"Ss. Cyril and Methodius" University in Skopje
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Object oriented programming

Exercises 12

Version 1.0, 9 May, 2017

Table of Contents

1. Virtual destructor and example problems

1.1. Virtual destructor

1.1.1. Example 1 – Why and when we need virtual destructor?

Solution without virtual destructor `oop_av121a_en.cpp`

```
#include <iostream>
using namespace std;
class Base {
public:
    Base() { cout << "Constructor of Base\n";}
    // this is the destructor
    ~Base() { cout << "Destructor of Base\n";}
};

class Derived : public Base
{
public:
    Derived() { cout << "Constructor of Derived\n"; }
    ~Derived() { cout << "Destructor of Derived\n"; }
};

int main() {
    Base *basePointer = new Derived();
    delete basePointer;
}
```

Output from the program

```
Constructor of Base
Constructor of Derived
Destructor of Base
```

Solution with virtual destructor `oop_av121b_en.cpp`

```
#include <iostream>
using namespace std;
class Base {
public:
    Base() { cout << "Constructor of Base\n";}
    // this is the destructor
    virtual ~Base() { cout << "Destructor of Base\n";}
};

class Derived : public Base
{
public:
    Derived() { cout << "Constructor of Derived\n"; }
    ~Derived() { cout << "Destructor of Derived\n"; }
};

int main() {
    Base *basePointer = new Derived();
    delete basePointer;
}
```

Output from the program

```
Constructor of Base
Constructor of Derived
Destructor of Derived
Destructor of Base
```

1.2. Second partial exam 2015/2016

1.2.1. Problem 1

Define a class `Goal` that keeps information for:

- the goal scorer (dynamically allocated array of characters),
- time (the minute) of the goal scoring event (integer),
- name of the team that scored the goal (char array max 50).

For this class implement:

- constructor with all three arguments
- operator `<<` for printing on `ostream` an object of class `Goal` (print the minute and the name)
- operator `++` (postfix) notation for incrementing the minute for one,
- operator `--` (prefix) notation for decrementing the minute for one.

Also create a class `Game` that keeps dynamically allocated array of objects from the class `Goal` and the names of the teams that play that game (two char arrays of 50). For this class implement:

- constructor with two arguments the names of the teams that play the game
- the unary operator `+=` for adding an object of class `Goal` in the array of objects. If the name of the object from the class `Goal` is not equal to the one of the names of the teams that play the game then throw an exception of type `InvalidTeamName`. Handling the exception should be in the appropriate place in the main function, by printing the message `Invalid team name: [the_name]`
- operator `<<` for printing on the `ostream` the teams that play the game and all the scorers.

Solution `oop_av122_en.cpp`

```
#include <iostream>
#include <cstring>
using namespace std;

class InvalidTeamName : exception {
    char msg[100];
public:
    InvalidTeamName(const char* name) {
        strcpy(msg, name);
    }
    const char* what() {
        return msg;
    }
};

class Goal {
private:
    char* name;
    int minute;
    char team[50];
    void copy(const Goal &g) {
        name = new char[strlen(g.name)];
        strcpy(name, g.name);
        minute = g.minute;
        strcpy(team, g.team);
    }
public:
    Goal() {
        name = NULL;
    }
    Goal(const char* n, int m, const char* t) {
        name = new char[strlen(n)];
        strcpy(name, n);
        minute = m;
        strcpy(team, t);
    }

    Goal(const Goal &g) {
        copy(g);
    }

    Goal& operator=(const Goal& g) {
        if(&g == this) return *this;
        delete [] name;
        copy(g);
        return *this;
    }

    friend ostream& operator<<(ostream& out, const Goal& g) {
        out << g.minute << " " << g.name;
        return out;
    }

    Goal operator++(int) {
        Goal g = *this;
        ++minute;
        return g;
    }

    Goal& operator--() {
        --minute;
        return *this;
    }

    char* getTeam() {
        return team;
    }
};

class Game {
private:
    Goal* goals;
    char teamHome[50];
```

```
    char teamGuest[50];
    int n;
    void copy(const Game& game) {
      goals = new Goal[game.n];
      n = game.n;
      for(int i = 0; i < n; ++i) {
        goals[i] = game.goals[i];
      }
      strcpy(teamHome, game.teamHome);
      strcpy(teamGuest, game.teamGuest);
    }
public:
    Game(const char* t1, const char* t2) {
        goals = NULL;
        n = 0;
        strcpy(teamHome, t1);
        strcpy(teamGuest, t2);
    }

    Game(const Game& game) {
        copy(game);
    }

    Game& operator=(const Game& game) {
        if(this == &game) return *this;
        delete [] goals;
        copy(game);
        return *this;
    }

    Game& operator+=(Goal &g) {
        if(strcmp(g.getTeam(), teamHome) != 0&&strcmp(g.getTeam(), teamGuest)) {
            throw InvalidTeamName(g.getTeam());
        }
        Goal* temp = goals;
        goals = new Goal[n + 1];
        for(int i = 0; i < n; ++i) {
            goals[i] = temp[i];
        }
        delete [] temp;
        goals[n] = g;
        ++n;
        return *this;
    }

    friend ostream& operator<<(ostream &out, const Game &n) {
        out << n.teamHome << " - " << n.teamGuest << endl;
        for(int i = 0; i < n.n; ++i) {
            cout << n.goals[i] << endl;
        }
        return out;
    }
};

int main() {
    char team1[50];
    char team2[50];
    cin >> team1;
    cin >> team2;
    Game n(team1, team2);
    int x;
    cin >> x;
    char player[100];
    int m;
    for(int i = 0; i < x; ++i) {
        cin >> player;
        cin >> m;
        cin >> team1;
        Goal g(player, m, team1);
        try {
              n += g;
        } catch(InvalidTeamName &e) {
            cout << "Invalid team name: " << e.what() << endl;
        }
    }
    cout << n << endl;
```

```
    return 0;
}
```

1.3. Problem 2

Implement class for `Ticket` that keeps info for ID of the ticket as an array of 50 characters and a length of the ID (integer that is not larger than 50).

From this class derive two classes for tickets `DigitsTicket`and `CharTicket` that should implement the following methods:

- constructor with one argument N the length of the ID that generates a random ID with the given length
- for the class `DigitTicket` generates N random digits and fills the ID with them.
- for the class `CharTicket` generates N random characters (A-Z) and fills the ID with them.
- **you should use the external functions for generating random digit `randomDigit` and random char `randomChar`**
- `bool validate()` to validate the ticket
- for the class `DigitTicket` the ID is valid if the sum of the digits is a number divisible with 7
- for the class `CharTicket` the ID is valid if the sum of the ASCII codes is a number divisible with 3.

Override the operators `==` and `!=` for comparing two tickets of any kind by their ID. Two tickets are equal if they have the same ID.

Implement an external function `int valid(Ticket **tickets, int n)` that for an array of pointers of the class `Ticket` and the length of the array, will return how many of the tickets are valid.

Implement an external function `int unique(Ticket **tickets, int n)` that for an array of pointers of the class `Ticket` and the length of the array, will return how many of the tickets are unique (does not have duplicate in the rest of the array). If there is at least one ticket with the same ID with given ticket, that ticket is NOT unique.

Solution `oop_av123_en.cpp`

```
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;

char randomDigit() {
    return '0' + rand() % 10;
}

char randomChar() {
    return 'A' + rand() % 26;
}

class Ticket {
protected:
    char id[50];
    int len;
public:
    virtual bool validate() const = 0;

    friend ostream& operator<<(ostream& out, const Ticket &t) {
        for(int i = 0; i < t.len; ++i) {
            out << t.id[i];
        }
        return out;
    }

    bool operator==(const Ticket& t) {
        if(len != t.len) return false;
        for(int i = 0; i < t.len; ++i) {
            if(id[i] != t.id[i]) return false;
        }
        return true;
    }
};

class DigitTicket : public Ticket {
public:
    DigitTicket(int n) {
        len = n;
        for(int i = 0; i < n; ++i) {
            id[i] = randomDigit();
        }
    }

    bool validate() const {
        int sum = 0;
        for(int i = 0; i < len; ++i) {
            sum += id[i] - '0';
        }
        return sum % 7 == 0;
    }

};

class CharTicket : public Ticket {
public:
    CharTicket(int n) {
        len = n;
        for(int i = 0; i < n; ++i) {
            id[i] = randomChar();
        }
    }

    bool validate() const {
        int sum = 0;
        for(int i = 0; i < len; ++i) {
            sum += id[i];
        }
        return sum % 3 == 0;
    }

};
```

```
int valid(Ticket** t, int n) {
    int valid = 0;
    for(int i = 0; i < n; ++i) {
        if(t[i]->validate()){
            ++valid;
        }
    }
    return valid;
}

int unique(Ticket **t, int n) {
    int duplicates = 0;
    for(int i = 0; i < n - 1; ++i) {
        for(int j = i + 1; j < n; ++j) {
            if(*t[i] == *t[j]&&i != j) {
                ++duplicates;
            }
        }
    }
    return n - duplicates;
}

int main() {
    int seed;
    cin >> seed;
    srand (seed);
    int n;
    cin >> n;
    Ticket **t = new Ticket*[n];
    cout << "===== ALL TICKETS (" << n << ") =====" << endl;
    for(int i = 0; i < n; ++i) {
        int x;
        cin >> x;
        if(i % 2 == 0) {
            t[i] = new DigitTicket(x);
        } else {
            t[i] = new CharTicket(x);
        }
        cout << *t[i] << endl;
    }
    cout << "===== VALID =====" << endl;
    cout << valid(t, n) << endl;

    cout << "===== UNIQUE =====" << endl;
    cout << unique(t, n) << endl;

    for(int i = 0; i < n; ++i) {
        delete t[i];
    }
    delete [] t;
    return 0;
}
```

2. Source code of the examples and problems

https://github.com/finki-mk/SP/

Source code ZIP