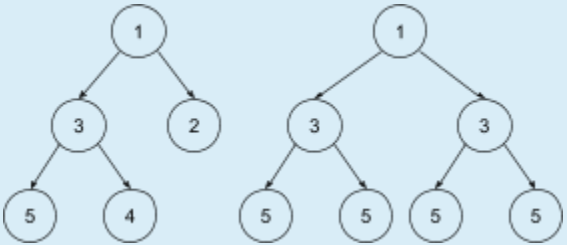
Задача 7: Двоичното дърво t_2 наричаме “ляво копие” на t_1, тогава и само тогава, когато:

- И двете дървета са празни, или
- Ако t_1 е дърво с корен x, ляво поддърво L_1 и дясно поддърво R_1,

- а t_2 е дърво с корен y, ляво поддърво L_2 и дясно поддърво R_2,
  то x==y, а L_2 и R_2 са леви копия на L_1.
-

Нека е дадена следната структура, описваща възел в двоично дърво:
`struct Node { int x; Node *left, *right;};`
Да се дефинира функция
`bool leftCopy ([подходящ тип]t1, [подходящ тип]t2)`
която проверява дали t2 е ляво копие на t1.

Пример за ляво копие:



Задача 8. Нека е дадена следната структура за възел в линеен едносвързан списък:

struct Elem { int x; Elem *next;};

Да се реализира функция:

void zip ([подходящ тип] L1, [подходящ тип] L2, [подходящ тип] p, [подходящ тип] R),

където L1 и L2 са указатели към първите елементи на два линейни едносвързани списъка <L1,1, L1,2,..., L1,n> и <L2,1, L2,2,..., L2,n> с числа, които имат еднакъв брой елементи, а p е двуместен предикат над числа (p: int × int ⟶ bool). Функцията zip да създава и връща чрез R списък от сумите на съответни елементи от L1 и L2, за които p(L1,k,L2,k) е истина.

Пример: за списъците L1 = [1, 2, 3, 4] и L2 =[5, 2, 1, 3] и предиката p(x,y) = “y дели x”, ще се получи списъкът R = [4, 4] (съответните двойки са (2, 2) и (3,1)).

Задача 9: Нека е дадена следната структура, описваща възел в двоично дърво:

struct Node { int x; Node *left, *right};

Да се дефинира клас LevelIterator, с чиято помощ може да се получи броят на различните елементи (пренебрегвайки повторенията) във всяко последователно ниво на дърво.

Класът да предоставя следните методи:

1. Конструктор, който създава итератор чрез указател към корена на дадено дърво
2. Метод int currentSetSize() const, който връща големината на множеството от елементите на текущото ниво (т.е. броят на различните елементи в нивото, пренебрегвайки повторенията)
3. Метод void next(), чрез който итераторът преминава към следващото ниво
4. Метод bool end() const, който проверява дали итераторът е преминал след последното ниво в дървото.

Пример: За дърветата на фигура 1, последователните стойности, получени от итератора, ще бъдат съответно 1, 2, 2 за първото дърво и 1, 1, 1 за второто.

Задача 10: Двоичното дърво от числа t_2 наричаме "пребroител" на двоичното дърво от числа t_1, тогава и само тогава, когато:

- И двете дървета са празни, или
- Ако t_1 е непразно дърво с ляво поддърво L_1 и дясно поддърво R_1, то t_2 е дърво със стойност на корена x, равна на броя на децата (непосредствените наследници, които не са празни дървета) на корена на t_1, ляво поддърво L_2, което е пребroител на L_1 и дясно поддърво R_2, което е пребroител на R_1.

Нека е дадена следната структура, описваща възел в двоично дърво:

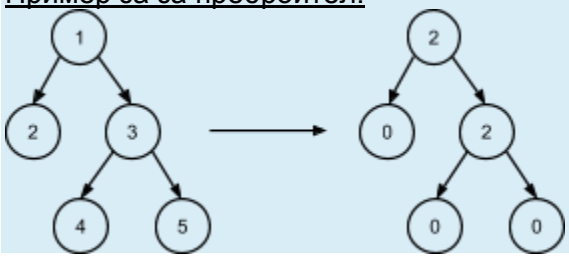
```
struct Node { int x; Node *left, *right};
```

Да се дефинира функция

```
bool counter ([подходящ тип]t1, [подходящ тип]t2)
```

която проверява дали дървото t2 е пребroител на дървото t1.

Пример за за пребroител:



Задача 11: Нека е дадена следната структура за възел в линеен едносвързан списък:

```
struct Elem { int x; Elem *next;};
```

Да се реализира функция:

```
void project ([подходящ тип] L, [подходящ тип] p, [подходящ тип] L1,
[подходящ тип] L2),
```

където L е указател към първия елемент на линеен едносвързан списък с четен брой (2n) числа, а p е едноместен предикат над числа (p: int ⟶ bool). Функцията `project` да създава и връща чрез L1 списък с всички елементи на L на четни позиции (започвайки от 0), за които p е истина, а чрез L2 списък с всички елементи на L на нечетни позиции (започвайки от 0), за които p е истина.

Пример: от списъка L = [1, 7, 4, 3, 8, 8] и предиката p(x) = “x > 3”, ще се построят списъците L1 = [4, 8] и L2 = [7, 8].

Задача 12: Нека е дадена следната структура, описваща възел в двоично дърво:
`struct Node { int x; Node *left, *right};`
Да се дефинира клас `LevelIterator`, с чиято помощ могат да се получат последователно сумата на елементите на поредните нива в дадено дърво. Класът да предоставя следните методи:

1. Конструктор, който създава итератор чрез указател към корена на дадено дърво
2. Метод `int currentSum() const`, който връща сумата от елементите на текущото ниво
3. Метод `void next()`, чрез който итераторът преминава към следващото ниво
4. Метод `bool end() const`, който проверява дали итераторът е преминал след последното ниво в дървото.

Пример: За дърветата на фигура 1, последователните стойности, получени от итератора, ще бъдат: 1, 5, 9 и 2, 2, 0.