Задачи за задължителна самоподготовка по Обектно-ориентирано програмиране

Функции от високо ниво, шаблони, виртуални методи

email: kalin@fmi.uni-sofia.bg

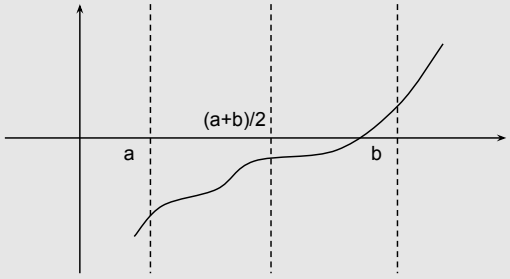
28 май 2016 г.

1. Да се дефинира функция `double root ([подходящ тип]f, double a, double b, double e)`, където $f : double \to double$ е непрекъсната и монотонна в интервала $[a, b]$ и притежава корен в него, а e е положително число. Чрез използване на двоично търсене (*bisection*), функцията `root` да намира приближение на корена на f в интервала $[a, b]$ с грешка най-много e.

   Упътване:

   Установете дали функцията е растяща или намаляваща. Да приемем, че функцията е растяща. За намаляващи функции алгоритъмът е аналогичен.

   За всеки интервал $[a, b]$ имаме точно три възможни случая:



(а) $|f(\frac{a+b}{2})| < e$. В този случай приближението е намерено и то е $\frac{a+b}{2}$

(б) $f(\frac{a+b}{2}) < 0$. В този случай търсим корена на функцията в интервала $[\frac{a+b}{2}, b]$

(в) $f(\frac{a+b}{2}) > 0$. В този случай търсим корена на функцията в интервала $[a, \frac{a+b}{2}]$

- Дефинирайте два варианта на функцията: итеративен и рекурсивен.
- Тествайте функцията `root` с поне два примера.

2. Да се дефинира функция `void zip (double a1[], double a2[], double res[], int n, [подходящ тип]f)`, където `a1`, `a2` и `res` са масиви с `n` на брой елементи, а `f` е функция от тип $f : double \times double \to double$. Като резултат от работата на функцията елементите на `res` да съдържат стойностите на функцията `f` върху съответните елементи на `a1` и `a2`, така че $res[i] = f(a1[i], a2[i])$ за $i = 0..n-1$.

   - Тествайте функцията с поне два примера.

3. Функцията `zip` от предишната задача да се преобразува до шаблон, така че масивите `a1`, `a2` и `res` да са от произволен тип `T`.

   - Тествайте функцията с поне два примера.

4. Функцията `zip` от предишната задача да се преобразува до шаблон, така че всеки от масивите `a1`, `a2` и `res` да са от различни помежду си типове T_1, T_2 и T_3, а $f : T_1 \times T_2 \to T_3$.

   - Тествайте функцията с поне два примера.

5. Към разработената на лекции йерархия от изчислими изрази да се добави клас, представящ оператора умножение.

6. Към разработената на лекции йерархия от изчислими изрази да се добави клас `IfExpression`, представящ оператора разклонение (`if`). Операторът да зависи от три израза - `cond` (условие), `then_expr` (израз в случай на вярно условие) и `else_expr` (израз в случай на невярно условие). Стойността на оператора `if` да е стойността на израз `then_expr` или `else_expr` в зависимост от стойността на израза `cond`.

   Упътване: Ако се налага, може да разширите клас `Value` с метод за проверка на това дали съответната стойност е истинна или не.

   Пример:

   Нека `c`, `t` и `e` са обекти от някой наследник на клас `Expression`, като стойността на `c` е различна от 0. Тогава, при конструиране на обекта `ife` по следния начин: `IfExpr ife (&c,&t,&e)`, то стойността на `ife.execute()` ще е същата като на `t.execute()`.

7. Към разработената на лекции йерархия от изчислими изрази да се добави клас `ArithExpression`, представящ едновременно четирите аритметични оператора `+`, `*`, `-` и `/`. При конструиране на обектите да се уточнява конкретният оператор чрез един от символите `'+'`, `'*'`, `'-'` и `'/'`, съответно. Например, следният обект: `ArithExpr ae ('*',&e1, &e2)` да представя умножение на изразите, представени от `e1` и `e2`.

8. Към разработената на лекции йерархия от изчислими изрази да се добави допустим тип на изразите `char`. Т.е. да се дефинира нов наследник на клас `Value`, представящ символния тип.

9. Към разработената на лекции йерархия от изчислими изрази да се добави допустим тип на изразите символен низ. Т.е. да се дефинира нов наследник на клас `Value`, представящ символен низ.