



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

## Домашна работа 2

курс Функционално програмиране  
за специалности Информатика и Компютърни науки (2-ри поток)  
зимен семестър 2020/21 г.

### Редакции

**2020-12-02** Корекция и допълване на примера с представяне на дърво. Добавено е пояснение за балансирането по височина. Тъй като все още не сме покрили материала за потоци, функцията `tree->stream` остава като незадължителна, бонус задача.

**2020-11-22** Добавено пояснение за типа на елементите във възлите на дървото.

**2020-11-15** Добавено пояснение за празните (whitespace) символи в представянето на дърво.

### Изисквания

Освен на това, което е описано в документа “Схема за оценяване”, решенията ви трябва да отговарят и на дадените по-долу изисквания:

Решението си организирате в два файла по следния начин:

- `tree.rkt`, който съдържа кода на решението;
- `tree-test.rkt`, който съдържа unit test-овете към решението.

Задачата трябва да се реши на език `racket/base`. За да можете да работите с потоци, във файла с решението включете `racket/stream`. Във файла с unit test-овете включете `rackunit` и `rackunit/gui`.

# Описание

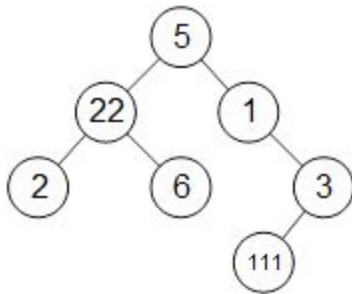
Двоично дърво ще представяме като символен низ по следния начин:

- Празното дърво се представя със символа звезда.
- Дърво с корен N и наследници L и R представяме като "{N L R}".

В програма на Scheme ще представяме дърво по следния начин:

- Празното дърво се представя като празния списък ().
- Дърво с корен N и наследници L и R представяме като списъка (N L R).

По-долу е даден пример за дърво и неговите представяния:



Представяне като символен низ:

"{5 {22 {2 \* \*} {6 \* \*}} {1 \* {3 {111 \* \*} \*}}}"

**Редакция (2020-12-02):** Корекция и допълване на примера с представяне на дърво:

Възможно представяне в Scheme:

```
(quote
  (5 (22 (2 () ()) (6 () ())) (1 () (3 (111 () ()) ())))
)
```

Друг начин да се реализира същото представяне:

```
(list 5
      (list 22
            (list 2 '() '())
            (list 6 '() '()))
      (list 1
            '()
            (list 3
                  (list 111 '() '())
                  '())))
```

**Редакция (2020-11-22):** За целите на задачата, можем да считаме, че елементите във възлите на дървото ще бъдат произволни естествени числа.

Реализирайте дадените по-долу функции. Всички освен `string->tree` и `visualize` трябва да се покриват с подходящи unit test-ове.

```
(tree? str)
```

Проверява дали подаденият `str` като аргумент символен низ е коректно представяне на дърво.

**Редакция (2020-11-15):** За определеност, допускаме, че потребителят може да постави произволен брой (вкл. нула) празни символи между отделните елементи на един възел. Например следните са валидни представяния на едно и също дърво:

```
{2 {4 * *} *}
{2{4**} *}
{ 2      {4      * * }          * }
```

```
(string->tree str)
```

Конструира и връща двоично дърво по подаденото му текстово представяне `str`.

Функцията трябва да направи проверка за коректност на въведените данни (използвайте наготово `tree?`). В случай, че низът не е валидно представяне на двоично дърво, да се върне стойност `#f`.

```
(balanced? tree)
```

Проверява дали двоичното дърво `tree` е балансирано по височина. Връща `#t` или `#f`.

**Редакция (2020-12-02):** Напомняме, че дефиницията за “балансирано дърво” зависи от някакъв критерий, който определя какво точно разбираме под “баланс” в дървото. В случая в задачата се изисква балансиране по височина. Напомняме, че това означава, че за всеки възел в дървото трябва да е изпълнено, че разликата във височините на лявото и дясното му поддърво трябва да е най-много 1. По-конкретно, ако с  $h$  означим височината на дадено дърво, а  $l$  и  $r$  връщат съответно лявото и дясното поддърво на даден възел, условието ще изглежда така:

за всеки възел  $n$  трябва да бъде изпълнено:  $|h(l(n)) - h(r(n))| \leq 1$

Това не трябва да се бърка с балансирането по тегло, при което съществен е броят на възлите, а не височината!

```
(ordered? tree)
```

Проверява дали двоичното дърво `tree` е двоично наредено дърво. Връща `#t` или `#f`.

```
(tree->string tree)
```

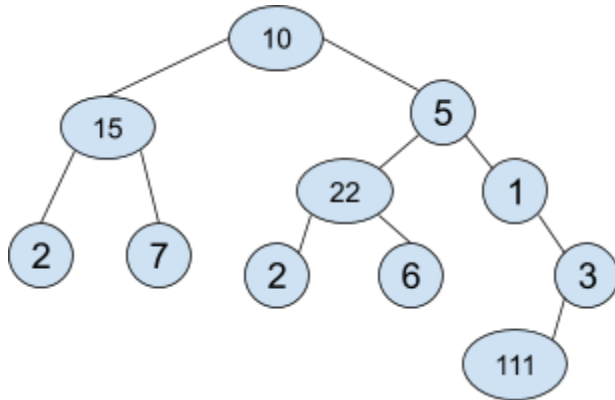
Преобразува двоичното дърво `tree` до неговото представяне като символен низ.

**Редакция (2020-11-15):** Когато преобразувате дърво до символен низ, трябва да слагате точно по един интервал между елементите на възлите и да няма разстояние между тях и фигурните скоби. Например `{2 {4 * *} *} *` е валидна стойност, която `tree->string` може да върне, а следните НЕ СА:

```
{ 2 { 4 * * } * }
{2{4**}*}
{2      {4      * *}      *}
```

`(visualize tree)`

Визуализира двоичното дърво `tree`. При визуализацията, левият наследник на даден възел трябва да се покаже точно под него, а десният му -- вдясно от него. По-долу е показана визуализацията на дървото от нашия пример. Забележете, че дължината на дъгите, които излизат от всеки възел зависи съответно от височината на десния и широчината на левия му наследници:



```

10-----5-----1--3
|         |         |
|         |         111
|         |         |
|         22--6
|         |
|         2
15--7
|
2
```

**Редакция (2020-12-02):** Тъй като все още не сме покрили материала за потоци, тази функция остава като незадължителна, бонус задача:

`(tree->stream tree order)`

Преобразува двоичното дърво `tree` до поток от неговите елементи. Редът на обхождането се определя от параметъра `order`. Той може да бъде един от символите `'inorder`, `'postorder` или `'preorder`. Справка за редовете на обхождане можете да направите в тази статия на Wikipedia: [https://en.wikipedia.org/wiki/Tree\\_traversal](https://en.wikipedia.org/wiki/Tree_traversal)

Например:

- Поток с inorder ред на обхождане, за дървото от примера по-горе, би върнал следната последователност: 2, 22, 6, 5, 1, 111, 3
- Поток с inorder ред на обхождане, за дървото от примера към функцията `visualize` по-долу, би върнал следната последователност: 2, 15, 7, 10, 2, 22, 6, 5, 1, 111, 3