Programozás alapjai 2.

Házi feladat

Dokumentáció

Belákovics Ádám BROJTK

Feladat

Készítsen objektummodellt telefonálás modellezésére! A "Készülék" objektumok egymásnak küldött üzenetekkel modellezzék a telefonálás leegyszerűsített folyamatát (hívás, hívásfogadás, beszéd, kapcsolat bontása)! Tervezzen egy "Kapcsolat" objektumot, mely képes két telefonálni tudó objektumot összekötni!

Készítsen egyszerű programot melyen egy "Kapcsolat" objektum segítségével véletlenszerűen összekapcsol készülékeket! Demonstrálja a működést külön modulként fordított tesztprogrammal! A megoldáshoz NE használjon STL tárolót vagy algoritmust!

A tesztprogramot úgy specifikálja, hogy az parancssoros batch alkalmazásként (is) működjön, azaz ne tételezzen fel semmilyen speciális be/kiviteli eszközt, a szabványos be/kimenetet ill. a hibakimenetet úgy kezelje, hogy az átirányítható legyen fájlba. Amennyiben a feladat lehetővé teszi, érdemes a gtest_lite ellenőrző csomagot használni a teszteléshez. Ezt a Cporta rendszerbe előre feltöltöttük, csak használnia kell. Ugyanígy elérhető a memtrace környezet is.

Lehetősége van grafikus, vagy kvázi grafikus interaktív felhasználói felület kialakítására is, de fontos, hogy a Cporta rendszerbe olyan változatot töltsön fel, ami ezt nem használja! Amennyiben a feladat teszteléséhez fájlból, vagy fájlokból kell input adatot olvasnia, úgy a fájl neve *.dat alakú legyen!

Pontosított feladatspecifikáció

A feladatkiírás elég világosan leírja a létrehozandó objektumok működését és a köztük lévő kapcsolatokat, ehhez csak annyit fűznék hozzá, hogy az Készülék objektumoknak lesz egy „telefonszáma” amivel egyértelműen azonosíthatók. Ezt létrehozáskor meg kell adni.

Amiben a feladat viszonylagos szabadságot ad az a tesztelés mikéntje. A Cportára egy egyszerűbb tesztprogramot fogok írni, mely bemutatja az alapvető működéseket. Ezen kívül tervezek készíteni egy grafikus tárcsázó felületet is, amivel lehet más készülékeket felhívni, ezzel demonstrálva a működést. Ha minden jól megy, lehet megpróbálkozom egy komplexebb rendszer modellezésével is ahol már sok telefon létesít egymás között véletlenszerű kapcsolatokat. A program ezt elemzi, mint egy telefontársaság segédszoftvere.

Még nem vagyok teljesen tisztában a megvalósítás módjával, tehát nem tudom felmérni, hogy mennyi fog beleférni a fent leírt dolgokból, de ez a következő feladatrésznél már nem lesz probléma.

Osztályok és hierarchiájuk

Connection (class): Osztály mely két Készülék közti kapcsolatot valósít meg. Adattagjai a 2 Device amelyek közti kapcsolatot jelképezi. Konstruktorának meg kell adni a két Devicet melyek között a kapcsolatot létrehozza.

Tagfüggvényei:

- *Ezen kívül Abort() tagfüggvénye a kapcsolat bontásában játszik szerepet.*

Device (class): Osztály mely egy készüléket reprezentál. Minden készüléknek van Phone_number (Egy 1 + 7 jegyű kód. A Készülékek (Device-ok) megkülönböztetésére szolgáló szám. Minden Készülék kódja különböző. Az első tag függelék (prefix) például a különböző szolgáltatók megkülönböztetésére szolgálhat.) adattagja melyet, akkor kap meg, amikor csatlakozik egy Centre-hez, aki mint szolgáltató nyilvántartja, és lehetővé teszi számára a kapcsolatfelvételt más Device-okkal. Rendelkezik még 2 pointerrel. Az egyik a Centre-re mutat, melyhez csatlakozik, a másik a Connection-re mutat, amely a kapcsolatot reprezentálja.

Tagfüggvényei:

- *ConnecttoServer(): A készülék bejelentkezik egy szerverre és az felveszi tagjai közé és telefonszámot ad neki mely egyértelműen azonosítja. (Az add_device() függvény használja)*
- *Call(): Hívás kezdeményezése (lásd Fontosabb algoritmusok)*
- *Busy(): Megadja, hogy egy Device épp kapcsolatban áll-e mással (az Establish_connection például használja)*
- *Answer(): A Device reakciója egy hívásra (lásd Fontosabb algoritmusok)*
- *Initiate_Abort() és Abort(): Kapcsolat bontása (lásd Fontosabb algoritmusok)*

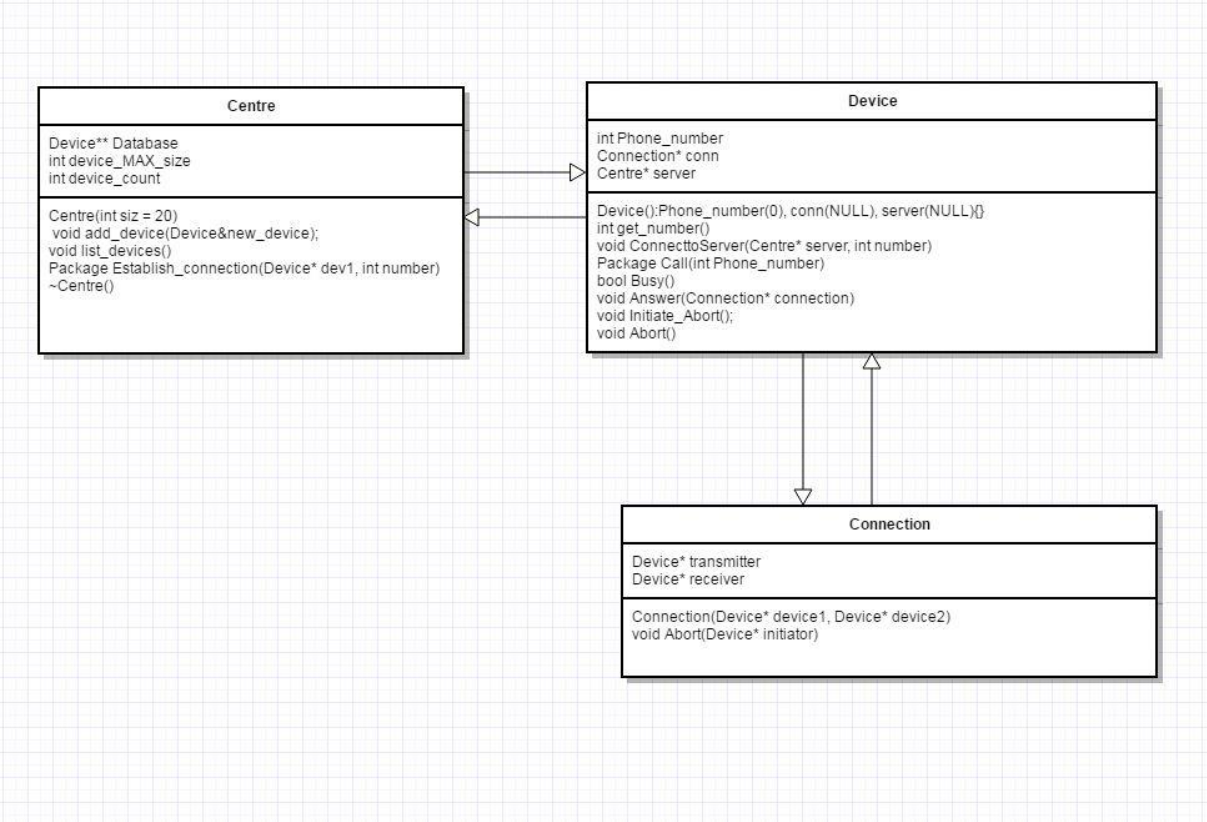
Centre (class): Osztály mely egy telefonközpontot modellez. A különböző készülékek nyilvántartására szolgál. Tartalmaz egy dinamikusan nyújtózkodó adatstruktúrát (Database), mely Device-okat tárol.

Tagfüggvényei:

- *Establish_connection(): Kapcsolat létehozása (lásd Fontosabb algoritmusok)*
- *add_device(Device*): hozzáad egy Device-ra mutató pointert a benne tárolt adatstruktúrához (Database). Ha nem fér bele std::out_of_range hibát dob.*
- *list_devices(): kilistázza a Device-okat az azonosítójuk (phone_number) szerint.*

Fontos algoritmusok

- **Hívás:** A Device Call() tagfüggvényével indítható el. Ez meghívja a Centre Establish_connection tagfüggvényét mely először ellenőrzi, hogy létezik-e az elérni kívánt fél. Ha létezik, megvizsgálja, hogy foglalt-e a Device Busy() tagfüggvénye segítégével. Amennyiben minden rendben létrehoz egy Connection objektumot a két Device-ra mutató pointerrel. A hívott fél az Device::Answer() függvényen kapja meg a Connection-re mutató pointert, míg a hívó visszakap egy Package-t az Establish_connection-től. Ebben, amennyiben a kapcsolatfelvétel sikeres megkapja a Connection-re mutató pointert, amennyiben nem, a message adattagból értesül ennek okáról.
- **Kapcsolat bontása:** Mindkét fél (Device) kezdeményezheti a Device::Initiate_Abort függvény segítségével. Ez meghívja a Connection::Abort függvényt a saját pointerével. Ez a függvény eldönti, hogy ki a kezdeményező (a kapott pointer alapján), és meghívja a Device::Abort() fgv.-t a kapcsolat másik résztvevőjére. Ez az objektum a saját conn pointerét NULL-ra állítja, majd visszattér. A Connection::Abort is visszatér, amennyiben ez megvan. Már csak annyi van hátra, hogy a Device::Initiate_Abort() felszabadítsa a Connection által elfoglalt memóriaterületet majd kinulláza a saját oda mutató pointerét.
- **Üzenet küldés:** Send_message() -> Forward_message() -> Get_message függvények valósítják meg. (később hozzáadva).



Tesztelési dokumentáció

A program tesztelése a gtest_lite keretrendszer segítségével történt. A tesztesetek 2 nagy csoportba vannak sorolva:

- test_1_basics: Ahogy a nevéből is fakad egyszerű teszteket tartalmaz az alapfunkcók ellenőrzésére. Vizsgálja többek közt a Device konstruktorát, az Busy() tagfüggvényt és a Centre::add_device() – t.
- test2_Call: Komplexebb teszteket tartalmaz a Kapcsolatfelvétel és bontás tesztelésére.

Maga a teszt alább látható:

```
void test_1_basics(){
TEST(Telecom, Device_construct) {
   Device telo;
   EXPECT_EQ(0, telo.get_number());
 } END

TEST(Telecom, Is_Busy) {
   Device telo;
   EXPECT_EQ(false, telo.Busy());
 } END

TEST(Telecom, add_device) {
   Device telo;
   Centre Relay_1(2);
   Relay_1.add_device(telo);
   EXPECT_EQ(telo.get_number(), 10000001);
 } END

TEST(Telecom, Too_much_devices) {
   Device telo;
   Centre Relay_1(2);
   Relay_1.add_device(telo);
   Relay_1.add_device(telo);
   EXPECT_THROW(Relay_1.add_device(telo), std::out_of_range);
 } END
}

void test_2_Call(){
TEST(Telecom, Non_valid_Call) {
   Device samsungA6;
   Centre Relay_1(20);
   Relay_1.add_device(samsungA6);
   std::string str = "Ezen a szamon elofizeto nem kapcsolhato.";
   EXPECT_STR_EQ(samsungA6.Call(121).message.c_str(), str.c_str());
} END

TEST(Telecom, Valid_Call) {
   Device samsungA6;
   Device iphone6s;
   Centre Relay_1(20);
```

```
   Relay_1.add_device(samsungA6);
   Relay_1.add_device(iphone6s);
   std::string message = samsungA6.Call(10000002).message;
   std::string str = "A kapcsolat letrejott 10000001 es 10000002 kozott.";
   std::cout<<std::endl;
   EXPECT_STR_EQ(message.c_str(), str.c_str());
} END

TEST(Telecom, Check_Busy) {
   Device samsungA6;
   Device iphone6s;
   Device lgspirit;
   Centre Relay_1(20);
   Relay_1.add_device(samsungA6);
   Relay_1.add_device(iphone6s);
   Relay_1.add_device(lgspirit);
   samsungA6.Call(10000002);
   std::string message = lgspirit.Call(10000002).message;
   std::string str = "A szam foglalt.";
   std::cout<<std::endl;
   EXPECT_STR_EQ(message.c_str(), str.c_str());
} END

TEST(Telecom, Check_Abort_receiver) {
   Device samsungA6;
   Device lgspirit;
   Centre Relay_1(20);
   Relay_1.add_device(samsungA6);
   Relay_1.add_device(lgspirit);
   samsungA6.Call(10000002);
   lgspirit.Initiate_Abort();
   bool busy = lgspirit.Busy();
   std::cout<<std::endl;
   EXPECT_EQ(busy , 0);

} END

TEST(Telecom, Check_Abort_transmitter) {
   Device samsungA6;
   Device lgspirit;
   Centre Relay_1(20);
   Relay_1.add_device(samsungA6);
   Relay_1.add_device(lgspirit);
   samsungA6.Call(10000002);
   samsungA6.Initiate_Abort();
   bool busy = samsungA6.Busy();
   std::cout<<std::endl;
   EXPECT_EQ(busy, 0);
} END
}
```

Programozási leírás

Az alábbiakban röviden leírom a program használatát egy példakóddal illusztrálva. Először is létre kell hozni egy Centre objektumot majd ebbe tetszőleges módon (statikusan vagy dinamikusan) Device-okat hozzá lehet adni. Ha túl sok Device kerülne egy adott Centre-be az hibát dob. A Device-ok között kapcsolatok hozhatók létre a Device::Call() tagfüggvénnyel. A Device Phone_number-ét a Centre-től kapja és bármikor lekérdezhető. Üzenetek küldhetők más Device-ok számára a Device::Send_message() függvénnyel. A program a standard kimeneten tájékoztat a létrejött kapcsolatokról, küldött üzenetekről, vagy esetleges hibákról.
A példakód:

```
Centre Relay_1(20); //Centre létrehozása maximum 20 készülékkel
Device samsungA6, sonyxperiae1); //készülékek létrehozása
Relay_1.add_device(samsungA6); //készülékek szerverhez kapcsolása
Relay_1.add_device(sonyxperiae1);
samsungA6.Call(10000002); //hívás
sonyxperiae1.Initiate_Abort(); //kapcsolat bontása
sonyxperiae1.Call(samsungA6.get_number()); //hívás más módon
sonyxperiae1.Send_message("Szia cica van gazdad?"); //üzenetküldés
samsungA6.Send_message("nincs miau");
```

Összegzés

Magát a feladatot megvalósítottnak tekintem. Sikerült létrehozni egy objektummodellt a telefonálás modellezésére. Majdnem minden alapvető funkció belekerült a programba, ami egy ilyen modellbe belefér. Ezzel viszont koránt sem tekinthető lezártnak, hisz a modell, a keretrendszer nem sokat ér használat nélkül, a későbbiekben ezt még meg lehet valósítani, esetleg egy grafikus szoftver keretein belül (pl.: telefontársaság szimulátor), ez viszont már nem képzi a Nagyházi követelményének részét.

Mellékletek

Centre.h

```
#ifndef CENTRE_H_INCLUDED
#define CENTRE_H_INCLUDED
#include <stdexcept>
#include <iostream>
#include "header.h"


class Centre{
   Device** Database; //device-ok tárolása
   int device_MAX_size;
   int device_count;

public:
   Centre(int siz = 20){
      Database = new Device*[siz];
      device_MAX_size = siz;
      device_count = 0;
```

```
    }
    void add_device(Device& new_device);
    void list_devices();
    Package Establish_connection(Device* dev1, int number);
    ~Centre(){
        delete[] Database;
    }
};


#endif // CENTRE_H_INCLUDED
```

Connection.h

```
#ifndef CONNECTION_H_INCLUDED
#define CONNECTION_H_INCLUDED
#include <stdexcept>
#include "header.h"

class Connection{
    Device* transmitter;
    Device* receiver;
public:
    Connection(Device* device1, Device* device2){
        transmitter = device1;
        receiver = device2;
    }
    void Abort(Device* initiator);
    void Forward_message(Device* dev, Package message);
};



#endif // CONNECTION_H_INCLUDED
```

Device.h

```
#ifndef DEVICE_H_INCLUDED
#define DEVICE_H_INCLUDED
#include <cstdlib>
#include <string>
#include "header.h"


class Device{
   int Phone_number;
   Connection* conn;
   Centre* server;
public:
   Device():Phone_number(0), conn(NULL), server(NULL){};
   int get_number(){
      return Phone_number;
   };
   void ConnecttoServer(Centre* server, int number){
      Phone_number = number;
      this->server = server;
   };
   Package Call(int Phone_number);
   bool Busy(){// ha foglalt igaz
      return conn != NULL;
   };
   void Answer(Connection* connection){
      conn = connection;
   };
   void Initiate_Abort();
   void Abort(){
      conn = NULL;
   };
   void Send_message(std::string message);
   void Get_message(Package message);
};

#endif // DEVICE_H_INCLUDED
```

Package.h

```
#ifndef PACKAGE_H_INCLUDED
#define PACKAGE_H_INCLUDED
#include "header.h"

struct Package{
   Connection* connection;
   std::string message;
   Package(Connection* connection, const char* message): connection(connection), message(message){};
};

#endif // PACKAGE_H_INCLUDED
```

header.h

```
#ifndef HEADER_H_INCLUDED
#define HEADER_H_INCLUDED

//elodeklaracio linkelesi okokbol
class Device;
class Centre;
class Connection;
class Package;

#endif // HEADER_H_INCLUDED
```

functions.cpp

```
#include <iostream>
#include <cstdio>
#include <string>
#include <sstream>
#include "Centre.h"
#include "Connection.h"
#include "Device.h"
#include "Package.h"
#include "header.h"



Package Device::Call(int Phone_number){
      Package response = server->Establish_connection(this, Phone_number);
      if ((conn = response.connection) != NULL){ //connection beallítasa
         std::stringstream ss;
         ss <<"A kapcsolat letrejott "<<this->Phone_number<<" es "<<Phone_number<<"
kozott.";
         response.message = ss.str();
         std::cout<<"A kapcsolat letrejott "<<this->Phone_number<<" es "<<Phone_number<<"
kozott."<<std::endl;
         return response;
      }
      else{
         std::cout<<response.message<<std::endl;
         return response;
      }
}

void Device::Initiate_Abort(){
   if (conn != NULL){
      conn->Abort(this);
      delete conn;
      conn = NULL;
   }
   else
      throw std::out_of_range("Nem letezo kapcsolat");
};

void Device::Send_message(std::string message_str){
   if(conn != NULL){
      Package message(NULL, message_str.c_str());
      conn->Forward_message(this, message);
   }
   else
```

```
      throw std::out_of_range("Nem lehet uzenetet kuldeni nincs kapcsolat!!!");
};

void Connection::Forward_message(Device* dev, Package message){
   if(dev == transmitter){
      receiver->Get_message(message);
   }
   else if (dev == receiver){
      transmitter->Get_message(message);
   }
}

void Device::Get_message(Package message){
   std::cout<<Phone_number<<" uzenetet kapott: "<<message.message<<std::endl;
}

Package Centre::Establish_connection(Device* dev1, int number){
         Device* dev2 = NULL;
         for (int i = 0; i < device_count; i++){ //Keresés telefonszám alapján
            if(Database[i]->get_number() == number)
               dev2 = Database[i];
         }
         if (dev1 == dev2){
            throw std::out_of_range("Sajat iranyba torteno hivas!!!");
         }
         else if (dev2 == NULL){
            return Package(NULL, "Ezen a szamon elofizeto nem kapcsolhato.");
         }
         else if (dev2->Busy()){
            return Package(NULL, "A szam foglalt.");
         }
         else{
            Connection* connect_dev1_dev2 = new Connection(dev1, dev2);
            dev2->Answer(connect_dev1_dev2);
            return Package(connect_dev1_dev2, "");
         }
}

 void Centre::add_device(Device& new_device){
      if (device_count < device_MAX_size){
         Database[device_count++] = &new_device;
         new_device.ConnecttoServer(this, 10000000 + device_count);
      }
      else
         throw std::out_of_range("Kozpont tulterhelve: Database");
}
```

```
void Centre::list_devices(){
	for (int i = 0; i < device_count; i++)
	   std::cout<<(*Database[i]).get_number()<<std::endl;
}


void Connection::Abort(Device* initiator){
	if(initiator == transmitter){
	   receiver->Abort();
	}
	else if(initiator == receiver){
	   transmitter->Abort();
	}
	else
	   throw std::out_of_range("hibas kapcsolat: Connection/Abort()");
};
```

main.cpp

```
#include <iostream>
#include <cstdlib>
#include <time.h>
#include <stdexcept>
#include "Device.h"
#include "Centre.h"
#include "Connection.h"
#include "Package.h"
#include "header.h"
#include "tests.h"



int main(void)
{
   try{
   Device samsungA6, iphone6s, lgSpirit, sonyxperiae1;
   Centre Relay_1(20);
   Relay_1.add_device(samsungA6);
   Relay_1.add_device(iphone6s);
   Relay_1.add_device(lgSpirit);
   Relay_1.add_device(sonyxperiae1);
   std::cout<<"Keszulekek:"<<std::endl;
   Relay_1.list_devices();
   samsungA6.Call(121);
   samsungA6.Call(10000003);
   iphone6s.Call(10000003);
   lgSpirit.Initiate_Abort();
   iphone6s.Call(10000003);
   sonyxperiae1.Call(10000001);
   sonyxperiae1.Send_message("Szia cica van gazdad?");
   samsungA6.Send_message("nincs miau");
   iphone6s.Send_message("Segitunk! Ne hagyja el a gepjarmuvet! Ha elfogy az uzemanyaga,
uljon at masik gepjarmube! Belugyminiszterium");
   lgSpirit.Send_message("Error: invalid conditions");
   }
   catch(std::exception& e){
      std::cerr<<std::endl<<"HIBA----->"<<e.what();
   }
   return 0;
}
```