Проекти по обектно-ориентирано програмиране

Правила и критерии за оценка

Проектите да бъдат предавани под формата на ZIP файл, съдържащ:

- изходни файлове (.cpp)
- заглавни файлове (.h)
- използвани библиотеки (ако има такива)
- документация на проекта в PDF формат

Всеки проект може да бъде избиран от не повече от 8 пъти. Срокът за избор на проект е 7 май 2014 г. Срокът за предаване на проект е до края на семестъра.

Всеки проект трябва да бъде защитен в рамките на 15 мин. При установено взаимстване от друг източник или недостатъчно познаване на кода проектът се анулира.

Всеки проект ще бъде оценяван по следните критерии:

- коректност на решението
- уместно използване на концепциите на ООП
- подреденост и четимост на кода
- документация на проекта и кода
- представяне на проекта при защитата

Проекти

Матрици и вектори

Да се напишат подходящи класове, които позволяват операции с матрици и вектори от обекти от произволен тип. Класовете да поддържат следните операции:

1. Матрици

- въвеждане от входен поток
- извеждане към изходен поток
- операции за събиране и изваждане на матрици чрез операциите +, -, +=, -=
- добавяне и изваждане на число към всички елементи на матрицата чрез операциите +, -, +=, -=
- умножение на матрици с подходящи размерности операция *, *=
- умножение и деление на матрици с числа с операции *, *=, /, /=

- достъп до елементи с операция []
- транспониране на произволна правоъгълна матрица с операция ~
- намиране на детерминанта на матрица
- намиране на обратна матрица с операция !

2. Вектори

Да се реализират следните операции за работа с вектори:
- въвеждане от входен поток
- извеждане към изходен поток
- операции за събиране и изваждане на вектори чрез операциите +, -, +=, -=
- събиране, изваждане, умножение и деление на вектор с число с операции +, +=, -, -=, *, *=, /, /=
- достъп до елементи с операция []
- скаларно и векторно произведение на вектори чрез операции *, *=, ^, ^=
- нормализиране на вектор с операция !
- умножение на вектор с матрица с операции *, *=

Да се напише подходяща примерна програма, която демонстрира работата с класовете.

Бонуси:
- да се поддържат n-мерни матрици (т.е. с произволен брой размерности)
- да се реализира клас “подматрица” с подходящи операции, който позволява достъп до произволна правоъгълна подобласт от дадена матрица
- да се реализира клас “курсор”, който позволява различни обхождания на матрицата

Големи числа

Да се реализира клас, който представя “големи числа”, т.е. цели числа, чиято дължина не е предварително ограничена, като на примитивните типове short, int, long. Числата могат да са отрицателни или положителни. За класа да се реализират:
- подходящи конструктори с “обикновени” числа
- голяма четворка
- операция за въвеждане (>>) от стандартния вход на голямо число в:
    - десетична бройна система
    - шеснайсетична бройна система (ако започва с 0x)
- операция за извеждане (<<) на голямо число на стандартния изход в:
    - десетична бройна система
    - шеснайсетична бройна система
- аритметични операции: +, +=, -, -=, *, *=, /, /=, %, %=
- операции за сравнение: ==, !=, <, >, <=, >=

- операция [] за получаване на поредна цифра

Да се напише подходяща примерна програма, която демонстрира работата с класа.

Бонуси:

- Да се реализират не само цели, но и десетични дробни числа с произволен краен брой цифри (няма нужда да се поддържат числа с безкраен запис, например периодични дроби или π)
- Да се реализират побитовите операции: &, &=, |, |=, ~, ^, ^=, <<, <<=, >>, >>=

Геометрия

Да се реализират класове, които представят точка, вектор, права и равнина в тримерното пространство.
Точка да може да се конструира като

- тройка координати
- сечение на две дадени прави
- сечение на дадена права с дадена равнина

Вектор да може да се конструира като:

- тройка координати
- дадени две точки, задаващи начало и край на вектора
- нормала на дадена равнина

Права да може да се конструира като:

- минаваща през две дадени точки
- перпендикулярна на дадена равнина в дадена точка
- сечение на две дадени равнини
- минаваща през дадена точка и успоредна на дадена вектор

Равнина да може да се конструира като:

- зададена с общо уравнение с четири коефициента
- успоредна на дадена равнина и минаваща през дадена точка
- перпендикулярна на дадена права в дадена точка
- успоредна на два дадени вектора
- минаваща през три дадени точки

Да се реализират следните операции:

- извеждане на общо уравнение на дадена равнина
- извеждане на параметрично уравнение на дадена равнина
- извеждане на параметрично уравнение на дадена права
- представяне на права като сечение на две равнини

- транслиране на точка с вектор
- сумиране на вектори
- скаларно и векторно произведение
- проверка дали точка лежи на дадена права
- проверка дали точка лежи на дадена равнина
- проверка за взаимното положение на дадена права и дадена равнина (успоредни, пресичащи се)
- проверка за взаимното положение на две прави (успоредни, пресичащи се, кръстосани)
- проверка за взаимното положение на две равнини (успоредни, пресичащи се)

Бонуси:

- реализиране на клас за триъгълник, зададен с три точки и реализиране на операции:
    - селектор за равнината на триъгълника
    - селектори за страните на триъгълника като прави
    - селектори за височини, медиани, ъглополовящи и симетрали на триъгълника
    - селектори за ортоцентър, медицентър и център на описаната и вписаната окръжност
- реализиране на следните допълнителни операции:
    - построяване на ортогонална проекция на права върху равнина
    - намиране на ъгъл между права и равнина
    - намиране на разстояние между точка и права
    - намиране на разстояние между точка и равнина
    - намиране на разстояние между кръстосани прави

Да се напише подходяща примерна програма, която демонстрира работата с класа.

Дати

Да се напише клас, който да представя дата от Григорианския календар с ден, месец и година. За класа да се реализират:

- подходящи конструктори с проверка за коректност
- селектори за ден, месец и година
- селектор за ден от седмицата
- мутатори за ден, месец и година с проверка за коректност
- операция за въвеждане от стандартния вход (>>) с проверка за коректност
- операция за извеждане на стандартния изход (<<) в предварително зададен от потребителя формат
- конструктор на дата по описание с ден от седмицата, например:
    - втората събота от месец октомври
    - последният понеделник от месец ноември

- аритметични операции с дати:
    - +, +=, ++ (добавяне на дни)
    - -, -=, -- (изваждане на дни)
    - *, *= (добавяне на месеци)
    - /, /= (изваждане на месеци)
    - &, &= (добавяне на години)
    - |, |= (изваждане на години)
    - ^, ^= (добавяне на седмици)
    - %, %= (изваждане на седмици)
    - да се обработват правилно граничните случаи, например 31 януари 2012 г. + 1 месец = 29 февруари 2012 г.
- аритметични операции за намиране на интервал между дати
    - - (разлика в дни)
    - / (разлика в месеци)
    - | (разлика в години)
    - % (разлика в седмици)
- селектор, който определя дали даден ден е работен
    - да се заложи списък на официалните празници в България, като се вземе предвид алгоритъма за определяне на датата на Великден
- селектор, който намира най-близкия работен ден до текущата дата (да се подава параметър, който дали определя дали се търси назад във времето, напред във времето или без значение от посоката)
- мутатор, който променя режима на работа на аритметичните операции +, +=, -, -=, така че да смятат с работни, а не с календарни дни
    - пример: 15 април 2014 г. + 4 работни дни = 23 април 2014 г.
- мутатор, който променя режима на работа на всички аритметични операции, така че ако след изпълнение на операцията резултатът е почивен ден, резултатът да се “наглася” да бъде най-близкия работен ден
    - пример: 15 април 2014 г. + 4 дни = 17 април 2014 г. (ако е избрано търсене назад)
    - пример: 15 април 2014 г. + 4 дни = 22 април 2014 г. (ако е избрано търсене напред)
    - пример: 15 април 2014 г. + 4 дни = 17 април 2014 г. (ако е избрано търсене без значение на посоката)

Не е позволено използването на системна библиотека за работа с дати.

Да се напише подходяща примерна програма, която демонстрира работата с класа.

Бонуси:

- да се добави възможност за задаване на допълнителни почивни дни, които се отработват (напр. петък, 2 май 2014 г. е почивен ден, който ще се отработва в

събота, 10 май 2014 г.), като изключенията се вземат предвид при пресмятанията, които зависят от работни/почивни дни

- да се реализира калкулатор за аритметични операции с дати, който чете низ от символи и го разглежда като израз с дати и връща резултата. Да се реализира възможно най-гъвкав формат за въвеждане на изрази. Примери:
  - днес + 2 дни
  - 30 април 2014 г. + 3 работни дни
  - утре - 1 седмица + 3 години
  - 30 април 2014 г. - 5 април в работни дни
  - вчера - 1 година - (5 януари 2012 г. + 1 месец) в седмици

Кодирания RLE и Base64

1. RLE

Run-length encoding (RLE) представлява кодиране на низове, при което последователности от еднакви символи се заменят с двойка от символа и броя повторения. Например S = AAAABBBBCCCAABBBB може да се представи с RLE списъка R = (4,A) (4,B) (3,C) (2,A) (4,B). Да се напише клас, който представя низ от символи чрез RLE кодиране. За класа да се реализират:

- конструктор по низ (кодиране)
- операция за преобразуване до тип низ (декодиране)
- операция за извеждане (<<)
- голяма четворка
- операция за индексиране [], която да връща пореден символ в даден RLE списък без да го декодира
- операции +, += за конкатениране на два RLE списъка
- операция ++ (в префиксен и постфиксен вариант), който добавя към даден RLE списък още един символ, който е същият като последният в него. Ако операцията се изпълни над празен списък, той не се променя
  - Пример: след R++, R става (4,A) (4,B) (3,C) (2,A) (5,B), резултатът е старата стойност на R
  - Пример: след ++R, R става (4,A) (4,B) (3,C) (2,A) (5,B), резултатът е самото R (с новата си стойност)
- операция -- (в префиксен и постфиксен вариант), която премахва последния символ от списъка. Ако операцията се изпълни над празен списък, той не се променя.
  - Пример: след R--, R става (4,A) (4,B) (3,C) (2,A) (3,B), резултатът е старата стойност на R
  - Пример: след --R, R става (4,A) (4,B) (3,C) (2,A) (3,B), резултатът е самото R (с новата си стойност)
- операция () за намиране на подсписък от дадена позиция и с дадена дължина
  - Пример: R(7,5) да върне (1, B) (3, C) (1, A)

- операция () за вмъкване на един RLE списък на произволна позиция в друг като позицията се указва в брой символи **от началото на декодирания низ**.
    - Пример: ако P = (3, C) (1, B), тогава R(6, P) да получава списъка (4,A) (2, B) (3, C) (3, B) (3, C) (2, A) (4, B)
- операция, която изтрива последователност от символи от дадена позиция
    - Пример: след като изтрием последователност от 8 символа от позиция 6 в R се получава списъкът (4,A) (5, B)
- операции ==, != за сравнение на два списъка
- операции A < B, A <= B, B > A, B >= A, който проверяват дали A е (строг) подсписък на B, т.е. низът, кодиран от RLE списъка A е (строг) подниз на низа, кодиран от RLE списъка B.
    - пример: списъкът (2, B) (1, C) е подсписък на R
    - пример: списъкът (1,C) (3,A) (1, B) не е подсписък на R (4,B)
- едноместна операция *, която по дадено RLE кодиране на низ, да построява честотната таблица на низа (за всеки символ, който се среща в низа, се указват броят на срещанията му в низа). Резултатът да е под формата на RLE списък, в който всеки символ се среща най-много веднъж
    - пример: *R да връща (6,A) (8,B) (3,C)

За всяка една от горните операции е забранено да се извършва декодиране и повторно кодиране на RLE списък, те трябва да се извършват директно над вътрешното RLE представяне с цел ефективност.

2. Base64

Да се реализира клас, който представя масив от байтове (unsigned char) като низ с произволна дължина, съдържащ само символите “A-Za-z0-9+/”, чрез така нареченото Base64 кодиране, както е описано по-долу.

Целта на кодирането е да се избегнат специалните символи като ‘\t’, ‘\n’, ‘\0’ и други. Забележете, че броят на символите в множеството “A-Za-z0-9+/” е $2^6 = 64$ (= 26+26+10+2). Така всеки символ носи 6 бита информация. Следователно, всеки 3 байта = 24 бита се представят с 4 символа.

Алгоритъм на кодиране:

1. разглеждаме входния двоичен масив като последователност от групи по 24 бита (три байта от по 8 бита)
2. всеки 24 бита разбиваме на четири 6-битови байта
3. на всеки 6-битов байт съпоставяме съответния символ от списъка “A-Za-z0-9+/”
4. записваме получените символи в текстовия файл

Имаме от един от следните случаи:

1. броят байтове е кратен на 3, тогава всичко е ОК
2. броят байтове дава остатък 1 при деление на 3. Тогава при последното четене са останали само 8 бита. Допълваме до 12 бита с нули и кодираме с два символа. Допълваме с два символа ‘=’ (padding), така че общият брой на символите да се дели на 4.
3. броят байтове дава остатък 2 при деление на 3. Тогава при последното четене са останали само 16 бита. Допълваме до 18 бита с нули и кодираме с три символа Допълваме с един символ ‘=’ (padding), така че общият брой на символите да се дели на 4.

Това кодиране се нарича base64 и се използва при прикрепяне на файл към e-mail съобщение, заради изискванията на протокола да не се използват специални символи. Резултатът от кодирането е текстов файл с размер 4/3 спрямо оригиналния, но без специални символи и върши работа за всякакви файлове. Недостатък е, че съдържанието на кодирания файл е неразбираемо за човек. Съществува и друго широко използвано кодиране наречено quoted-printable – като при него се разчита, че се изпраща главно текст и малък брой специални символи, които се escape-ват.

Подробна спецификация на base64 и quoted-printable кодиранията можете да намерите в секции 6.7 и 6.8 на rfc2045 (например на адрес http://www.faqs.org/rfcs/rfc2045.html).

За класа да се реализират:

- конструктор по масив от байтове
- операция за преобразуване до тип масив от байтове
- голяма четворка
- операция за извеждане (<<)
- операция за индексиране [], която да връща пореден /айт в оригиналния масив
- операции +, += за конкатениране на два base64-кодирани масива
- операция () за намиране на base64-кодиран подмасив от дадена позиция и с дадена дължина
- операция () за вмъкване на base64-кодиран масив на произволна позиция в друг като позицията се указва в брой байтове **от началото на декодирания масив**.
- операция, която изтрива последователност от байтове от дадена позиция
- операции A < B, A <= B, B > A, B >= A, който проверяват дали A е (строг) подмасив на B, т.е. масивът, base64-кодиран от низа A е (строг) подмасив на низа, base64-кодиран в низа B.

Не се позволява използването на стандартни библиотеки и готови решения!

Да се напише подходяща примерна програма, която демонстрира работата с класовете.

Бонуси:

- да се реализира възможност на двата класа да работят с произволен входен поток вместо с фиксиран низ/масив от байтове
- да се реализира възможност за “смесване” на операции от двата типа кодиране, например вмъкване на RLE-списък в base64-кодиран подмасив или сравнение на RLE-списък с base64-кодиран масив

XML Parser

Да се реализира клас, който представя XML елемент, който съдържа следната информация:

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

Класът да поддържа следните операции:

- конструктор с низ или входен поток, съдържащ валиден XML
- операция за извеждане (<<) в текстов вид на стандартния вход
- селектори:
  - проверка за наличие на атрибут с определен ключ
  - достъп до атрибут с определен ключ
  - брой вложени елементи
  - достъп до елемент на дадена позиция
  - достъп до вложен текст
- мутатори:
  - добавяне на атрибут
  - изтриване на атрибут
  - добавяне на вложен елемент на дадена позиция
  - изтриване на вложен елемент на дадена позиция
- 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]” два 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”)

Да се напише подходяща примерна програма, която демонстрира работата с класовете.

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,...)

Детерминиран краен автомат

Да се реализира клас, който представя детерминиран краен автомат над азбука, състояща се от цифрите и малките латински букви. За класа да се реализират:

- подходящ конструктор
- операция (<<) за извеждане на информация за автомата
- (private) разширена функция на прехода
- булева функция, която проверява дали дадена дума е в езика на автомата
- мутатор, който добавя “error” състояние към автомата, за да го превърне в тотален
- обединение на два автомата
- сечение на два автомата
- допълнение на автомат
- операция за преобразуване на автомат до низ, съдържащ регулярен израз (теорема на Клини)

Бонуси:

- да се реализира операция, която намиран минимален на дадения автомат и операция == за сравнение на два автомата (два автомата са еквивалентни, ако минималните им автомати съвпадат)
- да се реализират операции, които проверяват дали езикът на даден автомат е:
    - празен
    - краен
    - пълен

Недетерминиран краен автомат

Да се реализира клас, който представя недетерминиран краен автомат с ε-преходи. над азбука, състояща се от цифрите и малките латински букви. За класа да се реализират:

- подходящ конструктор
- операция (<<) за извеждане на информация за автомата
- (private) разширена функция на прехода
- булева функция, който проверява дали автомат е детерминиран
- булева функция, която проверява дали дадена дума е в езика на автомата
- обединение на два автомата
- конкатенация на два автомата
- позитивна обвивка на автомат
- конструктор на автомат по даден регулярен израз (теорема на Клини)

Бонуси:

- да се реализира мутатор, който детерминира даден автомат
- да се реализират операции, които проверяват дали езикът на даден автомат е:
    - празен
    - краен

Стеков автомат и контекстно-свободна граматика

Да се реализира клас, който представя контекстно-свободна граматика, използваща главни латински букви за променливи (нетерминали) и малки латински букви и цифри за терминали. За класа да се реализират:

- подходящ конструктор
- операция (<<) за извеждане на информация за граматиката
- мутатори за добавяне и премахване на правила
- селектор, който проверява дали дадена граматика е в нормална форма на Чомски
- функция, която проверява дали дадена дума е в езика на дадена граматика (CYK алгоритъм)
- обединение на две граматики
- конкатенация на две граматики
- итерация на граматика

Да се реализира клас, който представя недетерминиран стеков автомат. За класа да се реализират:

- подходящ конструктор
- операция (<<) за извеждане на информация за автомата
- функция, която проверява дали дадена дума се разпознава от стековия автомат
- конструктор на стеков автомат по контекстно-свободна граматика

Бонуси:

- да се реализира операция, която проверява дали езика на дадена контекстно-свободна граматика е празен
- да се реализира операция, която проверява дали езика на дадена контекстно-свободна граматика е краен
- мутатор, който преобразува граматика в нормална форма на Чомски
- конструктор на контекстно-свободна граматика по стеков автомат с единствено състояние

Машина на Тюринг

Да се реализира клас, който представя потенциално безкрайна двупосочна лента от малки латински букви, цифри и интервал. Лентата да бъде реализирана чрез масив, който може да се разширява в двете посоки при нужда.

Да се реализира клас, който представя детерминирана машина на Тюринг с потенциално безкрайна лента, първоначално инициализирана с интервали. За класа да се реализират:

- подходящ конструктор
- операция (), която реализира функцията над думи, пресмятана от машината
- композиция на две машини на Тюринг
- разклонение на две машини на Тюринг относно трета
- функция, която връща машина на Тюринг, реализираща while-цикъл над дадената машина на Тюринг

Бонуси:

- да се реализира многолентова машина на Тюринг. Да се реализира конструктор на еднолентова машина на Тюринг по дадена многолентова машина на Тюринг
- да се реализира клас, представящ недетерминирана машина на Тюринг. Да се реализира конструктор на детерминирана машина на Тюринг по дадена недетерминирана такава
- да се реализира операция, която намира Гьоделовия номер на дадена машина на Тюринг (добре е да се използва клас представящ големи числа). Да се конструира машина, която генерира Гьоделовия си номер над празна лента