Programming in C/C++
Exercises set two: inheritance

Christiaan Steenkist
Jaime Betancor Valado
Remco Bos

November 17, 2016

Exercise 12, matrix copying

We were tasked with initializing an array with copies of a matrix `mat` with only a `new` statement.

Solution

Because the default constructor is always called we made a new class that is derived from matrix. This `Copymatrix` holds a static `Matrix` that is used to initialize the class. By returning a `Matrix` pointer the unnecessary data is sliced off.

Code listings

Listing 1: main.ih

```
#include "main.h"

using namespace std;
```

Listing 2: main.h

```
#ifndef MAIN_H_
#define MAIN_H_

#include "matrix/matrix.h"

Matrix *factory(Matrix const &mat, size_t count);
```

```

#endif
```

Listing 3: factory.cc

```
#include "main.ih"
#include "copymatrix.h"

Matrix CopyMatrix::d_blueprint;

Matrix *factory(Matrix const &mat, size_t count)
{
  CopyMatrix::d_blueprint = mat;
  return new CopyMatrix[count];
}
```

Listing 4: main.cc

```
#include "main.ih"

int main(int argc, char **argv)
{
  Matrix mat({{0, 1, 2, 3,}, {4, 5, 6, 7}});

  size_t count = 8;
  Matrix *matArray = factory(mat, count);

  for (size_t index = 0; index != count; ++index)
  {
    cout << matArray[index]
      << '\n';
  }

  delete[] matArray;
}
```

Copymatrix

Listing 5: copymatrix.ih

```
#include "copymatrix.h"
```

Listing 6: copymatrix.h

```cpp
#ifndef COPYMATRIX_H_
#define COPYMATRIX_H_

#include "matrix/matrix.h"

class CopyMatrix: public Matrix
{
  public:
    static Matrix d_blueprint;

    CopyMatrix();
};

#endif
```

Listing 7: copymatrix1.cc

```cpp
#include "copymatrix.ih"

CopyMatrix::CopyMatrix()
:
  Matrix(d_blueprint)
{}
```

Exercise 13, red thread

We were tasked with using inheritence to include the `Limits` class into a number of other classes.

Code listings

Listing 8: fighter.h

```cpp
#ifndef INCLUDED_FIGHTER_
#define INCLUDED_FIGHTER_

#include "../limits/limits.h"
#include "../time/time.h"
#include "../coordinates/coordinates.h"
#include "../speed/speed.h"
```

```
#include "../altitude/altitude.h"
#include "../heading/heading.h"
#include "../registerdata/registerdata.h"
#include "../units/units.h"


class Fighter: private Limits
{
    RegisterData d_rd;

  // keeps track of time-related info
    Time        d_time;
    Units       d_units;
    Coordinates d_coord;
    Speed       d_speed;
    Altitude    d_altitude;
    Heading     d_heading;

  // true: inside the box
    bool   d_inTheBox = false;

    static size_t s_nFighters;
    static size_t s_nRegisteredFighters;

    public:
        Fighter(RegisterData const &rd,
        // 1
                int xCoord, int yCoord, int units);
        ~Fighter();

        void setUnits(int type);

        void altitudeTo(size_t altitude, size_t rate);
        void headingTo(char direction, size_t hdg,
      double acceleration);
        void speedTo(size_t kts);

    // default: no update time changes
        void info(size_t silentTime= 0);
    private:
        void boxStatus();
};

#endif
```

Listing 9: fighter's altitudeto.cc

```
#include "fighter.ih"

void Fighter::altitudeTo(size_t req, size_t rate)
{
    d_altitude.set(
            d_units.setAlt(req),
            rate == 0 ? DEFAULT_CLIMBRATE
        : d_units.setRate(rate)
    );
}
```

Listing 10: monitor.h

```
#ifndef INCLUDED_MONITOR_
#define INCLUDED_MONITOR_

    // messages to the monitor are received
  // on fifo '0'

#include <string>

#include "../limits/limits.h"
#include "../fightermap/fightermap.h"

class Monitor: private Limits
{
    FighterMap d_fighter;
    std::string d_fifo;

    public:
        Monitor(char const *dir);
        Monitor(Monitor const &other) = delete;

        void run();

    private:
    // 1st char already removed
        void insert(std::istringstream &instr);
        // 1st char already removed
        void remove(std::istringstream &instr);
};

#endif
```

Listing 11: time.h

```
#ifndef INCLUDED_TIME_
#define INCLUDED_TIME_

#include <iosfwd>

#include "../limits/limits.h"

class Time: private Limits
{
    size_t d_TOtime = 0;    // take-off time

    // time elapsed since the previous update;
    size_t d_delta;
    size_t d_time;
    // time in seconds at the last update

  // clock-time set by updateTime()
    static size_t s_sec;

    public:
        Time();
        // update the time for a Fighter
        void step();
        // since take-off
        size_t elapsed() const;
        void registerTOtime();
        size_t delta() const;
        size_t fuelRemaining() const;
        // called by Monitor::childProcess before
        // updating the Fighters' data
        static void updateTime();

    // returns the common clock-time
        static size_t clock();
};

inline Time::Time()
:
    d_time(s_sec)
{}

inline size_t Time::delta() const
{
    return d_delta;
}

inline size_t Time::elapsed() const
{
    return d_time - d_TOtime;
}

inline size_t Time::fuelRemaining() const
{
    return (FUEL_EMPTY - elapsed()) / 60;
}

inline size_t Time::clock()
{
    return s_sec;
}

#endif
```