Софийски университет „Св. Климент Охридски“
Факултет по математика и информатика

Писмен изпит

курс Структури от данни и програмиране
за специалност Компютърни науки, поток 2
зимен семестър 2021/2022 г.

Основните критерии при оценяването ще бъдат:

- успешно изпълнение на поставеното условие;
- използването на най-подходящите структури от данни;
- добро стилизиране и форматиране на кода;
- сложности;
- следване на добри практики за писане на код;
- **спазване на ООП парадигмата**
- **за съответните стурктури от данни може да се използва STL**

Задача 1 (1.0)
Иван е продавач-консултант. В магазина, в който работи има един огромен рафт с купчини сгънати дрехи. Рафтът може да съдържа различен брой купчини от дрехи. Всяка купчина дрехи е независима от останалите и се характеризира с това, че всяка нова сгъната дреха може да се постави на **произволно място** в купчината. Всяка дреха има цена и цвят, който е един от изброените - жълт, червен, зелен, син, оранжев и се означават с един символ, съответно - ‘y’, ‘r’, ‘g’, ‘b’, ‘o’.
А) Да се напише функция `printShelf`, която приема като параметър рафта в магазина(`[подходящ_тип]shelf`) и извежда на екрана дрехите с техните цветове и цени, разделени със знака ‘_’, по подобие на действителната им подредба върху рафт, а имменно:

- на първия ред се извеждат дрехите, които са на върха на купчините. Ако има купчина, която е с по-малко количество дрехи, за нея не се отпечатва нищо.
- на втория ред се извеждат дрехите, които се намират под тези на върха, ако в съответната купчина има дреха на тази позиция и т.н.
- на последния ред се извеждат дрехите, които са най-отдолу на купчините.

В) Да се напише функция, която приема списък от допустими цветове(`acceptableColorsByPiles`), минимална(`minPriceByPiles`) и максимална цена(`maxPriceByPiles`) за всяка купчина на рафта, рафтът с всички купчини от дрехи (`shelf`) и пренарежда дрехите, както следва:

- във всяка купчина да няма дреха, която е в различен цвят от посочените за допустими цветове в списъка за съответната купчина.

- да няма две последователни купчини, които на дъното си да имат две дрехи в един и същ цвят.
- да няма две последователни дрехи в един цвят, в една и съща купчина
- цената на дрехите във всяка една купчина да е в затвореният интервал [`minPriceByPiles; maxPriceByPiles`]

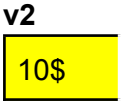
Всички дрехи, които нарушават посочените условия се премахнат от съответните купчини. Складът на магазина е огромен, ако има нужда от допълнителни дрехи, за да бъде удовлетворено дадено свойство, от него може да се извади нова дреха в желан от нас цвят и цена.

Пример:
Разглеждаме рафт с две купчини от дрехи
А) Функцията `print` отпечатва
y_10$
r_6$
y_4$
g_15$
r_3$ b_3$
g_1$ r_15$
y_17$ y_12$
r_20$ b_25$

В) За всяка от двете купчини се подават следните допустими цветове и ценови интервали:
v1

pile 1 pile 2
[‘y’,‘r’,‘g’] [‘y’,‘r’,‘b]’ `acceptableColorsByPiles`
[20,100] [50,70] [`minPriceByPiles; maxPriceByPiles`]
Наредбата е коректна

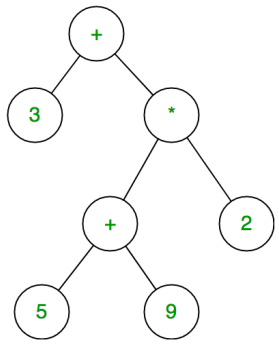
v2



['y','r','g'] ['y','r','b'] `acceptableColorsByPiles`
[20,100] [50,70] [`minPriceByPiles; maxPriceByPiles`]
Наредбата не е коректна:

- в първата купчина има две дрехи с един цвят - една от двете дрехи може да бъде премахната.
- във втората купчина дрехите не са в допустимия ценови интервал - от склада ще извадим нова дреха в жълт или червен цвят, която е на стойност 7$.

Задача 2 (1.0) Дадено е двоично дърво, което представя израз. В корена и междинните върхове на дървото са аритметичните оператори събиране, изваждане, умножение и деление, а в листата се намират различните операнди - **реални числа**. Предполага се, че подизразите в лявото и дясното поддърво са оградени в скоби и операциите в тях се извършват с по-висок приоритет.
Пример:



Дървото е еквивалентно на следния израз - 3.0 + ((5.0+9.0) * 2.0)

Двоичното дърво се представя, чрез указател към следната структура:

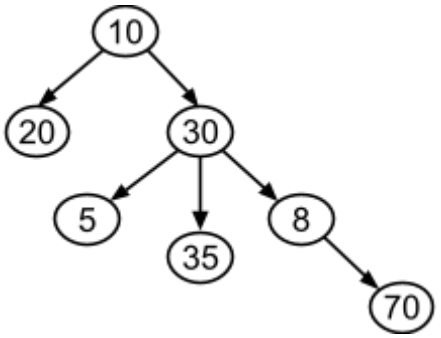
```
template <typename T>
struct BinTreeNode {
        T data;
        BinTreeNode<T>* left;
        BinTreeNode<T>* right;
};
```

Да се напише функция, която приема указател към корен на двоично дърво от описания вид, реално число `result` и константа `c` и намира поддървото, чийто резултат от пресмятането на израза, който е записан в него, е равен на `result` с точност до константата `c`. Функцията да връща указател към корена на това поддърво, ако то съществува или `nullptr`, ако такова дърво не съществува.

Пример:

Дърво с резултат 27.95 и с точност 0.1 е поддървото с корен `'*'`.

Задача 3 (2.0) Дадено е дърво от неповтарящи се цели числа - tree. **Дървото се представя като свързан граф без цикли чрез списък на наследниците.**



- Да се напише функция, която създава дърво по подаден валиден низ от вида: "`[r<интервал><t1>...<интервал><tk>]`", където `r` е корен на дървото, а `<t1>`… `<tk>` са представянията на съответните поддървета.
  Пример: дървото по-долу има представяне "`[10 [20] [30 [5] [35] [8 [70]]]]`"
- "Програма" в дървото се нарича серия от последователни инструкции, описващи обхождане на възли в дървото, започващо от корена. Допустими са следните инструкции:
  - "`(x)`" където `x` е положително цяло число: изисква преминаване към **прекия** наследник с пореден номер `x` в списъка с наследници, ако такъв съществува и извежда съобщение за грешка в противен случай.
  - "`e(x)`", където `x` е цяло положително число: изисква преминаване към първия наследник, чиято стойност е четно число в списъка с наследници, ако такъв съществува и извежда съобщение за грешка в противен случай.
  - "`>x`", където `x` е цяло число: изисква извеждане на всички елементи на поддървото с корен текущия възел (не само преки наследници), които са по-големи от `x`.

Да се реализира функция `void compile (char*,[подходящ параметър]t, [подходящ параметър] rootPtr)`, която по списък с инструкции, разделени една от друга с интервали, дърво t и/или указател към корена `rootPtr`, изпълнява последователно всички инструкции над дървото t.

Пример: Ако е дадено дървото на фигурата, то заявката "`(2) >8`" отпечатва "`35 70`".

Упътване:

В инструкцията “`e(x)`”, стойността се задава само за консистентност на инструкциите, но не се използва и не се изисква композиция на инструкции.