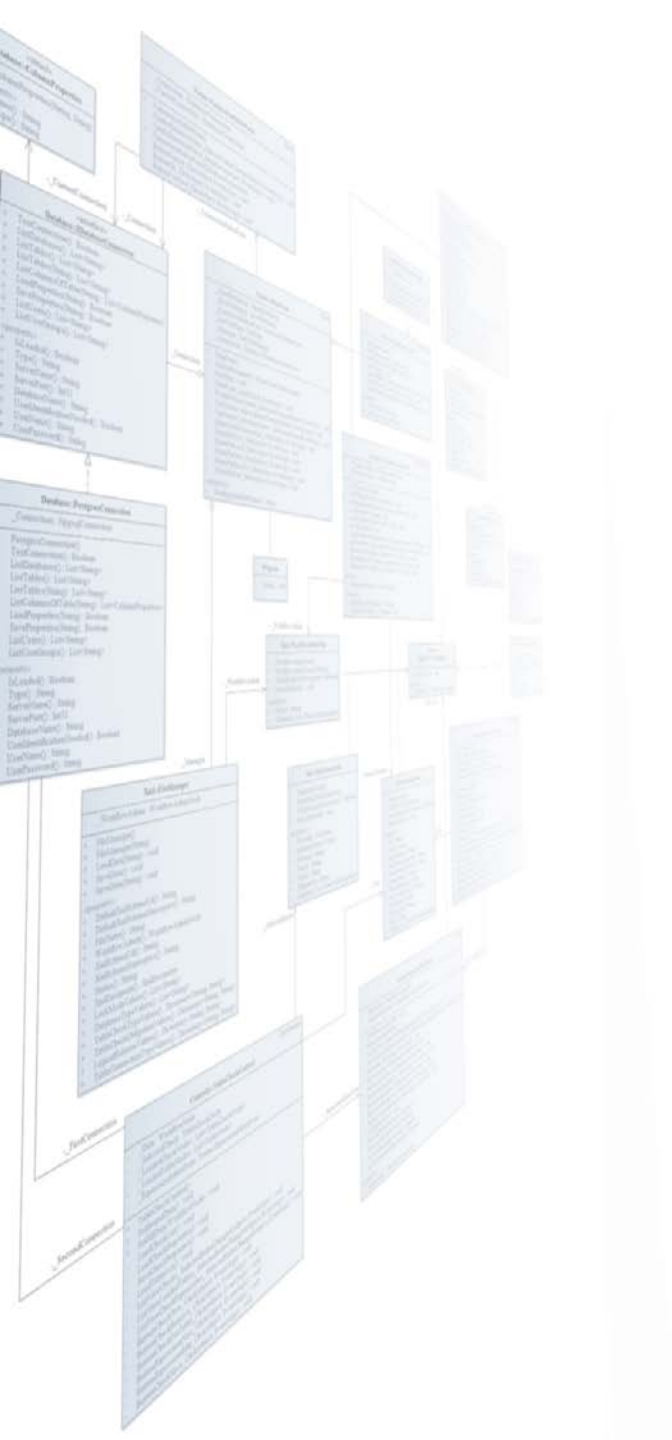


Pázmány Péter Katolikus Egyetem
Információs Technológiai Kar

Bevezetés a programozásba II

1. gyakorlat

A grafikus könyvtár használata, alakzatok rajzolása

© 2014.02.10. Giachetta Roberto
groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

A grafikus könyvtár

Programkönyvtárak

- Az alap C++ nem képes grafikát kezelni, csak a konzolt, de különböző bővítmények (programkönyvtárak) találhatóak hozzá, amelyek használatával már tudja kezelni
- A C++-hoz rengeteg könyvtár található, ezek közül csak kevés kerül telepítésre egyetlen fordító/környezet telepítésekor
  - a legtöbbet külön kell telepítenünk, használatba vennünk
- Az egyik ilyen könyvtár az *SDL* (*Simple DirectMedia Layer*), amely teljes körű 2D-3D grafikát biztosít
  - ingyenes, azonban kezelésének megtanulása sok időt vesz igénybe, ezért nem tudjuk direkt használni

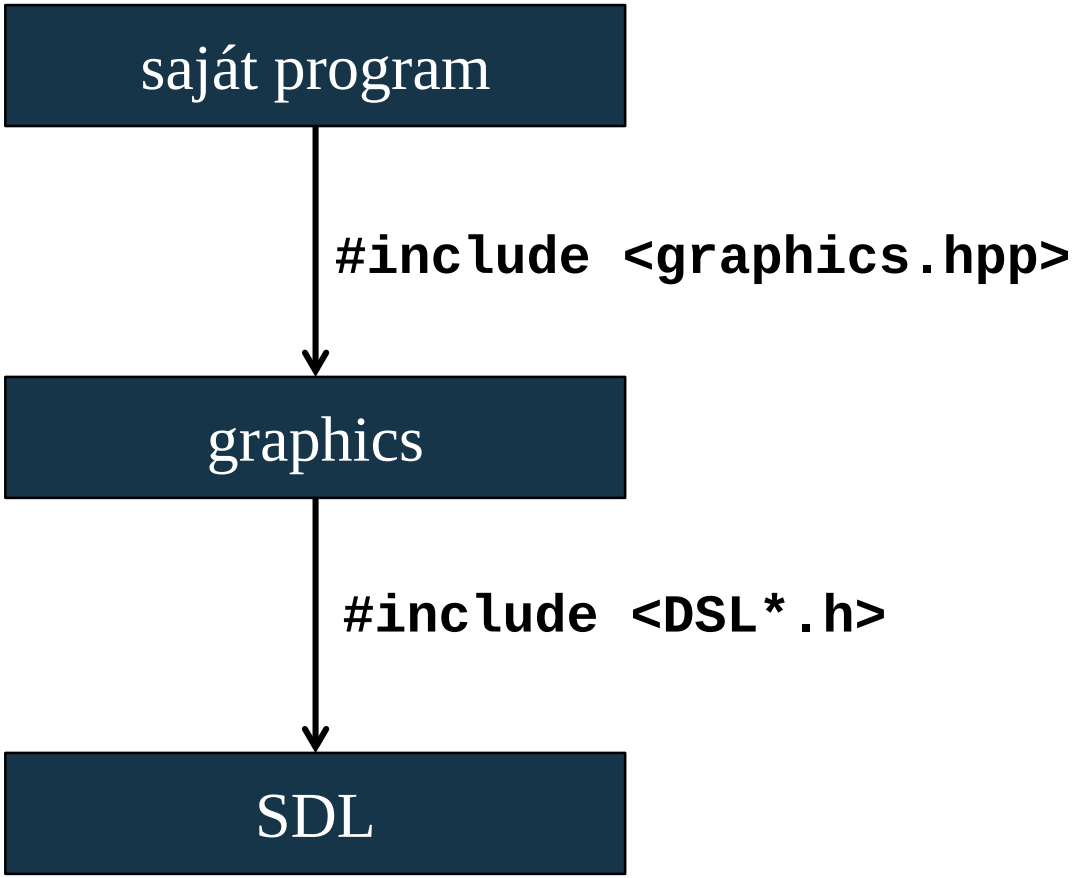
A grafikus könyvtár

A Graphics könyvtár működése

- Az SDL bonyolultsága miatt mi egy saját fejlesztésű könyvtárat használunk a grafikához
  - 2D-s egyszerű alakzatokat tartalmazó grafikus felület
  - használható a konzol felülettel párhuzamban
  - két fájl kell hozzá: **`graphics.hpp`**, **`libgraphics.a`**
- Hátránya: nem teljes, különálló grafikus könyvtár, hanem az SDL-t használja, és annak az utasításait egyszerűsíti le a számunkra, ezért az SDL-t is fel kell tennünk
  - tehát le kell töltenünk az SDL-t és telepíteni, majd le kell töltenünk a Graphics-t és telepíteni, ezután tudunk programokat készíteni

A grafikus könyvtár

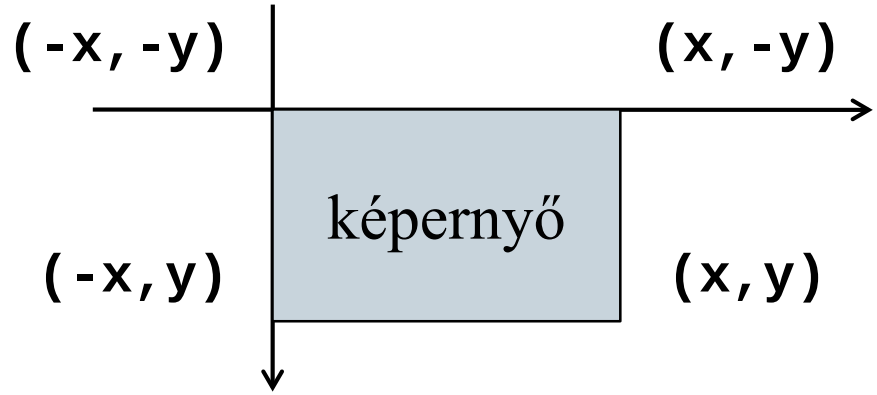
A Graphics könyvtár működése



A grafikus könyvtár használata

A képernyő kezelése

- A grafikus képernyő egy koordinátarendszerben van
  - origó a bal felső sarok: `(0,0)`
  - az értékek lefelé, illetve jobbra nőnek pixelenként
  - a képernyő méretét mi adhatjuk meg a képernyő indításakor
  - kezelhetünk a képernyőn kívül lévő pontokat is (lehetnek negatív értékek is)



A grafikus könyvtár használata

Utasítások

- A parancsok a **`genv`** névtérben vannak
- A grafikus képernyőt a **`gout.open(int width, int height)`** utasítással nyitjuk meg az ablak szélességének és magasságának megadásával
  - használhatunk teljes képernyős üzemmódot is egy harmadik paraméterrel: **`gout.open(int width, int height, bool full_screen)`**
- Lehetőségünk van a képernyő aktuális tartalmának elmentésére egy **`bmp`** fájlba a **`gout.save(string filename)`** utasítással menthetünk a megadott fájlnévre
  - igaz értékkel tér vissza, ha sikerül a mentés

A grafikus könyvtár használata

Utasítások, rajzolás

- A rajzolás során nyilvántartunk egy pontot, amelyből a rajzolás indul
  - kezdetben a **`(0,0)`** pont, de minden utasítás végén a rajzolás befejező pontja lesz
  - a rajzpont helyzete lekérdezhető a **`gout.x()`** és **`gout.y()`** függvényekkel
  - a rajzpontot eltolhatjuk a
    - **`gout << move(x,y)`** utasítással, amely az aktuális ponthoz képest tolja el
    - **`gout << move_to(x,y)`** utasítással, amely a megadott koordinátába tolja el

A grafikus könyvtár használata

Utasítások, rajzolás

- Az aktuális rajzoló színt a **`gout << color(r,g,b)`** utasítás változtatja meg
  - a következő színbeállító utasításig érvényben van, alapértelmezetten a szín fehér
  - az új színt vörös (R), zöld (G) és kék (B) komponensekből állítja elő, mindhárom komponens egy 0 és 255 közötti egész szám lehet, pl.:

```
gout << color(100,0,0); // bordó szín
… // minden kirajzolás bordó lesz
gout << color(0,255,0); // zöld szín
```

- A képernyőt frissítenünk kell ahhoz, hogy a rajzunk megjelenjen: **`gout << refresh;`**

A grafikus könyvtár használata

Utasítások, rajzolás

- Pont rajzolása: **`gout << dot;`**
  - az aktuális rajzpont színét megváltoztatja az aktuális rajzolási színre
  - a rajzpontot nem mozgatja
- Vonal: **`gout << line(x,y); gout << line_to(x,y);`**
  - egyenes szakaszt rajzol a rajzpontból kiindulva a célkoordinátába (amely lehet relatív, vagy abszolút)
- Téglalap: **`gout << box(x,y); gout << box_to(x,y);`**
  - vízszintes alapú téglalapot rajzol a rajzpontból kiindulva a célkoordinátába (amely lehet relatív, vagy abszolút)

A grafikus könyvtár használata

Utasítások, rajzolás

- Szöveg kiírása: **`gout << text(t);`**
  - kirajzolja a szöveget a rajzpontból vízszintesen induló alapvonalra, a rajzpontot a szöveg végén, az alapvonalon hagyja
  - a szöveg típusa lehet **`string`** és **`char`** is, a **`'\n'`** karaktert sorvége jelként értelmezi
- Rajzolásnál ügyelni kell arra, hogy:
  - a pixelek indexeltek, így az $n \times m$ méretű kijelző $(0 \ldots n-1, 0..m-1)$ koordinátákkal rendelkezik
  - a rajzolás a kezdő koordinátákat is magában foglalja, pl. a **`box(10, 10)`** egy 11×11 méretű téglalapot készít

A grafikus könyvtár használata

Szövegkezelés

- A szövegnek van egy ún. *alapvonala* (angolul *baseline*), ez egy láthatatlan vonal, amire „ráülnek” a karakterek
- Az alapvonal feletti rész maximális magasságát angolul *ascent*nek, az alapvonal alatti rész maximális magasságát *descent*nek hívjuk
  - **`gout.cascent()`** - egy egész számot ad eredményként, az aktuális betűkészlet ascent értékét pixelekben
  - **`gout.cdescent()`** - egy egész számot ad eredményként, az aktuális betűkészlet descent értékét pixelekben
  - **`gout.twidth(s)`** - visszaadja az s szöveg szélességét pixelekben, az aktuális betűkészlettel megjelenítve

A grafikus könyvtár használata

Eseménykezelés

- A képernyőnk *esemény-érzékeny*, azaz érzékeli a billentyűlenyomásokat, az egészmozgást, és kattintást, illetve időzíthetünk is eseményeket (részletek majd később)
- Ezeket az eseményeket felhasználva interakcióra van lehetőségünk a képernyővel, pl. ha kattintunk a képernyőn, akkor rajzolhatunk egy téglalapot
- Az eseményeket a **`gin`** változó fogadja a **`>>`** operátorral,
  - az operátor logikai értéket ad vissza, így használható feltételként, ami addig igaz, amíg az ablak nyitva van
  - az ablak bezárása után hamis lesz, ami azt jelzi, hogy több esemény nem következhet be a programban

A grafikus könyvtár használata

Az eseményciklus

- Az eseménynek típusa van, ez az **`event`** rekord, amely különböző mezőkkel rendelkezik
- Az eseményfogadó művelet a **`gin >> ev`**
  - várakozik egy eseményre, majd beteszi azt az **`event`** típusú **`ev`** változóba, eredményként a gin-t adja vissza
  - az ablak bezárása esemény kivételével mindig érvényes eseményt tesz **`ev`**-be
  - ez használható az ablak nyitva tartására:

```
event ev;
while (gin >> ev){
   // a ciklusmag minden processzorütemre lefut
}
```

A grafikus könyvtár használata

Az eseményciklus

- A legegyszerűbb grafikus program az, amely csak megnyitja a grafikus képernyőt, és nem csinál semmit:

```
#include <graphics.hpp>
using namespace genv;

int main(){
   gout.open(200,200);
      // létrehozunk egy 200*200-as képernyőt
   event ev;
      // ez az esemény csak a várakozáshoz kell
   while (gin >> ev) { } // várunk a bezárásra
   return 0;
}
```

A grafikus könyvtár használata

Példák

Feladat: Rajzoljuk ki egy 200×200-as képernyő közepére egy 100×100-as négyzetet, ehhez:

- induljunk ki az előző programból
- rajzoljunk egy 100×100-as téglalapot:

```
gout << box(99,99);
```

- a téglalapot középre kell tennünk, ezért toljuk el a rajzpontot az (50,50)-es pontba még a rajzolás előtt:

```
gout << move_to(50,50) << box(99,99);
```

- a piros színhez állítsuk be a színt még a rajzolás előtt:

```
gout << move_to(50,50) << color(255,0,0)
     << box(99,99);
```

A grafikus könyvtár használata

Példák

Megoldás:

```
#include <graphics.hpp>
using namespace genv;

int main(){
   gout.open(200,200);
   gout << move_to(50,50) // mozgatás
        << color(255,0,0) // szín beállítás
        << box(99,99);    // téglalap rajzolás
   gout << refresh;       // frissítés
   event ev;
   while (gin >> ev) {}
   return 0;
}
```

A grafikus könyvtár használata

Példák

Feladat: Készítsük el egy háromszöget az előbbi képernyő közepére, amelynek a vonala fehér, a kitöltése fekete

- ehhez három vonalat kell húznunk egymás után (azaz két rajzolás között nem kell módosítanunk a rajzpontot)

Megoldás:

```
int main(){
   gout.open(200,200);
   gout << move_to(50,100) << line_to(150,50)
        << line_to(150,150) << line_to(50,100)
        << refresh;
   event ev; while (gin >> ev) {}
   return 0;
}
```

A grafikus könyvtár használata

Példák

*Feladat: R*ajzoljunk véletlenszerűen elhelyezett, illetve változó méretű és színű téglalapokat a képernyőre

- használjunk konzolt is, a képernyő méretét, és a négyzetek számát kérjük be konzol felületen keresztül, továbbá kérjünk egy fájlnevet a felhasználótól, ahova a képernyőt elmentjük
- a grafikus képernyőt csak akkor indítsuk, ha az adatokat már beolvastuk
- használjuk a megszokott véletlenszám-generátorunkat, persze mindig a megfelelő korlátok között
- miközben fut a grafikus képernyő, a konzolon tovább dolgozhatunk

A grafikus könyvtár használata

Példák

Megoldás:

```
#include <iostream>   // újra kell az iostream
#include <string>     // string a fájlnév miatt
#include <graphics.hpp>
using namespace std;
using namespace genv; // mindkét névtér szükséges

int main(){
   srand(time(0)); // véletlengenerátor
   int x, y, z, r1, r2, r3; // változók
   cout << "A képernyő szélessége: "; cin >> x;
   cout << "A képernyő magassága: "; cin >> y;
   cout << "A négyzetek száma: "; cin >> z;
   gout.open(x,y);  // megnyitjuk a képernyőt
```

A grafikus könyvtár használata

Példák

```
// berajzoljuk egy ciklusban a négyzeteket:
for (int i = 0; i < z; i++) {
   // a véletlen szín komponensei 0..255:
   r1 = rand() % 256; r2 = rand() % 256;
   r3 = rand() % 256;
   gout << color(r1, r2, r3);

   // a kezdő pozíció 0..maximum:
   r1 = rand() % x; r2 = rand() % y;
   gout << move_to(r1,r2);
   // a téglalap mérete 0..maximum:
   r1 += rand() % x; r2 += rand() % y;
   gout << box_to(r1,r2);
}
```

A grafikus könyvtár használata

Példák

```
    gout << refresh; // elég egyszer frissíteni
    string s;
    cout << "Fájlnév: ";
       cin >> s;
       if (gout.save(s)) // ha sikeres a mentés
          cout << "A mentés megtörtént."  << endl;
       else
          cout << "A mentés sikertelen."  << endl;

    event ev;
    while (gin >> ev) {}
    return 0;
}
```

A grafikus könyvtár használata

Feladatok

1. Rajzoljuk ki az alábbi alakzatokat: