Ebben a fejezetben olyan feladatokat tűzünk ki, amelyek megoldásához egy kezdetleges „ablakozó” felületet biztosító programkönyvtárat fogunk használni.[1] Először áttekintjük ennek a könyvtárnak a szolgáltatásait, majd ezután térünk rá a feladatok megoldására.

Egy ablakozó könyvtár

- Billentyűzet kezelés
  - Leütött billentyűt beolvasó függvény
  - Billentyű konstansok
- Tároló osztályok
  - Egy őselemosztály
  - Általános tároló osztály
  - Rendezett tároló osztály
- Ablakozó osztályok
  - Általános ablak osztály
  - Keretes ablak osztály
  - Fejléces-keretes ablak osztály
  - Adat beolvasó ablak osztály
  - Görgető ablak osztály
  - Fájl tartalmát görgető ablak osztály
  - Színkonstansok

Billentyűzet kezelés

Billentyű konstansok és a leütött billentyűt beolvasó függvény

```
namespace Key{
   const int  esc      = ...;
   const int  f1       = ...;
   const int  f2       = ...;
   ...
   const int  ins      = ...;
   const int  tab      = ...;

   const int  up       = ...;
   const int  down     = ...;

   int GetKey(void);
}
```

[1] A feladatok és a programkönyvtár elvi váza a Gábor Dénes Főiskola Objektum orientált programozás tantárgyának segédanyagaiból származik.

Tároló osztályok

Egy őselemosztály

```
class TItem {
public:
   TItem() {};
   virtual ~TItem() {};
   TItem* Next() { return FNext; };
   ...
private:
   TItem* FNext;
friend class TList;
friend class TOrderedList;
};
```

Általános tároló osztály

```
typedef void (*TItemProcedure)(TItem* item);

class TList : public TItem {
public:
   TList();

   virtual void Insert(TItem* item);
   bool Exists(TItem* item);
   void Delete(TItem* item);
   void DeleteAll();

   bool  Empty();
   Item* First();
   Item* RNext(TItem* item); // ciklikusan
   void  Each(TItemProcedure todo);
   ...
   virtual ~TList();        // meghívja: DeleteAll()

};
```

Rendezett tároló osztály

```
class TOrderedList: public TList{
public:
   TOrderedList() {};

   void Insert(TItem* item); // meghívja: Ordered()
   virtual bool Ordered(TItem* item1, TItem* item2) = 0;
};
```

Ablakozó osztályok

Színkonstansok

```
namespace Color{
   const int white = ...;
   const int black = ...;
   const int gray  = ...;
   const int cyan  = ...;
   ...
}
```

Általános ablak osztály

```
class TWindow : public TItem{
public:
   TWindow(int x, int y,
           int w, int h, int bc, int pc);
   virtual ~TWindow();
   ...
   virtual void Show();
   void Hide();
   void SetColor(int bc, int pc);
protected:
   int  xcoord, ycoord, width, height;
   int  backcolor, pencolor;
   int  cursorx, cursory;
};
```

Keretes és fejléces-keretes ablak osztály

```
class TFramedWindow: public TWindow{
   ...
   void SetFrame(...);
};

class TTitledWindow: public TFramedWindow{
public:
   TTitledWindow(string str, int x, int y,
           int w, int h, int bc, int pc);
   ...
   virtual string GetTitle();
   virtual void   SetTitle(string str);
};
```

Adat beolvasó ablak osztály

```
class TReadWindow: public TTitledWindow{
public:
   TReadWindow(string str, int x, int y,
           int w, int h, int bc, int pc);
   ...
   virtual char   ReadChar();
   virtual double ReadNum();
   virtual string ReadString();
};
```

Görgető ablak osztály

```
class TScrollWindow: public TTitledWindow{
public:
   TScrollWindow(string str, int x, int y,
          int w, int h, int bc, int pc);
   ...
   virtual void Show();          // FLines[] sorai
   virtual void ScrollUp();
   virtual void ScrollDown();
   void Insert(string& str);
protected:
   int FLinesNum;
   int Flines[size];        // size=200 konstans
   ...
};
```

Fájl tartalmát görgető ablak osztály

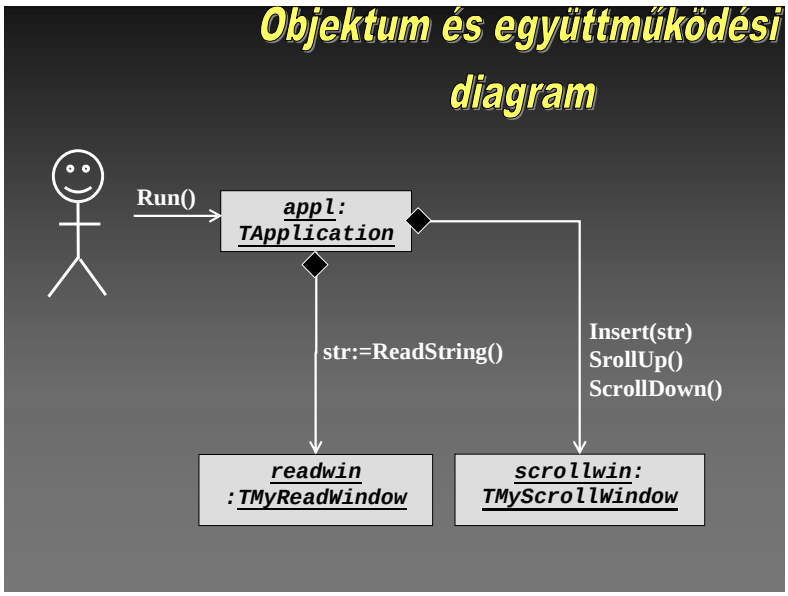
```
class TScrollFileWindow: public TScrollWindow {
public:
   TScrollFileWindow(string str, int x, int y,
            int w, int h, int bc, int pc);
   ...
private:
   string FFileName;
};
```

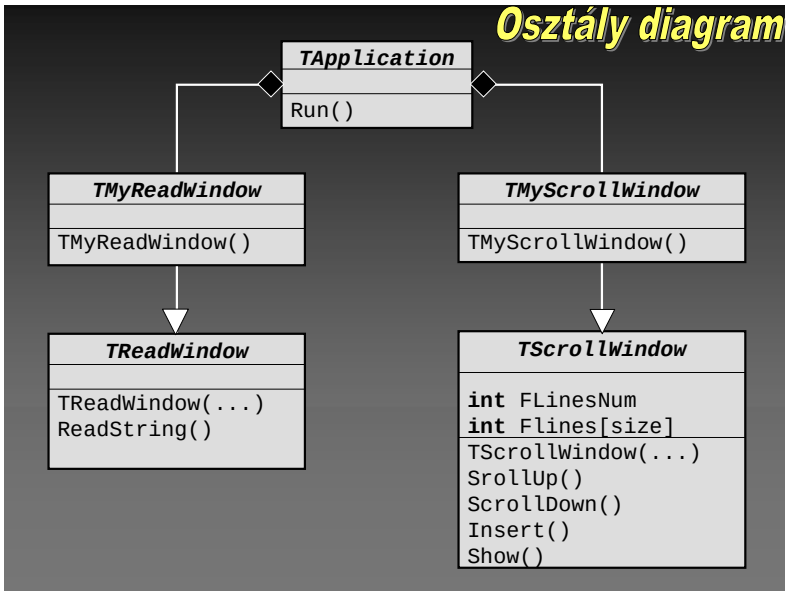
1. Feladat

A képernyő felső felét lefedi egy szürke-kék színű keretes görgetőablak, mely kezdetben üres. Ezt az ablakot paraméter nélkül inicializáljuk.
Ezután a program a következő billentyűkre reagál:

- *Ins gomb: Megjelenik egy kék-fekete, 'Név' című ablak a képernyő alsó felén, melyből egy nevet kérünk be. Beolvasás után az ablak eltűnik a képernyőről. A beolvasott név megjelenik a görgetőablakban, utolsó névként.*
- *Kurzor fel, illetve le: A görgetőablakban levő neveket fel, illetve fel lehet görgetni.*
- *Az Esc-re tűnjön el a görgető ablak, és kilépés.*

Megoldás

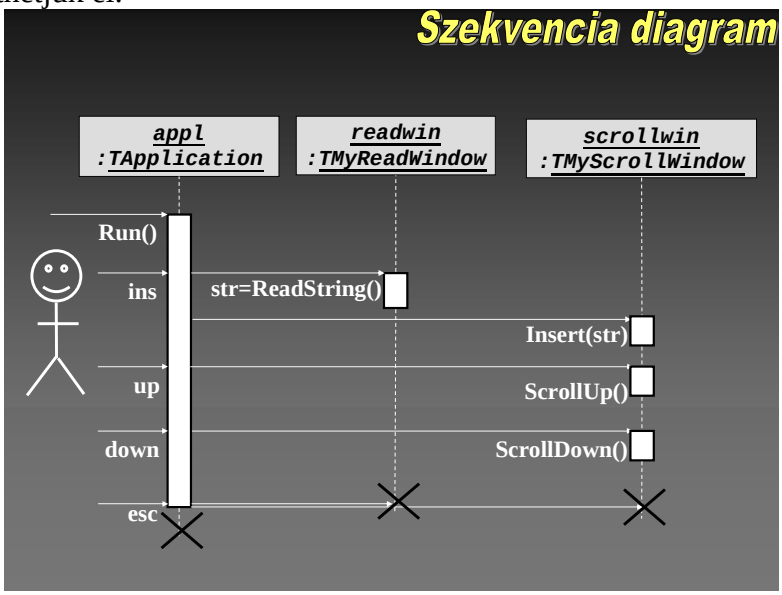




Megoldás C++-ban

A C++ programot egyetlen forrás fájlban helyezzük el. Itt írjuk le a `main` függvényt, és itt definiáljuk az osztály diagramnak megfelelő három saját osztályt.

Az egész alkalmazást vezérlő `Run()` metódust a feladat szekvencia diagramja alapján készíthetjük el.



Itt egy (billentyű) esemény-lekérdező ciklusra van szükségünk, amely négy különböző billentyű leütésére reagál.

```
void TApplication::Run()
{
    int key;
    for(;;){
        key = GetKey();
        switch (key){
        case ins : scrollwin.Insert(
                   readwin.ReadString());
                   break;
        case up  : scrollwin.ScrollUp();
                   break;
        case down: scrollwin.ScrollDown();
                   break;
        case esc : return;
        }
    }
}
```

Ezt a `Run()` függvényt az alkalmazás objektumon (`appl`) keresztül aktivizáljuk:

```
int main()
{
    TApplication appl;
    appl.Run();
}
```

Az alkalmazásunkat leíró `TApplication` osztály rejtett adattagjai az objektum diagramról olvashatók le: az alkalmazás egy saját beolvasó és egy saját görgető ablakból áll. Itt az alapértelmezés szerinti konstruktorra és destruktorra támaszkodunk.

```
class TApplication{
protected:
    TMyReadWindow   readwin;
    TMyScrollWindow scrollwin;
public:
    void Run();
};
```

Másképpen is implementálhatnánk a `TApplication` osztályt. Ekkor szükség van saját konstruktorra és destruktorra, és a `Run()` metódust értelemszerűen át kell majd írni (pl. `scrollwin.ScrollUp()` helyett `scrollwin->ScrollUp()`)

```
class TApplication{
protected:
    TMyReadWindow   *readwin;
    TMyScrollWindow *scrollwin;
public:
    TApplication(){
        readwin   = new TMyReadWindow;
        scrollwin = new TMyScrollWindow;
    }
    void Run();
    ~TApplication(){
        delete readwin;
        delete scrollwin;
    }
};
```

A saját beolvasó (`TMyReadWindow`) és görgető (`TMyScrollWindow`) ablak osztályokat specializációval származtatjuk úgy, hogy paraméter nélküli konstruktorokat definiálunk hozzájuk.

```
#include ”ukey.h”
#include ”uwin.h”
using namespace Key;
using namespace Color;

class TMyScrollWindow: public TScrollWindow{
public:
  TMyScrollWindow():TScrollWindow(”Görgő”,1,1,80,5,gray,black){}
};

class TMyReadWindow: public TReadWindow{
public:
  TMyReadWindow():TReadWindow(”Olvasó”,1,12,80,5,cyan,black){}
};
```

2. Feladat

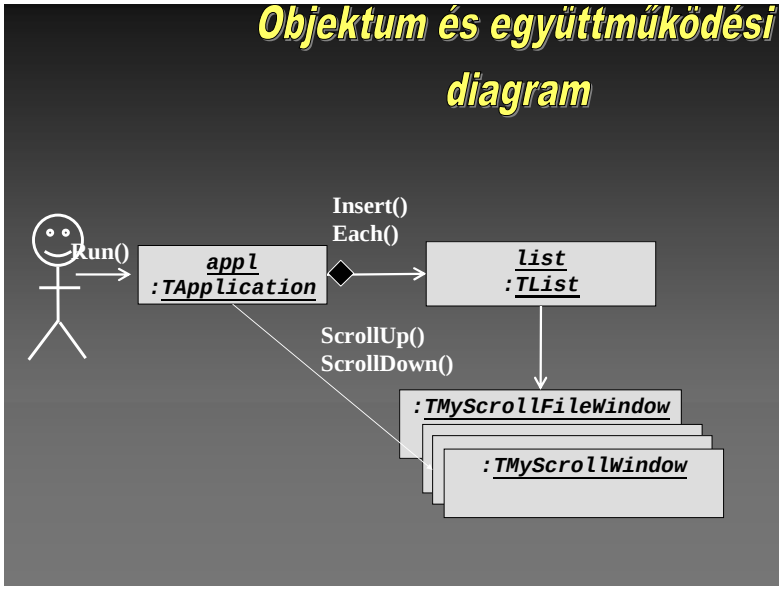
*A képernyőn két féle, 20*5 méretű ablak található:*

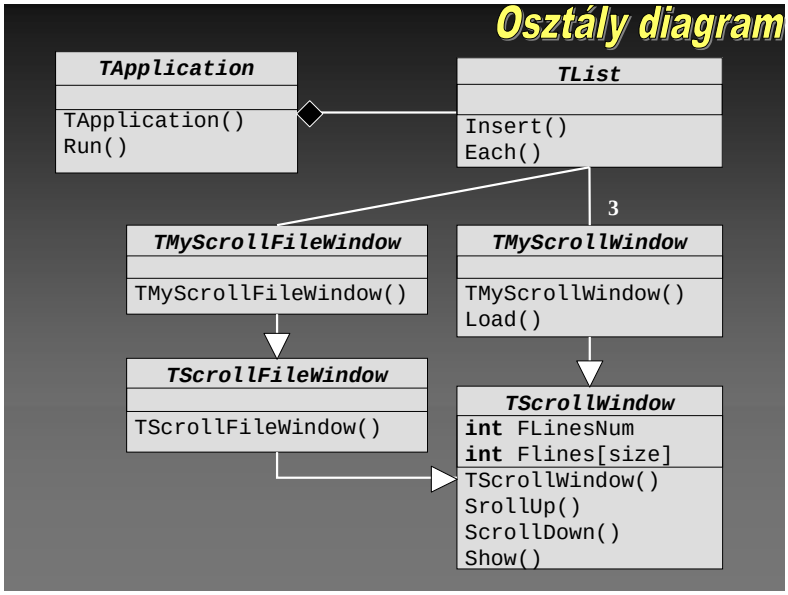
- *az egyikben egy megadott állományt lehet fel-le görgetni.*
- *a másikban a következő nevek görgethetők: Aladár, Béla, Gyula, Kázmér, Jenő.*

A program indulásakor jelenjen meg egy ablak az első fajtából, három pedig a második fajtából.

- *A fel-le kurzorvezérlő billentyűkkel az ablakokban levő szövegeket egyszerre lehessen görgetni!*
- *Az Esc-re tűnjenek el az ablakok, és kilépés.*

Megoldás





Megoldás C++-ban

A C++ kódot az első feladat mintájára könnyen elkészíthetjük. Végezzük most a kódolás felülről lefele (kívülről befele) haladva az úgynevezett top-down módszerrel.

Az alkalmazást egy alkalmazás objektummal vezéreljük.

```
#include ”ukey.h”
#include ”uwin.h”
#include ”ulist.h”
using namespace Key;
using namespace Color;

...

int main()
{
    TApplication appl;
    appl.Run();
}
```

Az alkalmazás egyetlen tárolót tartalmaz,

```
class TApplication{
protected:
    TList list;
public:
    TApplication();
    void Run();
};
```

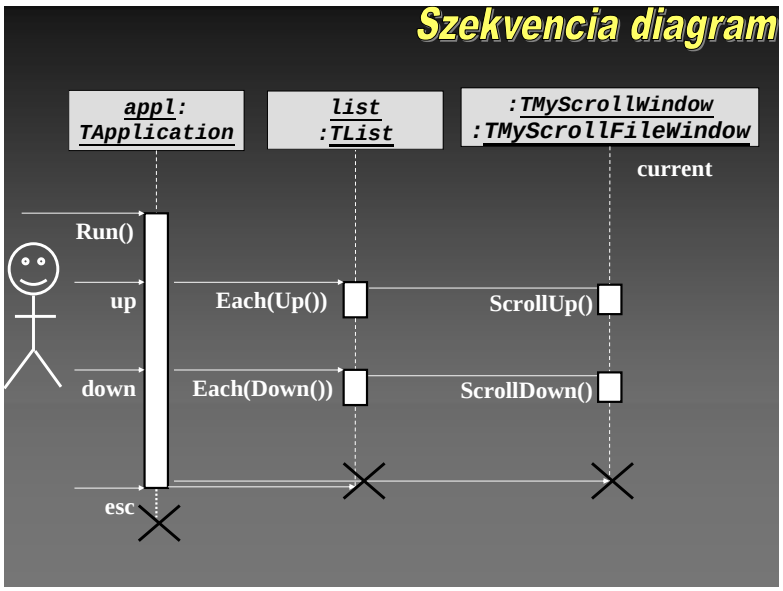
amelybe egy speciális állomány görgető ablakot és három speciális görgető ablakot fűzünk be. Ennek elvégzése az alkalmazás konstruktorának feladata.

```
TApllication::TApplication()
{
    list.Insert(new TMyScrollFileWindow
        (”input.txt”,1,1,red,white));

    for(int i = 1; i<=3; i++){
    list.Insert(new TMyScrollWindow
        (1,i*6+1,blue,white));
    }
}
```

A speciális ablakobjektumok osztályait az osztály diagram szerint az ablakozó könyvtár megfelelő osztályaiból kell származtatni úgy, hogy saját konstruktorokat készítünk hozzájuk. A saját görgető ablakhoz egy olyan védett (`Load()`) metódust definiálunk, amely az előre rögzített tartalommal tölti fel az ablakot.

```
class TMyScrollFileWindow: public TScrollFileWindow{
public:
    TMyScrollFileWindow(string str,int x,int y,int bc,int pc)
    :TScrollFileWindow(str,x,y,80,6,bc,pc){
        Show();
    }
};

class TMyScrollWindow: public TScrollWindow{
public:
    TMyScrollWindow(int x,int y,int bc,int pc)
    :TScrollWindow("",x,y,80,6,bc,pc){
        Load();
        Show();
    }
protected:
    void Load();
};
void TMyScrollWindow::Load()
{
    FLinesNum = 5;
    Flines[0] = "Aladár";
    Flines[1] = "Béla";
    Flines[2] = "Gyula";
    Flines[3] = "Kázmér";
    Flines[4] = "Jenő";
}
```

Az alkalmazás `Run()` metódusa a szekvencia diagramnak felel meg.



```
void TApplication::Run()
{
    int key;
    for(;;){
        key = GetKey();
        switch (key){
        case up  : list.Each(Up);
                   break;
        case down: list.Each(Down);
                   break;
        case esc : return;
        }
    }
}

void Up(TItem* w)
{
    w->ScrollUp();
}

void Down(TItem* w)
{
    w->ScrollDown();
}
```

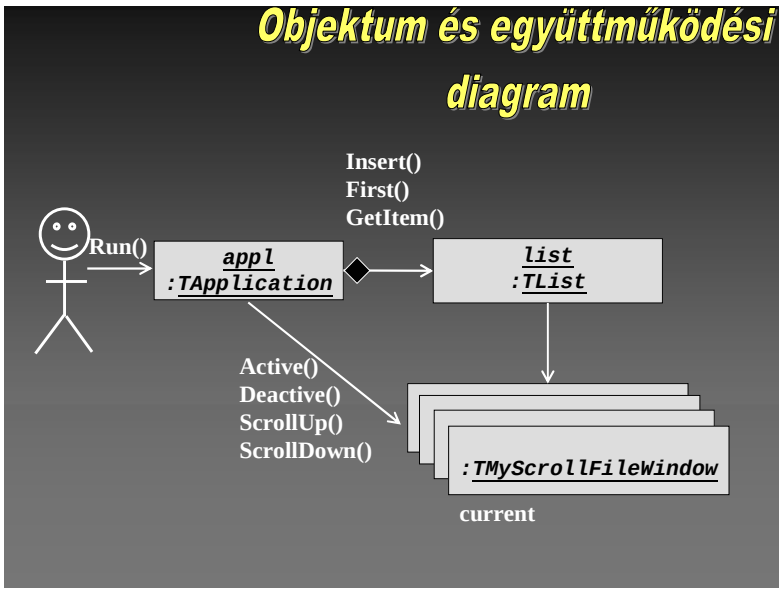
A programnak ebben a részében érdekes formájával találkozhatunk a dinamikus kötésnek. A `TList` tároló Each() metódusának csak olyan függvénypointert lehet átadni, amely `TItem` típusú objektumokkal dolgozik. Ezért definiáltuk az `Up()` és `Down()` függvényeket. Fordítási időben azonban a `w->ScrollUp()` hívásnak nincs értelme, hiszen a `TItem` típusú `w`-re nincs értelmezve a `ScrollUp()` metódus, azt csak a görgető ablakoknál vezettük be. Futás közben az `Each()` metódus a `list` tárolóban levő objektumokra (görgető ablakokra) hívja meg az `Up()` illetve és `Down()` függvényeket, így a `w` helyébe mindig a `TItem`-ből származtatott görgető ablakobjektum címe kerül. (A származtatási lánc miatt ez az értékadás szabályos.) Mivel a `ScrollUp()` (és a `ScrollDown()` is) virtuális metódus, és ezért a `w->ScrollUp()` hívás kiértékelése futási időben történik, így az az aktuális görgető ablakobjektumra vonatkozik.

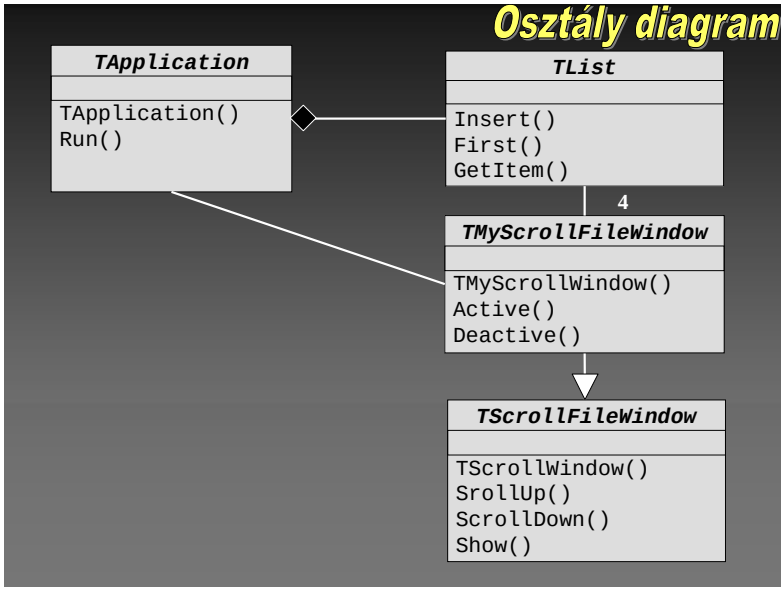
3. Feladat

Jelenítsen meg a képernyőn négy fájl-görgető ablakot egymás alatt. Bal felső sarkaik rendre az 1,1; 1,7; 1,13; 1,19. Szélességük egységesen 80, magasságuk: 6. Kezdetben az első görgető ablak az aktuális. Az aktuális ablak háttérszíne mindig piros, a többi ablak háttérszíne az inicializáláskor megadott háttérszín. Az 1. és a 3. ablakban a c:\autoexec.bat fájlt, a 2. és a 4. ablakban a c:\config.sys fájl jelenjen meg.
A program vezérlése:

- *A fel-le kurzorvezérlő billentyűkkel az aktuális ablak tartalmát lehessen görgetni*
- *Az F1, F2, F3, F4 billentyűkkel aktivizálhassuk az 1., 2., 3. vagy a 4. Ablakot.*
- *Az Esc-re tűnjenek el az ablakok, és kilépés.*

Megoldás





Megoldás C++-ban

Válasszuk most az alulról felfelé (belülről kifelé) haladó úgynevezett bottom-up módszert a kódoláshoz.
Most is egyetlen fájlban helyezzük el a kódot. Először az osztály diagramnak megfelelő osztályokat implementáljuk.

```
#include ”ukey.h”
#include ”uwin.h”
#include ”ulist.h”
using namespace Key;
using namespace Color;

class TMyScrollFileWindow: public TScrollFileWindow{
public:
    TMyScrollFileWindow(string str,int x,int y,int bc,int pc)
    :TScrollFileWindow(str,x,y,80,6,bc,pc){
        Show();
    }
    virtual void Active(){
        SetColor(red,white);
        Show();
    };
    virtual void Deactive(){
        SetColor(black,white);
        Show();
    };
};
```

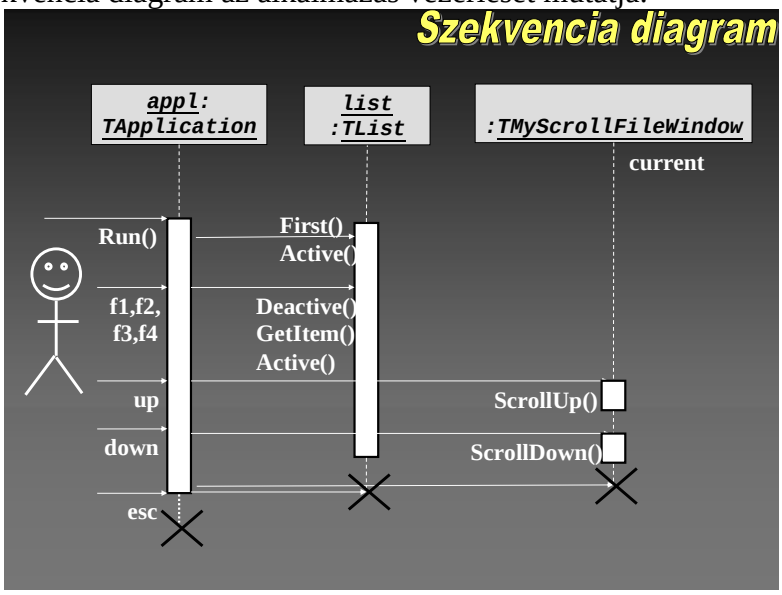
Az alkalmazás osztály definiálásánál az objektum diagramot használjuk.

```
class TApplication{
protected:
    TList list;
    TItem *current;       // TMyScrollWindow *current
public:
    TApplication();
    void Run();
};
```

Az alkalmazás osztály konstruktora létrehozza az állomány görgrtő ablakokat tartalmazó `list` tárolót, és beállítja az aktuális ablak `current` pointerét.

```
TApllication::TApplication()
{
    const string filename[2] =
        (”c:\config.sys”, ”c:\autoexec.bat”);
    for(int i = 0; i<4; i++) {
        list.Insert(new TMyScrollFileWindow(
            filename[i%2],i*6+1,1,black,white));
    }
    current = list.First();
    current->Active();
}
```

A szekvencia diagram az alkalmazás vezérlését mutatja.




```
void TApplication::Run()
{
    int key;
    for(;;){
        key = GetKey();
        switch (key){
        case f1  : case f2 : case f3 : case f4 :
                   current->Deactive();
                   current = list.GetItem(key-f1+1);
                   current->Active();
                   break;
        case up  : scrollwin.ScrollUp();
                   break;
        case down: scrollwin.ScrollDown();
                   break;
        case esc : return;
        }
    }
}
```

Végül a `main()` függvény.

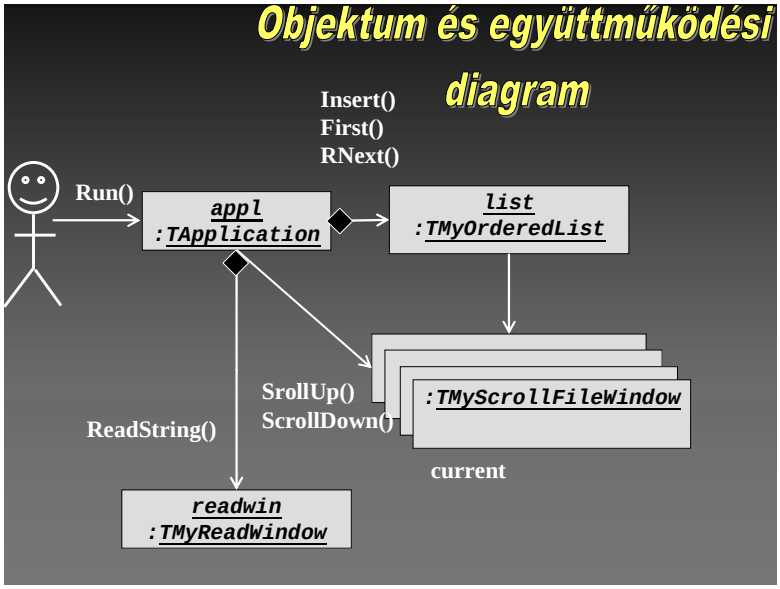
```
int main()
{
    TApplication appl;
    appl.Run();
}
```

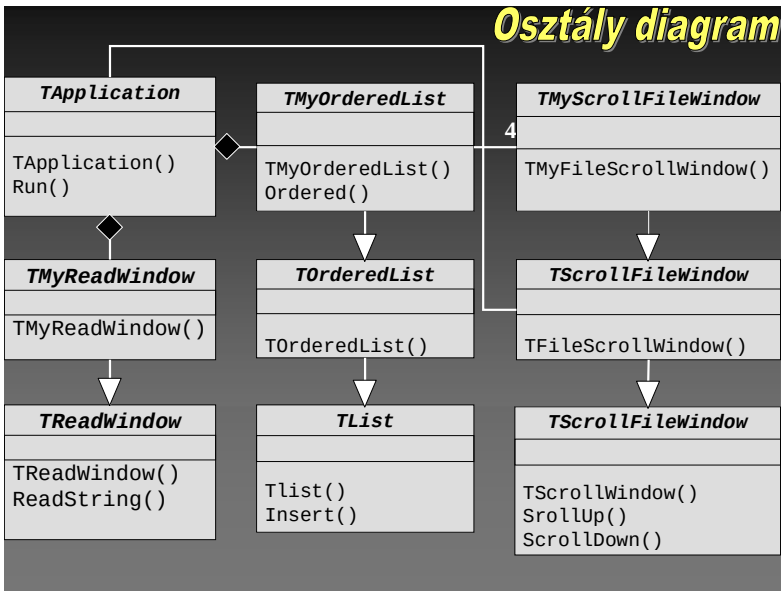
4. Feladat

*A képernyő alján jelenjen meg egy beolvasó ablak "Állomány neve:" címmel, amelyből egymás után kérjünk be négy darab állománynevet. (Tegyük fel, hogy az állományok léteznek.) Minden beolvasás után tegyünk a képernyőre egy-egy ablakot (mindig az előző ablak alá), mely a beolvasott nevű állományt tudja görgetni. Az állományablakok középen, egymás alatt helyezkednek el, kék alapon fehérek, és 40*5-ös méretűek. Az állományok beolvasása végén a beolvasó ablakot tüntessük el!*
Az ablakok bejárási sorrendje az állománynevek abc sorrendje legyen. Kezdetben az aktuális állományablak a névsorban első.

- *A Tab billentyűvel lehessen lépegetni az ablakok között.*
- *A fel-le billentyűre görgethessük az aktuális állományt.*
- *Az Esc-re tüntessük el az állományablakokat és kilépés.*

Megoldás





Megoldás C++-ban

Végezzük most a kódolást hibrid módon: haladjunk egyidejűleg szembe a felülről lefele és az alulról felfele implementálás módszerével.
A megoldó program szerkezete az előző megoldásokéval azonos.

```
#include ”ukey.h”
#include ”uwin.h”
#include ”ulist.h”
using namespace Key;
using namespace Color;

...

int main()
{
    TApplication appl;
    appl.Run();
}
```

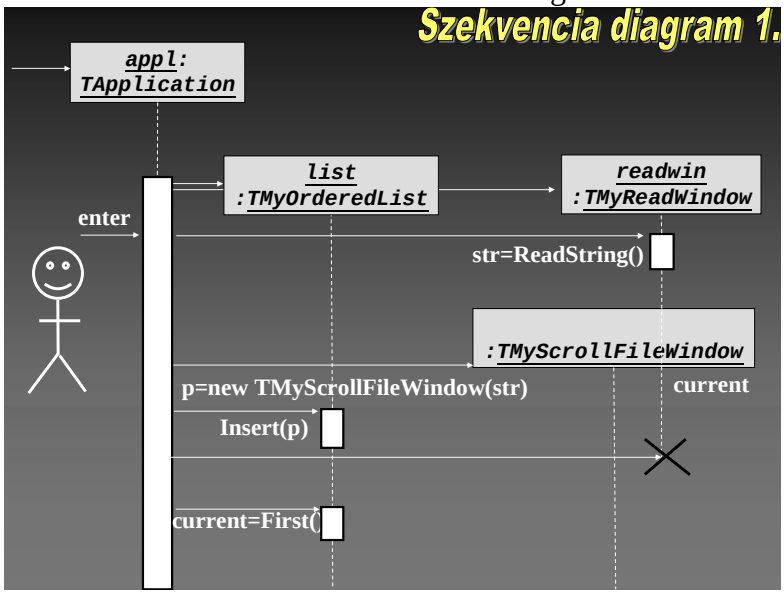
Az alkalmazás osztály definiálásánál támaszkodjunk az objektum diagramra.

```
class TApplication{
protected:
    TMyOrderesList list;
    TMyReadWindow  *readwin;
    TItem *current;       // TMyScrollWindow *current
public:
    TApplication();
    void Run();
};
```

Az osztály diagram alapján elkészíthetjük az osztályok definícióit. Először származtatással készítsük el a saját görgető, beolvasó ablak osztályunkat, és a rendezett tároló osztályt.

```
class TMyScrollFileWindow: public TScrollFileWindow{
public:
    TMyScrollFileWindow(string str,int x,int y)
    :TScrollFileWindow(str,x,y,40,5,gray,black){};
};

class TMyReadWindow: public TReadWindow{
public:
    TMyReadWindow()
    :TReadWindow(”Állománynév:”,1,21,40,5,cyan,black){};
};

class TMyOrderedList: public TOrderedList{
public:
    bool Ordered(TItem *w1, TItem *w2)
    {
        return w1->GetTitle() < w2->GetTitle();
    }
};
```

Az alkalmazás konstruktorához a szekvencia diagramot vesszük alapul.

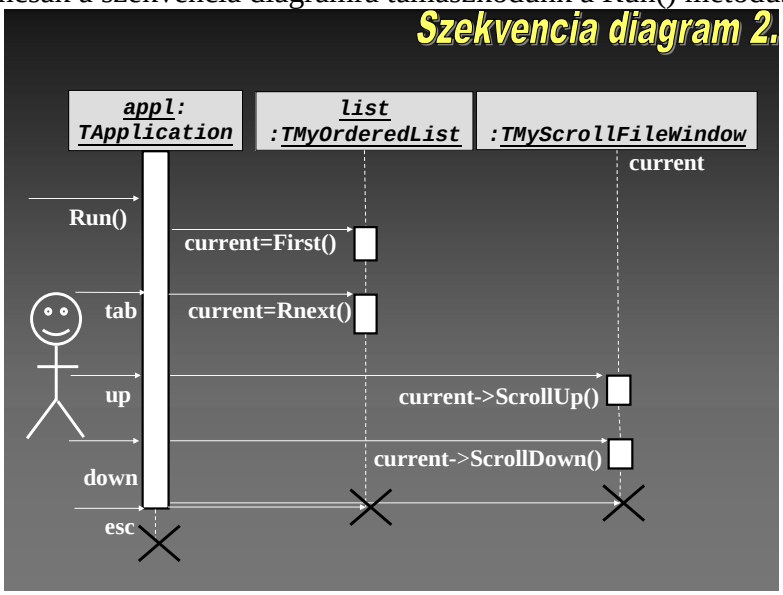



```
TApplication::TApplication()
{
    readwin = new TMyReadWindow;
    for(int i = 0; i<4; i++){
        list.Insert( new TMyScrollFileWindow
            (readwin->ReadString(),1,i*5+1) );
    }
    delete readwin;
    current = list.First();
}
```

A dinamikus kötés klasszikus példájával találkozunk a `list.Insert()` hívásnál. Az `Insert()` metódust a `TMyOrderedList` típusú `list` objektum a `TOrderedList` osztálytól örökölte. Ez a metódus egy absztrakt (így virtuális) `Ordered()` függvényt használ egy tárolóba betett két elem összehasonlítására. A `TMyOrderedList` osztályban definiáltuk a saját `Ordered()` függvényt. Ennek következtében, amikor aktivizáljuk a `list.Insert()` metódust, azon belül a dinamikus kötés miatt mindig a `list` objektumhoz tartozó `Ordered()` függvény hívódik meg.

Más vonatkozásban is találkozunk itt a dinamikus kötéssel. Az `Ordered()` függvény paraméterei `TItem` típusúak, de amikor a `list.Insert()` keretében sor kerül a meghívására, a paramétereibe a `list` tárolóban levő ablak objektumok adódnak át, ezért futási időben már teljesen korrekt a paraméterek által kijelölt ablakok fejlécének összehasonlítása. (Fordítási időben ez természetesen még értelmetlen.) Itt a dinamikus kötést a `GetTitle()` virtuális volta kényszeríti ki.

Ugyancsak a szekvencia diagramra támaszkodunk a Run() metódus implementálásánál.




```
void TApplication::Run()
{
    int key;
    for(;;){
         key = GetKey();
         switch (key){
         case tab : current = list.RNext(current)
                    break;
         case up  : current->ScrollUp();
                    break;
         case down: current->ScrollDown();
                    break;
         case esc : return;
         }
    }
}
```