

# kurs języka C++

## drzewa poszukiwań binarnych

Instytut Informatyki  
Uniwersytetu Wrocławskiego

Paweł Rzechonek

### Prolog

BST (ang. Binary Search Tree), czyli drzewo poszukiwań binarnych, to dynamiczna struktura danych przechowująca zbiór wartości pochodzących z jakiegoś uniwersum z porządkiem liniowym. W drzewie BST lewe poddrzewo każdego węzła zawiera wyłącznie elementy o kluczach mniejszych niż klucz w węźle a prawe poddrzewo zawiera wyłącznie elementy o kluczach większych niż klucz w węźle. Każdy węzeł, oprócz klucza, przechowuje jeszcze wskaźniki na swojego lewego i prawego syna (oraz w niektórych implementacjach również na swojego ojca).



### Zadanie

Zdefiniuj szablon klasy dla drzewa BST. Klasa szablonowa `bst<T>` ma reprezentować drzewo binarnych poszukiwań zbudowane na węzłach typu `node<T>`. W drzewie BST można przechowywać obiekty tylko takiego typu `T`, który zagwarantuje możliwość porównywania elementów w sensowny sposób (relacja porównywania  $\leq$  obiektów typu `T` musi być zwrotna, przechodnia i antysymetryczna).

Szablon klasy `bst<T>` reprezentującej drzewo BST ma być napisana zgodnie ze sztuką programowania dynamicznych struktur danych: w pełni funkcjonalny węzeł drzewa `node<T>` zdefiniuj jako prywatną klasę zagnieżdżoną w opakowaniu `bst<T>`, posiadającym wygodny dla programisty interfejs z operacjami słownikowymi (wstawianie, usuwanie i wyszukiwanie elementów) realizowanymi w drzewie BST. Obiekty drzew BST mają być kopiowalne (konstruktor i przypisanie kopiujące i przenoszące). Uzupełnij definicję klasy dla drzewa BST o inicjalizację za pomocą listy wartości `initializer_list<T>`. Nie zapomnij o operatorze strumieniowym do wypisania zawartości drzewa metodą *in-order*.

Co do szablonu to powinien on posiadać dwa parametry: typ danych przechowywanych w drzewie oraz trejta implementującego operację porównywania elementów wybranego typu. Trejt ma być parametrem domyślnym w szablonie ustawionym na obiekt zawierający operację tradycyjnego porównywania za pomocą operatora `<`; definiuj także trejta implementującego

porównywanie za pomocą operatora `>`. Zadbaj również o specjalizację dla wskaźników a w szczególności dla wskaźnika typu `const char*`.

Na koniec napisz interaktywny program testujący działanie drzewa BST (interpretuj i wykonuj polecenia wydawane z klawiatury). Obiekt drzewa, który będziesz testować utwórz na sterce operatorem `new` i nie zapomnij zlikwidować go operatorem `delete` przed zakończeniem działania programu!

### **Ważne elementy programu**

- Podział programu na pliki nagłówkowe i źródłowe.
- Definicja szablonu klasy `bst<>` dla drzewa BST.
- Zagnieżdżona definicja węzła `node<>`.
- Konstruktor bezargumentowy i konstruktor z listą wartości w klasie `bst<>`.
- Rekurencyjny konstruktor kopiujący w klasie `node<>`.
- Destrukcja całego drzewa BST.
- Implementacja kopiowania i przenoszenia.
- Definicja trejta realizującego porównania.
- Realizacja specjalizacji ogólnie dla wskaźników i w szczególności dla `const char*`.
- W funkcji `main()` należy przetestować całą słownikową funkcjonalność drzewa BST.