikor használunk globális azonosítókat, és címparamétert?

CIKLUSOK

1. Írja át az alábbi C++ nyelvű algoritmusrészletet `while` és `do-while` ciklussá. Milyen típusú lehet az `index` és az `A` tömb? Hogyan változnak az `A` tömb elemei a ciklus lefutása után?

```
for (index = 0; index<10; index++)
  A[index] = A[index]/A[0];
```

2. Írja át az alábbi C++ nyelvű algoritmusrészletet `while` és `do-while` ciklussá. Milyen típusú lehet az `index` és az `A` tömb? Hogyan változnak az `A` tömb elemei a ciklus lefutása után?

```
for (index = 9; index >=0; index--)
  A[index] = A[index] / A[9];
```

3. Irja át az alábbi ciklust while ciklussá, valamint adja meg e ciklus futásához szükséges deklarációkat:

```
A = 2.5;
index = -6;
do { A = A * index;
     index = index + 1;
   } while (index < 0);
```

   A programrészlet lefutása után az `A` változó értéke pozitív lesz, vagy negatív?

4. Irja át az alábbi ciklust for ciklussá, valamint adja meg e ciklus futásához szükséges deklarációkat:

```
A = 3;
index = 6;
do { A = A * index;
     index = index - 1;
   } while (index > 0);
```

   A programrészlet lefutása után az A változó értéke páros lesz, vagy páratlan?

EGYSZERŰ PROGRAMOZÁSI FELADATOK

1. Irjon C++ nyelvű programot, mely a felhasználó által megadott számú, és a felhasználótól bekért float típusú számokat összeadja, és az eredményüket a képernyőre kiírja.
2. Írjon C++ nyelvű programot, mely a klaviatúráról addig olvas be számokat, míg azok összege kisebb 1000-nél. Ekkor írja ki a beolvasott számok összegét és számát. (A beolvasott számok között egész, tört, pozitív és negatív egyaránt lehet!)
3. Írjon C++ nyelvű programot, mely megállapítja, hogy a felhasználó által megadott két egész szám által határolt tartományban van-e olyan egész, mely 19-el osztható. Ha igen, akkor ezek közül egyet kiír a képernyőre, ha nem, akkor pedig azt a szöveget, hogy "Nincs köztük megfelelő".
4. Írjon programot C++ nyelven, mely megállapítja és a képernyőre kiírja, hogy valamely, a felhasználó által megadott két szám közötti számtartományban hány darab olyan egész szám van, mely egy, szintén a felhasználó által megadott egész számmal osztható. (A felhasználó negatív számokat is megadhat!)
5. Írjon C++ nyelvű programot, mely megkeresi az 1000 és 10000 közötti tartományban a felhasználó által megadott két egész szám mindegyikével osztható legkisebb egész számot. Ha ilyen létezik, akkor ezt kiírja a képernyőre, ha nem, akkor pedig a "Nincs köztük megfelelő" szöveget.
6. Írjon programot C++ nyelven, mely a felhasználótól addig kér be új számot, míg a bevitt szám nem lépi át az első bevitt szám ±10-szeresének értékét. Ekkor írja ki a képernyőre a bevitt számok darabszámát. (A számok egészek, törtek egyaránt lehetnek!) **FONTOS!** A megoldáshoz **NE** használjon tömböt!
7. Írjon programot C++ nyelven, mely a felhasználó által megadott két pozitív egész számról eldönti, hogy relatív prímek-e. (Két pozitív egész szám relatív prím, ha nincs 1-től különböző közös osztójuk.)
8. Irjon C++ nyelvű programot, mely a felhasználó által megadott számértékek közötti legnagyobb eltérés mértékét kiírja a képernyőre. Az adatbevitel addig történik, amíg a felhasználó azt folytatni kívánja.
9. Irjon C++ nyelvű programot, mely a felhasználó által megadott lebegőpontos számok közül a legkisebb és a legnagyobb abszolút értékűt kiírja a képernyőre. Az adatbevitel addig történik, amíg a felhasználó azt folytatni kívánja.

ÖSSZETETT PROGRAMOZÁSI FELADATOK

1. Írjon programot C++ nyelven, mely a felhasználótól addig kér be új számot, míg a bevitt számok átlaga nem kerül a [-1, +1] tartományba. Ekkor írja ki a képernyőre a bevitt számok darabszámát. **FONTOS!** A megoldáshoz **NE** használjon tömböt, alkalmazza a folyamatos átlag meghatározást!
2. Adja meg azt az algoritmust C++ nyelven, melynek segítségével a felhasználó által megadott számok mindegyike után kiírja az addigiak átlagát. A folyamatos-átlag képletét használja, és akkor fejeződjön be az algoritmus, ha a felhasználó már nem kíván további adatot bevinni.
3. Írjon programot C++ nyelven, mely egy, a felhasználó által megadott pozitív egész számig előállítja a prímszámokat, és azokat a képernyőre kiírja.
4. Írjon programot C++ nyelven, mely egy, a felhasználó által megadott pozitív egész szám faktoriálisát egy függvényben rekurzióval előállítja, majd az eredményt a képernyőre kiírja.
5. Adja meg a másodfokú egyenlet megoldásának algoritmusát C++ nyelven. A megoldás során teljeskörű diszkussziót végezzen, azaz bármely együttható lehessen zérus, és az algoritmus állítsa elő az esetleges komplex gyököket is.

6. Írjon programot C++ nyelven, mely kiírja a képernyőre a legkisebb, a felhasználó által megadott egész számmal osztható Fibonacci számot. (A vizsgálat ne terjedjen ki a 30000-nél nagyobb számokra.)
7. Írjon programot C++ nyelven, mely az alábbi sorozat N-edik tagját kiírja a képernyőre. (N értékét a felhasználó adja meg.)

   7.1. f: 0, 5, -10, 15, -20, ...

   7.2. g: 2, 2, 5, 11, 20, ...
8. Írjon programot C++ nyelven, mely meghatározza az alábbi sor összegét az N-ik tagig bezárólag, és azt kiírja a képernyőre. (N értékét a felhasználó adja meg.)

   8.1. h = 1 + 1/2 + 1/3 + ...

   8.2. k = 3 - 6 + 9 - ...
9. Írjon programot C++ nyelven, mely a felhasználó által megadott legfeljebb ötbetüs szó betüiből az összes lehetséges ismétlés nélküli permutációt előállítja, majd az egyes permutációkat a képernyőre egymás alá kiírja.

TÖMBKEZELÉSI FELADATOK

1. Írjon programot C++ nyelven, mely egy 3*4-es kétdimenziós, int típusú tömb elemeit feltölti a sor és oszlopszámuk szorzatával, majd e tömböt 3*4-es táblázatként a képernyőre kiírja.
2. Írjon programot C++ nyelven, mely egy 5*5-ös kétdimenziós, int típusú tömb elemeit feltölti a sor és oszlopszámuk összegével, majd e tömb főátlóban lévő elemeinek összegét a képernyőre kiírja.
3. Írjon programot C++ nyelven, mely egy, a felhasználó által feltöltött karaktertömbben szereplő elemek sorrendjét a tömbben megfordítja, majd ennek elemeit a képernyőre kiírja.
4. Írjon programot C++ nyelven mely egy, a felhasználó által feltöltött karakteres tömb tartalmát buborékos módon rendezi. A rendezés az ábécé szerint növekvő irányú legyen. A tömb eredeti és rendezett állapotát egyaránt írja ki a képernyőre olymódon, hogy a tömb elemei egy-egy szóközzel elválasztva egymás mellett legyenek.
5. Írja le C++ nyelven egy karakter típusú körforgó puffer használatának algoritmusát az alábbiak szerint: Először töltse fel a felhasználó a puffert, majd adja meg, hogy hányat kíván léptetni, majd azt, hogy jobbra-e, vagy balra. Az algoritmus írja ki a feltöltött puffer tartalmát, végezze el az előírt léptetést, végül a puffer új tartalmát is írja ki. (A körforgó puffer olyan tömb, melyben jobbra léptetés esetén az utolsó elem értékét az első, balra léptetés esetén az első elem értékét az utolsó veszi fel.)
6. Irjon programot C++ nyelven, melynek segítségével a felhasználó egy `eredeti` nevű, 20 elemű tömbbe egész számokat tud írni mindaddig amíg folytatni kívánja az adatbevitelt, de legfeljebb a tömb deklarált méretéig. Ezután a program a felhasználó által megadott egész számmal való oszthatóság (mint szétválogatási feltétel) szerint helyezze sorrendhelyesen egy `jo` és egy `rossz` nevű tömbbe az eredeti tömbelemeket aszerint, hogy a megadott szétválogatási feltételnek eleget tesznek-e, vagy nem. Végül a program írja ki a képernyőre mindhárom tömb elemeit fordított sorrendben olymódon, hogy egy tömb elemei 8-as mezőszélességben jobbra tömörítve egy sorban legyenek, a különböző tömbök elemei pedig egymás alatti sorban.

KARAKTERLÁNC-KEZELÉSI FELADATOK

1. Irjon C++ nyelvű programot, mely a felhasználó által megadott számú, és a felhasználótól bekért stringeket összefűzi, és a kapott stringet a képernyőre kiírja.

2. Irjon C++ nyelvű programot, mely a felhasználó által megadott vezeték és keresztneveket táblázatszerűen egymás alá írja, miután a felhasználó egy "üres" név megadásával befejezte az adatbevitelt.
3. Irjon C++ nyelvű programot, mely megállapítja, hogy a felhasználó által megadott két karaktersorozat közül az elsőt tartalmazza-e részsorozatként a második. Ha igen, akkor adja meg a tartalmazás kezdőpozícióját, ha nem, akkor írja ki a "Nem tartalmazza" üzenetet.
4. Irjon C++ nyelvű programot, mely meghatározza a felhasználó által megadott mondat szavainak számát, és ezt a képernyőre ki is írja. A program legyen felkészítve arra az esetre is, ha a felhasználó "üres" mondatot ad meg (amely csak szóközt tartalmaz, vagy még azt se).

STRUKTÚRA-KEZELÉSI FELADATOK

1. Irjon programot C++ nyelven, mely bekéri a felhasználó anyja nevét, a felhasználó magasságát (méterben) és nevét ebben a sorrendben, majd kiírja azokat a képernyőre.
2. Irjon programot C++ nyelven, mely definiál egy struktúra típust név, kor és cím adattagokkal, majd ennek felhasználásával bekéri a felhasználó életkorát, nevét és címét ebben a sorrendben. A program a struktúra feltöltését követően írja ki az adattagok értékét a képernyőre.
3. Írjon programot C++ nyelven mely egy, a felhasználó által feltöltött, struktúra elemű tömb tartalmát buborékos módon rendezi a struktúra első (numerikus) mezője szerint. A rendezés növekvő irányú legyen. A tömb eredeti és rendezett állapotát egyaránt írja ki a képernyőre olymódon, hogy a tömb elemeit alkotó struktúrák egyes adatmezői egy-egy szóközzel elválasztva egymás mellett legyenek. A struktúra szerkezete a következő legyen: azonosito (long int), nev (20 hosszú string), cím (30 hosszú string).

EGYSZERŰ FILE-KEZELÉSI FELADATOK

1. Írjon programot C++ nyelven, mely a felhasználó által megadott float típusú számokat szövegfileba helyezi. A szövegfile neve DATA.TXT, és helye az A: meghajtó. A file-ba csak azok a számok kerüljenek, melyek a (-1000)-től (+1000)-ig terjedő értéktartományban vannak, és az ábrázolási pontosságuk legyen 2 tizedes. Abban az esetben, ha a felhasználó által megadott szám értéke túllépi a fenti tartományt bármely irányban, akkor a file-ba a tartomány megfelelő szélső értéke kerüljön.
2. Írjon programot C++ nyelven, mely a felhasználó által megadott int típusú számokat az A: meghajtóra, egy ADAT.DAT nevű bináris file-ba helyezi. A file-ba csak azok a számok kerüljenek, melyek a (-5000)-től (+5000)-ig terjedő értéktartományban vannak. Abban az esetben, ha a felhasználó által megadott szám értéke túllépi a fenti tartományt bármely irányban, akkor a file-ba a tartomány megfelelő szélső értéke kerüljön.
3. Írjon programot C++ nyelven, mely létrehoz egy, a felhasználó által megadott nevű szövegfile-t, majd abba írja a felhasználó által megadott float típusú számokat. Az adatbevitel addig tartson, amíg a felhasználó egy "üres" (azaz számmegadás nélküli) bevitelt nem végez (azaz csak az Enter billentyűt üti le). A file-ba csak azok a számok kerüljenek, melyek a (-5000)-től (+5000)-ig terjedő értéktartományban vannak, és az ábrázolási pontosságuk legyen 2 tizedes. Abban az esetben, ha a felhasználó által megadott szám értéke túllépi a fenti tartományt bármely irányban, akkor a file-ba a tartomány megfelelő szélső értéke kerüljön.
4. Írjon programot C++ nyelven, mely létrehoz egy, a felhasználó által megadott nevű bináris file-t, majd abba írja a felhasználó által megadott egész számokat. Az adatbevitel addig tartson, amíg a felhasználó egy "üres" (azaz számmegadás nélküli) bevitelt nem végez (azaz csak az Enter billentyűt üti le). A file-ba csak azok a számok kerüljenek, melyek a (-1000)-től (+1000)-ig terjedő értéktartományban vannak. Abban az esetben, ha a felhasználó által megadott szám értéke túllépi a fenti tartományt bármely irányban, akkor a file-ba a tartomány megfelelő szélső értéke kerüljön.

5. Irjon programot C++ nyelven, melynek segítségével a felhasználó egy A:\ADAT.DAT nevű szöveges file tartalmához addig írhat további sorokat, míg egy „üres” Enter-t le nem üt.
   A programban a file megnyitása előtt ellenőrizze a file létezését, és a program végén gondoskodjon a file lezárásáról.
6. Irjon programot C++ nyelven, melynek segítségével a felhasználó egy A:\ADATBE.DAT nevű szöveges file-t létrehoz, feltölt int típusú adatokkal egy-adat - egy-sor módon addig, amíg egy „üres” Enter-t le nem üt. Ezután e file-elemeinek négyzeteiből sorrendhelyesen létrehoz egy vele egyező szerkezetű A:\ADATKI.DAT nevű szöveges file-t. Végül ennek tartalmát írja ki a képernyőre.
7. Irjon C++ nyelvű programot, mely megállapítja, hogy egy, a felhasználó által megadott nevű szöveges file tartalmaz-e egy, szintén a felhasználó által megadott szót (sor végén nincs elválasztás). Ha igen, akkor adja meg a tartalmazás kezdőpozícióját (a találati sor számát, valamint a sorbeli kezdő karakter sorszámát), ha nem, akkor írja ki a "Nem tartalmazza" üzenetet.

ÖSSZETETT FILE-KEZELÉSI FELADATOK

1. Készítsen hallgatói nyilvántartást C++ nyelven, mely a felhasználótól bekéri a hallgatok adatait, (nevet, tankörszámot és tanulmányi átlagot tartalmazó struktúrában), majd azokat egy, a felhasználó által megadott nevű file-ba menti. Az adatbevitel mindaddig tart, amíg a felhasználó azt folytatni nem kívánja. Végül a program listázza ki a file tartalmát a képernyőre.
2. Irjon programot C++ nyelven, mely az int típusú input file adatainak négyzeteit az output file-ba másolja. Ha a négyzetszám nagyobb, mint a legnagyobb ábrázolható int típusú érték, akkor a file-ba ez a legnagyobb int érték kerüljön. Az input és output file-ok neveit a program a felhasználótól kérje be.
3. Irjon programot C++ nyelven, mely egy long int típusú bináris file-t feltölt az első N darab Fibonacci számmal, e számokat a képernyőre kiíratja a generálás sorrendjében, majd a létrehozott file tartalmát beolvassa a file végétől az elejéig, és ebben a sorrendben újra kiírja a képernyőre. A file nevét, és az N szám értékét a felhasználó adja meg. A file-ból való kiolvasásnál NE használja ki a file méretének ismeretét!
4. Irjon programot C++ nyelven, mely egy long int típusú bináris file-t feltölt az első N pozitív egész szám faktoriálisaival, e számokat a képernyőre kiíratja a generálás sorrendjében, majd a létrehozott file tartalmát beolvassa a file végétől az elejéig, és ebben a sorrendben újra kiírja a képernyőre. A file nevét, és az N szám értékét a felhasználó adja meg. A file-ból való kiolvasásnál NE használja ki a file méretének ismeretét!
5. Készítsen hallgatói nyilvántartó programot C++ nyelven, mely egy menüből való választás alapján lehetőséget ad a felhasználónak arra, hogy újabb adattal bővítse a nyilvántartását, módosítsa, vagy törölje valamelyik hallgatóra vonatkozó adatokat, és hogy adott nevű hallgató adatait, illetve az összes hallgató nevét a képernyőn listázza. A menü tartalmazza a programból való kilépés lehetőségét is. Egy hallgató megkereséséhez legyen elegendő a név egy alstringjének (például a vezetéknév) megadása is. Kereséskor az első (a kereső feltételeknek megfelelő) hallgató adatait írja ki a képernyőre, és kérjen megerősítést (hogy valóban a keresett hallgatót sikerült-e megtalálnia). Amennyiben a válasz nemleges keressen tovább, és kérjen újra megerősítést mindaddig, amíg meg nem találja a keresett személyt, vagy véget nem ér az adatfile. Sikertelen keresés esetén írja ki a "Nem találtam a keresett személyt..." üzenetet.

Ajánlott irodalom:

TURBO C++ Programmer's Guide
[Borland International, 1990.]
TURBO C++ Library Reference
[Borland International, 1990.]
Benkő - Meskó - Tóth - Schuler: Programozás C és C++ nyelven. Grafikus programok.
[BME Mérnöktovábbképző Intézet, 5331 sz. jegyzet, 1995.]
Schuszter - Simán: C programozás Borland C++ 3.11 környezetben
[KKMF-1180 jegyzet, 1997.]
Benkő-Benkő-Tóth: Programozzunk C nyelven
[ComputerBooks, 2000.]
Benkő-Benkő-Poppe: Objektum-orientált programozás C++ nyelven
[ComputerBooks, 2000.]
Herbert Schildt: C/C++ Referenciakönyv
[Panem, 1998.]

IV. rész Példatár

(C és C++ nyelvű forrásprogramok)

Lásd a Labor01 mintapéldáit