

Теми за проекти

курс *Обектно-ориентирано програмиране*
за специалност *Информационни системи*
Летен семестър 2017/2018 г.

Обща информация за проектите

Проектите се оценяват по редица от критерии, част от които са описани по-долу. Тъй като курсът се фокусира върху обектно-ориентираното програмиране и неговата реализация в езика C++, най-важното изискване за проектите е те да са изградени съгласно добрите принципи на ООП. Решението, в което кодът е процедурен, има лоша ООП архитектура и т.н. се оценява с нула точки. Други важни критерии за оценка на проектите са:

- Дали решението работи коректно, съгласно спецификацията. Решение, което не работи и/или не се компилира носи минимален брой (или нула) точки.
- Дали решението отговаря на заданието на проекта.
- Каква част от необходимата функционалност е била реализирана.
- Дали решението е изградено съгласно добрите практики на обектно-ориентирания стил. Тъй като курсът се фокусира върху ООП, решения, които не са обектно-ориентирани се оценяват с нула или минимален брой точки.
- Оформление на решението. Проверява се дали кодът е добре оформен, дали е спазена конвенцията за именуване на променливите, дали е добре коментиран и т.н.
- Дали решението е било добре тествано. Проверява се какви тестове са били проведени върху приложението, за да се провери дали то работи коректно. Очаква се по време на защитата да можете да посочите как сте тествали приложението, за да проверите дали то работи коректно и как се държи в различни ситуации.

По време на защитата се очаква да можете да отговорите на различни въпроси, като например: (1) каква архитектура сте избрали, (2) защо сте избрали именно нея, (3) дали сте обмислили други варианти и ако да — кои, (4) как точно работят различните части от вашия код и какво се случва на по-ниско ниво и др.

Оценката на всеки от проектите се формира от онази негова част, която е била самостоятелно разработена от вас. Допустимо е да използвате код написан от някой друг (напр. готова библиотека или помощ от ваш приятел/колега), но (1) той не носи точки към проекта и (2) това трябва да бъде ясно обявено както при предаването, така и при защитата на проекта, като ясно обозначите коя част от проекта сте разработили самостоятелно. Това означава, че:

1. Използваният наготово код трябва да се маркира ясно, като поставите коментари на подходящи места в кода си.
2. По време на защитата трябва да посочите кои части сте разработили самостоятелно и кои са взети от други източници.

Както е написано по-горе, когато в проекта си използвате чужд код, сам по себе си той не ви носи точки. Допълнителни точки могат да се дадат или отнемат, според (1) способността ви за внедряване на кода във вашето решение (напр. в случаите, когато се използва външна библиотека) и за това (2) дали добре разбирате какво прави той.

Проект 1: XML Parser

Да се напише програма, реализираща четене и операции с [XML](#) файлове. Характеристиките на XML елементите, поддържани от програмата, да се ограничат до:


- идентификатор на елемента
- списък от атрибути и стойности
- списък от вложени елементи или текст

Да се поддържат уникални идентификатори на всички елементи по следния начин:

- Ако елементът има поле "id" във входния файл и стойността му е уникална за всички елементи от файла, да се ползва тази стойност.
- Ако елементът има поле "id" във входния файл, но стойността му не е уникална за всички елементи от файла, да се ползва тази стойност, но към нея да се конкатенира някакъв низ, който да допълни идентификатора до уникален низ. (например, ако два елемента имат поле id="1", то единият да получи id="1_1", а другият -id="1_2")
- Ако елементът няма поле "id" във входния файл, да му се присъедини уникален идентификатор, генериран от програмата.

Програмата да дава следните възможности команди:

Open	Отваря и прочита валиден XML файл
Print	Извежда на екрана прочетената информация от XML файла (в рамките на посочените по-горе ограничения за поддържаната информация). Печатането да е XML коректно и да е "красиво", т.е. да е форматирано визуално по подходящ начин (например, подчинените елементи да са по-навътре)
Select <id> <key>	Извежда стойност на атрибут по даден идентификатор на елемента и ключ на

	атрибута
Set <id> <key> <value>	Присвояване на стойност на атрибут
Children <id>	Списък с атрибути на вложените елементи
Child <id> <n>	Достъп до n-тия наследник на елемент
Text <id>	Достъп да текста на елемент
Delete <id> <key>	Изтриване на атрибут на елемент по ключ
Newchild <id>	Добавяне на НОВ наследник на елемент. Новият елемент няма никакви атрибути, освен идентификатор
XPath <id> <XPath>	операции за изпълнение на прости XPath 2.0 заявки към даден елемент,  която връща списък от XML елементи

Минимални изисквания за поддържаните XPath заявки

Примерите по-долу са върху следния прост XML низ:

```
<people>
  <person id="0">
    <name>John Smith</name>
    <address>USA</address>
  </person>
  <person id="1">
    <name>Ivan Petrov</name>
    <address>Bulgaria</address>
  </person>
</people>
```

- да поддържат оператора / (например “person/address” дава списък с всички адреси във файла)
- да поддържат оператора [] (например “person/address[0]” дава адресът на първия елемент във файла)
- да поддържат оператора @ (например “person(@id)” дава списък с id на всички елементи във файла)
- Оператори за сравнение = (например “person(address=“USA”)/name” дава списък с имената на всички елементи, чиито адреси са “USA”)

Забележка: За проекта не е позволено използването на готови библиотеки за работа с XML. Целта на проекта е да се упражни работата със структурирани текстови файлове, а не толкова със самия XML. **Внимание:** Не се изисква осигуряване на всички условия в XML и XPath спецификациите! Достатъчно е файловете да “приличат на XML” (както файла в горния пример, който не е валиден XML), а завките да “приличат” на XPath.

Бонуси:

- да се реализират [XML namespaces](#)
- да се реализират различните XPath оси (ancestor, child, parent, descendant,...)

Проект 2: Бази от данни

Да се реализира програма, поддържаща операции с прости бази от данни. Базите данни се състоят от серии от таблици, като всяка таблица е записана в собствен файл.

Поддържани типове данни

Всяка “колона” на таблица в базата данни има тип, като в една таблица може да има едновременно колони от различни типове. Вашето приложение трябва да може да поддържа следните типове:

Цяло число – поредица от цифри, без никакви други символи между тях. В началото на числото може да има знак '+' или '-'. Например:

```
123
-123
+123
```

Дробно число – поредица от цифри, следвана от символ за точка и след нея друга поредица от цифри. В началото на числото може да има знак '+' или '-'. Например:

```
123.456
-123.456
+123.456
```

Символен низ (стринг) – поредица от произволни символи оградени в кавички. Подобно на низовете в C++, ако искате да включите символа за кавичка в даден низ, трябва да го представите като \", а ако искате да включите наклонена черта, трябва да я представите като \\. Например:

```
"Hello world!"
"C:\\temp\\"
"\\"This is a quotation\\""
```

Освен конкретна стойност, дадена клетка в даден ред на таблицата може да е “празна”. Такива клетки да се обозначават специално и да е изписват като “NULL”.

Програмата да позволява следните операции:

Load <file name>	Зарежда таблица от файл. Във файла е записана информация за типа на всяка колона. Всяка таблица има име. При опит за зареждане на таблица с име, което съвпада с името на някоя вече заредена таблица, системата да дава грешка.
Showtables	Показва списък с имената на всички заредени таблици
Describe <name>	Показва информация за типовете на колоните на дадена таблица
Print <name>	Показва всички редове от дадена таблица. Да се реализира диалогов режим, позволяващ съдържанието на таблицата да се преглежда по страници (такива, че се събират на един екран) със следните команди: следваща страница, предишна страница, изход.
Save <name> <file name>	Записва таблица във файл
Select <column-n> <value> <table name>	Извежда всички редове от таблицата, които съдържат стойността "value" в клетката с дадения пореден номер. Да се реализира извеждане по страници
AddColumn <table name> <column name> <column type>	Добавя нова колона (с най-голям номер) в дадена таблица. За всички съществуващи редове от таблицата,

	стойността на тази колона да е празна.
Update <table name> <search column n> <search value> <target column n> <target value>	За всички редове в таблицата, чиято колона с пореден номер <search column n> съдържа стойността <search column value> се променят така, че колоната им с пореден номер <tagret column n> да получи стойност <taget value>. Да се поддържа стойност NULL.
Delete <table name> <search column n> <search value>	Изтрива всички редове в таблицата, чиято колона <search column n> съдържа стойността <search column value>
Insert <table name> <column 1> ... <column n>	Вмъква нов ред в таблицата със съответните стойности
InnerJoin <table 1> <column n1> <table 2> <column n2>	Извършва операцията Inner Join над две таблици спрямо колони <column n1> в първата таблица и <column n2> във втората. Създава нова таблица и извежда идентификатора и.
Rename <old name> <new name>	Преименува таблица. Отпечатва грешка, ако новото име не е уникално.
Count <table name> <search column n> <search value>	Намира броя на редовете в таблицата, чиито колони съдържат дадената стойност

<p>Aggregate <table name> <search column</p> <p>n> <search value> <target column n></p> <p><operation></p>	<p>Извършва дадена операция върху стойностите от колоната <target column</p> <p>n> на всички редове, чиито колони с номер <search column n> съдържат стойността <search value>.</p> <p>Възможните операции са sum, product, maximum, minimum. Системата да дава грешка, ако колоните не са числови.</p>
--	---

Проект 3: Рали шампионат

Да се реализира система за симулиране на Световния рали шампионат:

Световният рали шампионат се характеризира със следните данни:

1. Опеделен брой състезания
2. Определен брой участници – екипи с аналогични по качества автомобили и двама, различни по качества пилоти

3. Титла при конструкторите

4. Титла при пилотите

Регламент на рали шампионата:

1. Във всяко състезание участват и двамата пилоти от всеки екип
2. Състезанието се печели от пилота преминал всички етапи с най-добро време
3. Победителите се награждават по следната скала:

Първо място	20 – точки за пилота	20 – точки за екипа
Второ място	15 – точки за пилота	15 – точки за екипа
Трето място	10 – точки за пилота,	10 – точки за екипа
Четвърто място	8 – точки за пилота,	8 – точки за екипа
Пето място	6 – точки за пилота,	6 – точки за екипа
Шесто място	5 – точки за пилота,	5 – точки за екипа
Седмо място	4 – точки за пилота,	4 – точки за екипа
Осмо място	3 – точки за пилота,	3 – точки за екипа
Девето място	2 – точки за пилота,	2 – точки за екипа
Десето място	1 – точки за пилота,	1 – точки за екипа

4. Титла при конструкторите се печели от екипа събрал най-много точки

5. Титла при пилотите се печели от пилота събрал най-много точки

Всяко състезание се характеризира със следните данни:

1. Брой завои
 - a. Леки завои
 - b. Резки завои
2. Брой прави
 - a. Къса права
 - b. Средна права
 - c. Дълга права
3. Тип настилка
 - a. Асфалт
 - b. Чакъл
 - c. Сняг

Пример:

Рали Швеция е състезание на сняг, което се характеризира с много на брой резки завои, къси и средни по дължина прави.

Рали Мексико е състезание на чакъл, което се характеризира с много на брой дълги прави, леки завои, но с малък брой резки завои.

Всеки автомобил притежава следните характеристики:

1. Сцепление в завой – признак, който указва колко добре автомобилът се представя при леки и резки завои.
2. Еластичност на двигателя – признак, който определя колко добре един автомобил се справя при ускорение.
3. Поведение при различна настилка – признак, който определя колко добре един автомобил се справя при дадена настилка

Пример:

1) Subaru WRX STI е автомобил, който е с нисък център на тежестта и има предимство в завоите и късите прави, но заради боксеровия си двигател изостава на дългите и средни прави. Симетрично задвижване на 4-те колела му дава голямо предимство на чакъл и сняг, но му носи известно забавяне на асфалт.

2) Mitsubishi Lancer Evolution е автомобил, който е също с нисък център на тежестта и има предимство в завоите и дългите и средни прави, но изостава на късите прави. Неговото задвижване на 4-те колела му носи предимство на асфалт и чакъл, но не се представя толкова добре на сняг.

Всеки пилот притежава следните характеристики:

1. Един пилот се справя по-добре при писта с повече завои, друг пилот се справя по-добре на писта с повече прави.
2. Всеки пилот има предимство на една от трите типа настилки (асфалт, чакъл или сняг)

Пример:

- 1.) Себастиен Льоб се справя много добре на състезания с повече завои и на състезания на асфалт и чакъл
- 2.) Петер Солберг се справя много добре на състезания с повече прави и на състезания на чакъл и сняг

Системата трябва да поддържа следните функционалности:

Да извежда резултатите за всяко състезание

Да извежда победителят за сезона при пилотите

Да извежда победителят за сезона при конструкторите

Да извежда крайното класиране(за сезона) при пилотите и конструкторите

Резултатите за всяко състезание трябва да се пазят в файл "races.txt". Всеки пилот трябва да има файл който пази неговите данни в формата "[Pilot_Name.txt]", където Name е името на пилота.

Проект 4: Шах

Да се имплементира популярната игра шахмат.

За целите на проекта ще се изисква само игра между двама играчи. Всеки играч започва със следните фигури:

- 1 - цар;
- 1 - царица;
- 2 - топа;
- 2 - офицера;
- 2 - коня;
- 8 - пешки.

Всяка фигура се мести по различен начин: <https://en.wikipedia.org/wiki/Chess#Movement>

Когато царят на един от играчите е в обсега на противникова фигура, тогава считаме, че играчът е в шах. Единствената възможност на този играч е да направи ход, в резултат от който той няма да бъде повече в шах.

Играта завършва при мат или при съгласие за равенство между двамата играчи (реми).

За правилен ход ще се счита наредената четворка $(x1, y1, x2, y2)$, където: $x1, y1$ - местоположението на фигурата, която играчът иска да премести; $x2, y2$ - новата позиция на фигурата.

Ходът на играча не трябва да приключва, докато той не премести някоя фигура. Примерно представяне на шахматната дъска:

```
8  R N B Q K B N R
7  P P P P P P P
6  - - - - - - -
5  - - - - - - -
4  - - - - - - -
3  - - - - - - -
2  P P P P P P P
1  R N B Q K B N R
   A B C D E F G H
```

Пример за **невалиден** ход:

A, 1, A, 4 = Искаме да преместим топа с три позиции напред, но топът не може да прескача фигури => ходът е невалиден и трябва да подадем нови координати.

Пример за **валиден** ход:

A, 7, A, 5 = Успешно преместваме пешката от A7 на A5.

След всеки успешно изигран ход данните трябва да бъдат запазени на диска, в случай че играта прекъсне(бъде прекратена), при следващо стартиране трябва да възстановено предишното състояние на играта.

Bonus:

В Unicode има дефинирани символи за всяка от фигурите в шаха (бели и черни).
Използвайте ги вместо гореописаните букви.
Направете лог файл, в който да се записват ходовете на двамата играчи. в реда
в които са изиграни.

Проект 5: Бойни кораби

Да се реализира играта Бойни Кораби (подробно обяснени правилата на играта: [https://en.wikipedia.org/wiki/Battleship_\(game\)](https://en.wikipedia.org/wiki/Battleship_(game))) .

Тип на кораба	Живот на кораба	Размер на кораба
Carrier	5	5
Battleship	4	4
Cruiser	3	3
Submarine	3	3
Destroyer	2	2

Всички кораби бият с всеки удар по 1 щета на противниковия кораб

Всеки кораб има специална способност:

Carrier - Способност да бие по две полета на ход .

Battleship - При удар се разкриват всички съседни полета .

Cruiser - Ще възстановява по 1 живот на ход, ако не е уцелен в този ход или не е потопен.

Submarine - 70 процента шанс за пропуск на противников удар .

Destroyer - Щит, който спестява еднократен удар от противника .

Играта трябва да се играе между човек и бот. За бота(AI) трябва да се реализира елементарна логика на игра.

Карта:

1	2	3	4	5	6	7	8	9
2
3	.	X	.	.	X	X	.	.
4	.	.	@	O	O	.	.	.
5
6
7	O	O	X	X	X	.	.	.
8
9

Легенда:

Символ	Дефиниция
.	място на картата, което не е било бомбардирано
1	позиция на картата
O	пропуск
X	удар в целта
@	пропуснат удар или удар по щит на кораб

Всеки един от играчите притежава карта и стреля по картата на противника. На всеки ход, играча има правото да стреля само с един кораб, като втори изстрел с същия кораб се позволява, ако е играно с останалите 4 кораба (тоест след 4 хода).

След всеки успешно изигран ход данните трябва да бъдат запазени на диска, в случай че играта прекъсне(бъде прекратена), при следващо стартиране трябва да възстановено предишното състояние на играта.