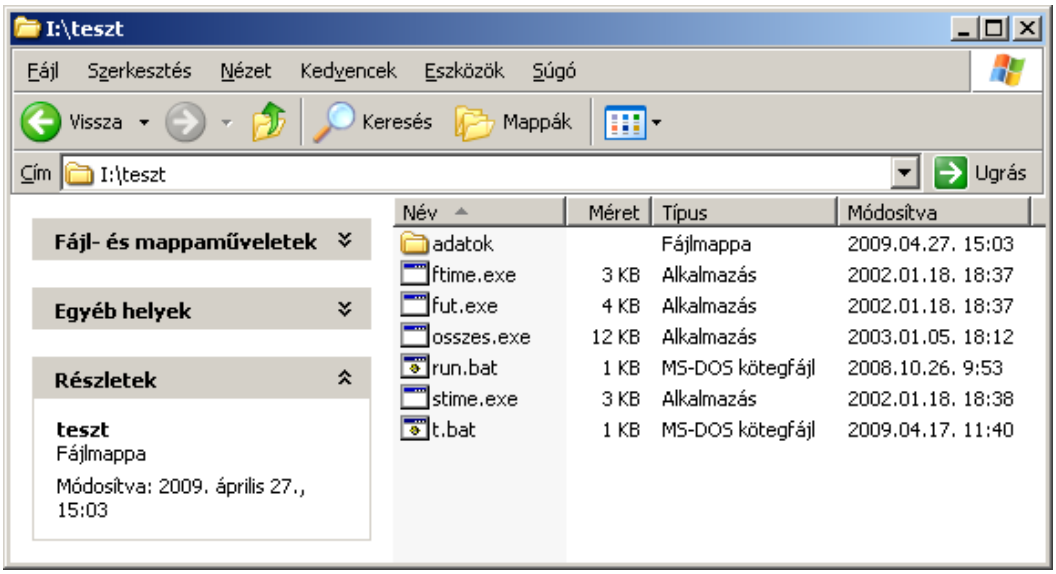
Géptermi zh-írás „forgatókönyve”

1. A feladat és a tesztelés körülményeinek a megértése

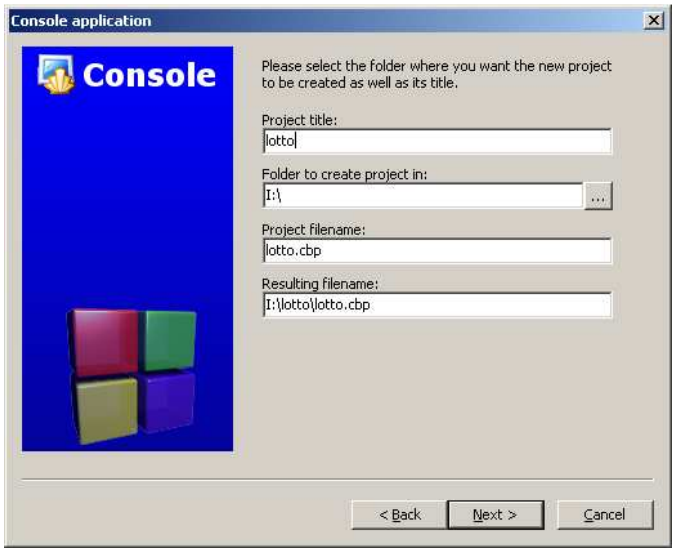
A feladat több részből áll. A megoldó program kötött szerkezetű fájlból kapja az adatokat, ezért azt „komolyan kell venni”! Az eredményt kötött szerkezetű fájlba kell írni, ezért ezt is tiszteletben kell tartani!
Az értékelés adott tesztesetek alapján fog történni, automatikusan. Egy tömörített formában letölthető tesztkörnyezet áll rendelkezésre, amelyet azon drive gyökeréből nyíló adott nevű könyvtárába kell kicsomagolni, amely drive-on lesz található a fejlesztendő program.

2. A tesztkörnyezet létrehozása

A kapott tömörített tesztkörnyezet-fájlt egy választott drive (legyen most „I:”) gyökeréből kiinduló (a feladat szövege által rögzített nevű) könyvtárba (legyen most „teszt”) kicsomagoljuk.



3. A fejlesztő környezet létrehozása



Az előbbi drive-on a Code::Blocks-szal létrehozzuk a fejlesztői környezetet:

1. A „Folder to create project in:” beviteli mezőbe (most!) az „I:” drive-ot írjuk;
2. A „Project title:” mezőbe (most!) a „lotto” név kerül;
3. A forrás első néhány sorába beírjuk a szerzőt azonosító adatokat, komment sorokba (*név+ETR-azonosító+drótposta cím+ csoportsorszám+gyakorlatvezető neve*)

4. Kódolás

Alább egy lehetséges megoldás látható:

```
//név: Szlávi Péter
//ETR-azonosító: SZPKAFT.ELTE
//drótposta-cím: szlavip@elte.hu
//…

#include <iostream>
#include <fstream>

using namespace std;

//az _fN nevű fájlból beolvassa a _t[_n] tömböt
int fajlbol_int_tombbe(string _fN, int &_n, int _t[], int _maxN);
//lényegi eljárások:
void A_elj(int _n, const int _t[], int &_nyDb);
void B_elj(int _n, const int _t[], int &_nyer);
void C_elj(int _n, const int _t[], int _maxN, int &_db, int _pech[]);
//kiírás fájlba:
void fajlbaIras(string _fN, int _nyDb, int _nyer, int _pDb, const int _pech[]);

int main()
{
    //Bemenet:
    const string befN="lotto.be";//bemeneti fájl neve
    const int maxN=100;
    int N;
    int tal[maxN];
    //Kimenet:
    const string kifN="lotto.ki";//kimeneti fájl neve
    int nyDb;//A)
    int nyer;//B)
    int pDb,pech[maxN];//C)

    if (fajlbol_int_tombbe(befN,N,tal,maxN)!=0)
    {
      cout << "Hibas fajl\n\n";
    }
      else
    {
      //a lényeg:
      A_elj(N,tal,nyDb);
      B_elj(N,tal,nyer);
      C_elj(N,tal,maxN,pDb,pech);
      fajlbaIras(kifN,nyDb,nyer,pDb,pech);
    }

    return 0;
}
```

```
//A) feladat:
void A_elj(int _n, const int _t[], int &_nyDb)
{
  _nyDb=0;
  for (int i=0;i<_n;++i)
  {
    if (_t[i]>1) ++_nyDb;
  }
}
```

```
//B) feladat:
void B_elj(int _n, const int _t[], int &_nyer)
{
  int i=0;
  while ((i<_n) && (_t[i]!=5)) ++i;
  if (i<_n) _nyer=i+1; else _nyer=-1;
}
```

```
//C) feladat:
void C_elj(int _n, const int _t[], int _maxN, int &_db, int _pech[])
{
  _db=0;
  for (int i=0;i<_n;++i)
  {
    if (_t[i]==0)
    {
      _pech[_db++]=i+1;
    }
  }
}
```

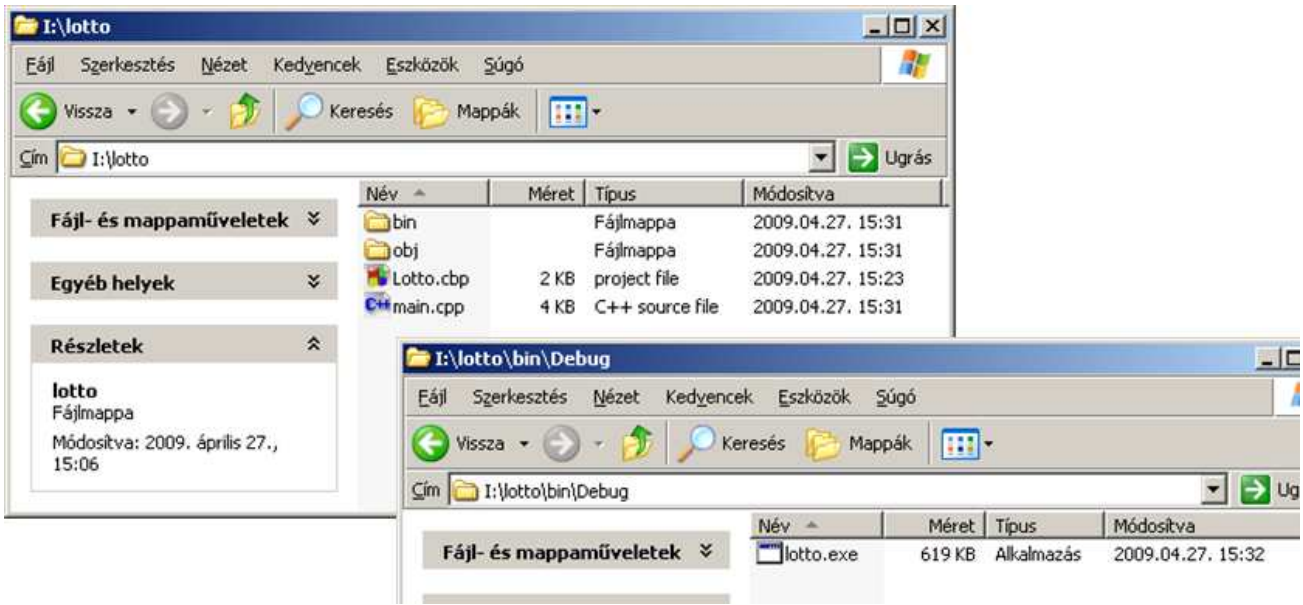
```
//az _fN nevű fájlból beolvassa a _t[_n] tömböt:
int fajlbol_int_tombbe(string _fN, int &_n, int _t[], int _maxN)
/*
  _fN fájlból feltölti a _t[n] tömböt egészekkel;
  elvárások: LÉTEZŐFÁJL(fN) ÉS 0<=_n<=_maxN ÉS FÁJL_SORAI_SZÁMA(_fN)=_n+1
  Uf: NEM LÉTEZŐFÁJL(_fN)          => eredmény=1 ÉS
      NEM 0<=_n<=_maxN             => eredmény=2 ÉS
      FÁJL_SORAI_SZÁMA(_fN)<>_n+1 => eredmény=3 ÉS
      hibátlanFájl(_fN)            => eredmény=0
*/
{
    int hibakod=0;//nincs hiba
    ifstream iF(_fN.c_str());
    if (!iF.is_open())
    {
      hibakod=1;//nem létező fájl
    }
      else
    {
      iF >> _n;
      string tmp; getline(iF,tmp,'\n');//az 1. sor maradékát eldobjuk
      if (_n>_maxN)
      {
         hibakod=2;//túl sok adat
      }
```

```
        else
      {
        int i=0;
        while (!iF.eof() && i<=_maxN)
        {
           iF >> _t[i++];
           getline(iF,tmp,'\n');//az 1. sor maradékát eldobjuk
        }
        if (i!=_n)
        {
           hibakod=3;//inkonzisztens fájl
        }
      }
    };
    iF.close();
    return hibakod;
}
```

```
//kiírás fájlba:
void fajlbaIras(string _fN, int _nyDb, int _nyer, int _pDb, const int _pech[])
{
      ofstream oF(_fN.c_str());
      oF << _nyDb << "\n";
      oF << _nyer << "\n";
      oF << _pDb << " ";
      for (int i=0;i<_pDb;++i)
      {
        oF << _pech[i] << " ";
      }
      oF.close();
}
```

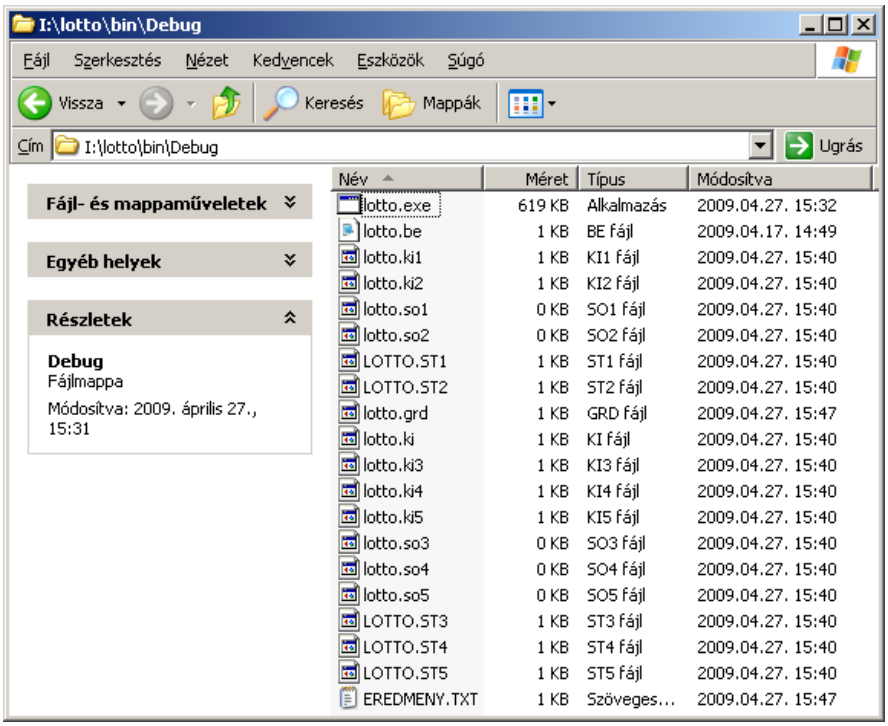
5. A helyesség ellenőrzése

A fejlesztői környezet finomabb szerkezete a fentiek után:

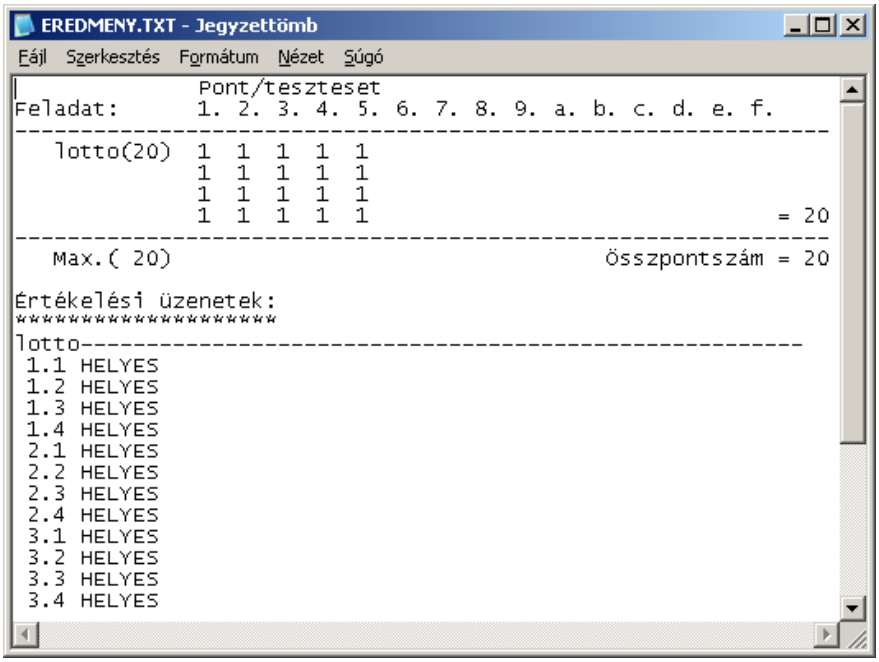


(Most!) Az „I:\lotto\bin\Debug” könyvtárba belépve pl. TotalCommander-rel végrehajtatjuk az „I:\teszt\” könyvtárban található „t.bat” batch-fájlt: „I:\teszt\t”.

Ugyanezt megtehetjük a „parancsablak” vagy „konzolablak” (Start → Futtatás…: cmd) megnyitása után, a kellő parancsok kiadásával: „I:” + „cd lotto\bin\Debug\” + „\teszt\t”.
Ha tesztelés sikeres volt formai szempontból, akkor számos fájl keletkezett ebben a könyvtárban:



Minket az „EREDMENY.TXT” fájl érdekel, ui. ebből derül ki, hogy mennyire volt helyes a megoldó programunk. A legsikeresebb esetben effélét láthatjuk ebben a fájlban:



6. A munkánk gyümölcsének feltöltése

A feladatlap tartalmazni fogja azt a helyet, ahova „fogd és vidd” módszerrel be kell húzni az addigra már **tömörített** és **EHA-kód**unkról elnevezett fejlesztői környezetet, azaz az **„I:\lotto\” teljes könyvtárat** (és nem az „I:\teszt\” könyvtárat).