### Struktury danych: stos, kolejka, lista, drzewo

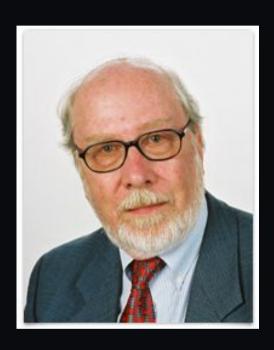
Wykład: dane w strukturze, funkcje i rodzaje struktur, LIFO, last in first out, kolejka FIFO, first in first out, push, pop, size, empty, głowa, ogon, implementacja tablicowa, wskaźnikowa, lista, węzeł, jednokierunkowa, dwukierunkowa, z wartownikiem, cykliczna, sort, reverse, insert, remove, drzewo binarne, parent, child node, dziel i zwyciężaj, add, find, print

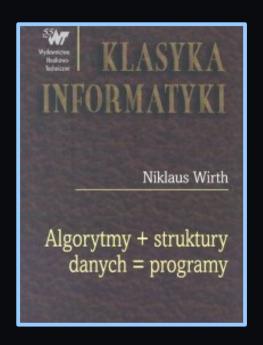


Każdy algorytm, każdy program operuje na różnorodnych danych:

- mają one przeważnie określoną formę zapewniającą mu pożądane właściwości
- do ich przechowywania i dalszej obróbki potrzebne jest:
  - □ ich zapamiętanie
  - □ stworzenie dodatkowych algorytmów zapewniających:
    - dostęp do wszystkich elementów
    - możliwość modyfikacji zawartości zbioru

#### programy = algorytmy + struktury danych





profesor Niklaus Wirth (twórca języka Pascal)



Struktury danych są zaawansowanymi pojemnikami na dane, które gromadzą je i układają w odpowiedni sposób

- ich różnorodność jest ogromna, a dla każdej znaleziono wiele zastosowań oraz interesujących algorytmów
- powszechnie spotykane jest używanie jednych struktur danych do przetwarzania informacji zgromadzonych w innych
- są one fundamentalnym narzędziem programisty i ich znajomość jest niezbędna w profesjonalnym programowaniu



# STOS (LIFO)



000000

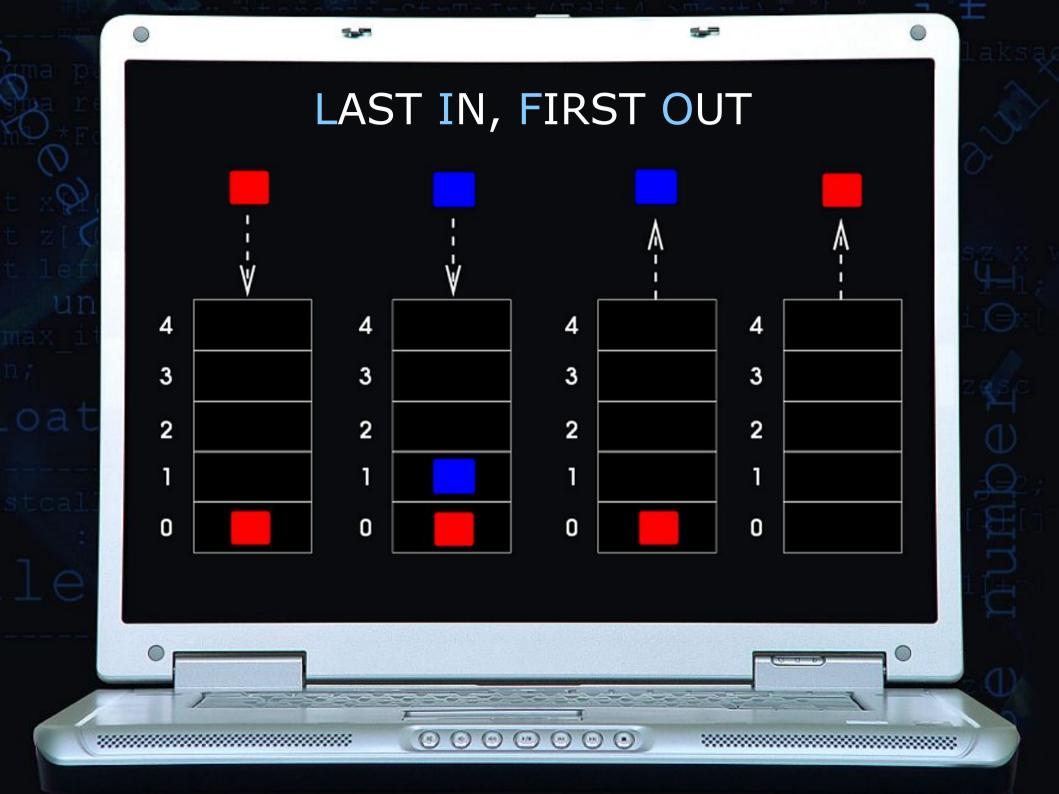
#### STOS JAKO STRUKTURA DANYCH

Stos jest strukturą liniowo uporządkowanych danych, z których jedynie ostatni element, zwany wierzchołkiem, jest w danym momencie dostępny.

W wierzchołku odbywa się dołączanie nowych elementów, również jedynie wierzchołek można usunąć.

Ze stosu usuwany jest element, który został dodany najpóźniej - architektura LIFO (Last In, First Out)

Stos jest bardzo często wykorzystywaną strukturą danych. Działanie na nim jest często porównywane do stosu zmywanych talerzy: nie można usunąć talerza znajdującego się na dnie stosu nie usuwając wcześniej wszystkich innych. Nie można także dodać nowego talerza gdzieś indziej, niż na samą górę.



#### FUNKCJE OBSŁUGUJĄCE STOS:

empty() zwraca **true** jeżeli stos jest pusty

pop() usuwa element znajdujący się na szczycie stosu

push() dodaje element na szczycie stosu

size() zwraca liczbę elementów stosu

top() zwraca referencję do elementu na szczycie stosu

## KOLEJKA (FIFO)



#### KOLEJKA - STRUKTURA DANYCH

Kolejka to struktura danych, w której element, który został wstawiony do kolejki jako pierwszy, jako pierwszy jest z niej pobierany. Tę strukturę można porównać do kolejki w sklepie - architektura FIFO (First In, First Out).

głowa kolejki – pierwszy element w kolejce

ogon kolejki – pierwsze wolne miejsce w kolejce

Podobnie jak stos, kolejka to struktura danych o dostępie ograniczonym. Zakłada ona dwie podstawowe operacje:

- wstaw (push) wprowadź dane na ogon kolejki
- ♦ obsłuż (pop) usuń dane z czoła kolejki

# FIRST IN, FIRST OUT 0000000

# IMPLEMENTACJA TABLICOWA



empty() zwraca **true** jeżeli kolejka jest pusta

pop() usuwa element znajdujący się na początku kolejki

push() dodaje element na końcu kolejki

size() zwraca liczbę elementów w kolejce

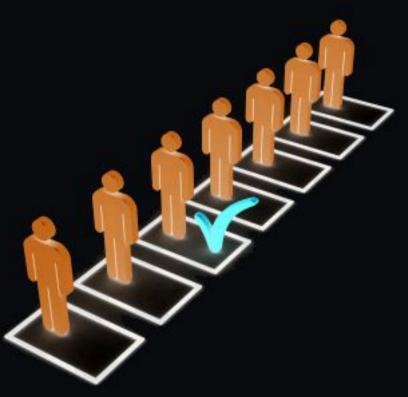
#### KOLEJKA PRIORYTETOWA

W tego typu kolejce każda ze znajdujących się w niej danych dodatkowo ma przypisany priorytet, który modyfikuje kolejność późniejszego wykonania.

Oznacza to, że pierwsze na wyjściu niekoniecznie pojawią się te dane, które w kolejce oczekują najdłużej, lecz te o największym priorytecie.

Taka kolejka służy np. do zarządzania procesami uruchomionymi w systemie operacyjnym. W jej implementacji wykorzystujemy strukturę kopca (*heap*) lub listy jednokierunkowej posortowanej.

# LISTA



000000

#### LISTA JAKO STRUKTURA DANYCH

Lista to <u>struktura danych</u> służąca do przechowywania nieznanej z góry ilości informacji <u>tego samego typu</u>.

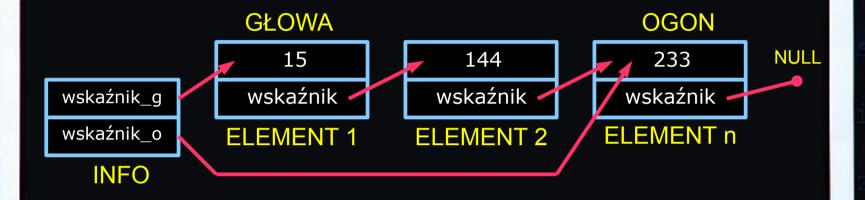
Składa się ona z węzłów, które zawierają dane przechowywane w liście oraz wskaźnik do kolejnego (a w przypadku listy dwukierunkowej także do poprzedniego) elementu.

Listę rozpoczyna wskaźnik do początku (głowa), a ostatni element wskazuje na NULL.

#### RODZAJE LIST

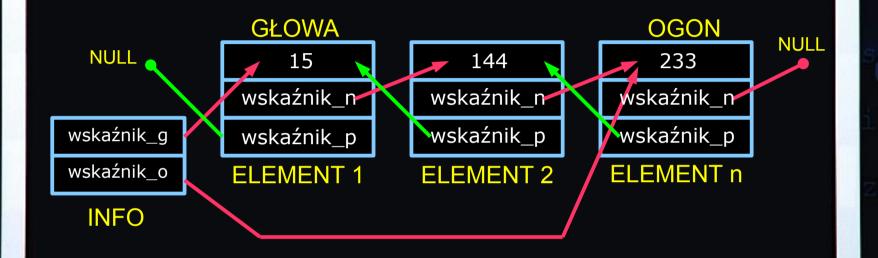
- ♦ Listy jednokierunkowe występuje blok INFO (informacyjny) zawierający głowę (wskaźnik do pierwszego elementu) i ogon (wskaźnik do ostatniego elementu). Każdy element zawiera wskaźnik do następnego elementu. Ostatni element wskazuje na NULL (patrz rys. 1)
- ♦ Listy dwukierunkowe różni się od jednokierunkowej tym, że każdy element zawiera wskaźnik zarówno do następnego jak i do poprzedniego elementu
- ♦ Listy posortowane / nieposortowane dane na liście mogą być uporządkowane lub też rozmieszczone zupełnie przypadkowo
- ♦ Listy z wartownikiem / bez wartownika na początku listy może znajdować się dodatkowy element zwany wartownikiem (wówczas pierwszy element listy to następnik wartownika; ostatni element listy to poprzednik wartownika; lista pusta składa się tylko z wartownika)

#### LISTA JEDNOKIERUNKOWA



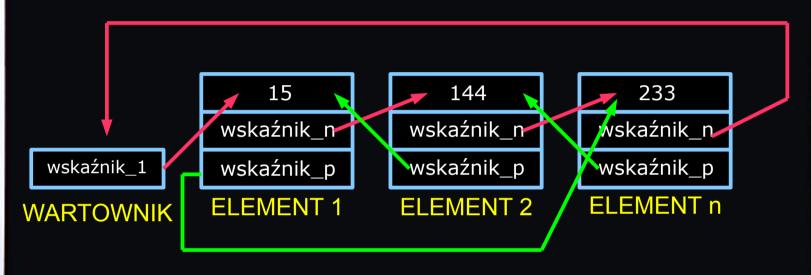
Rys. 1 Przykład listy jednokierunkowej

#### LISTA DWUKIERUNKOWA



Rys. 2 Przykład listy dwukierunkowej

#### LISTA DWUKIERUNKOWA Z WARTOWNIKIEM (CYKLICZNA)



Rys. 3 Przykład listy dwukierunkowej z wartownikiem – pierwszy element listy to następnik wartownika; ostatni element listy to poprzednik wartownika; lista pusta składa się tylko z wartownika. Inaczej nazywana cykliczną, bo następnikiem ostatniego elementu jest efektywnie pierwszy element, zaś poprzednikiem pierwszego ostatni. Po liście tej można więc przemieszczać się w sposób cykliczny. Lista jednokierunkowa również może być cykliczna.

#### FUNKCJE OBSŁUGUJĄCE LISTĘ:

push\_front() dodaje element na początku listy

push\_back() dodaje element na końcu listy

insert() dodaje element we wskazanym miejscu listy

pop\_front() usuwa element z początku listy

pop\_back() usuwa element z końca listy

size() zwraca liczbę elementów na liście

max\_size() zwraca maks. liczbę elementów jakie może zmieścić lista

empty() sprawdza czy lista jest pusta

remove() usuwa z listy wszystkie elementy mające daną wartość

sort() układa elementy na liście rosnąco

reverse() odwraca kolejność elementów na liście

#### DRZEWO BINARNE



#### DRZEWO JAKO STRUKTURA DANYCH

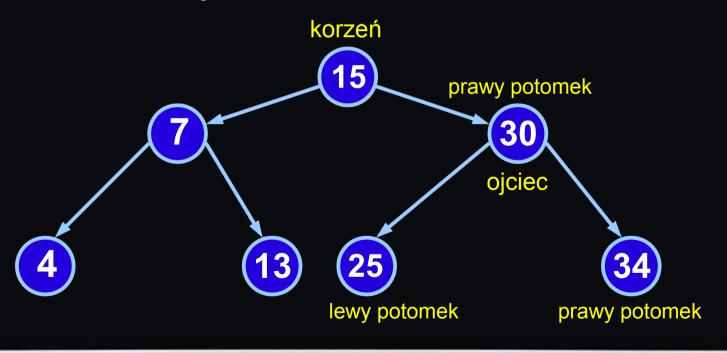
Drzewo binarne jest hierarchiczną strukturą danych, którego elementy nazywa się węzłami (ang. nodes) lub wierzchołkami.

W drzewie binarnym każdy węzeł posiada co najwyżej dwa następniki (stąd jego nazwa, bo binarny = dwójkowy, zawierający dwa elementy). Następniki te nazywamy potomkami, dziećmi lub węzłami potomnymi (child node) danego węzła - ojca (parent node).

Wszystkie elementy znajdujące się w lewym poddrzewie są mniejsze od swojego ojca, natomiast elementy prawego poddrzewa są większe od ojca. Reguła ta obowiązuje wszystkie poddrzewa.

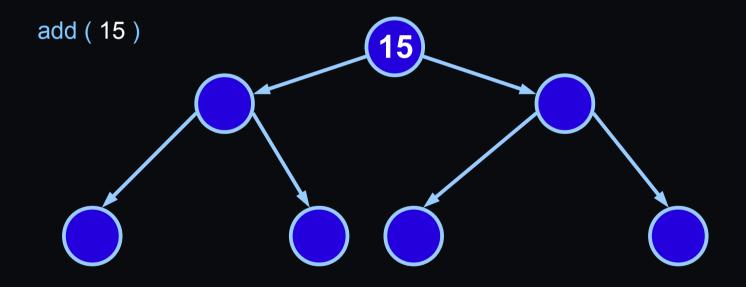
#### SCHEMAT DRZEWA BINARNEGO

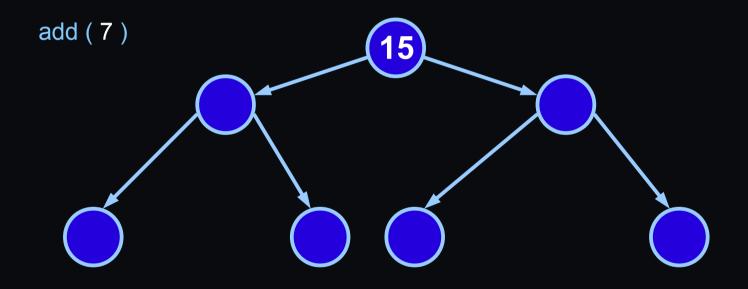
Węzeł nie posiadający rodzica nazywamy korzeniem drzewa binarnego (root node). W naszym przykładzie korzeniem jest węzeł o wartości 15. Każde drzewo binarne posiada dokładnie jeden korzeń.

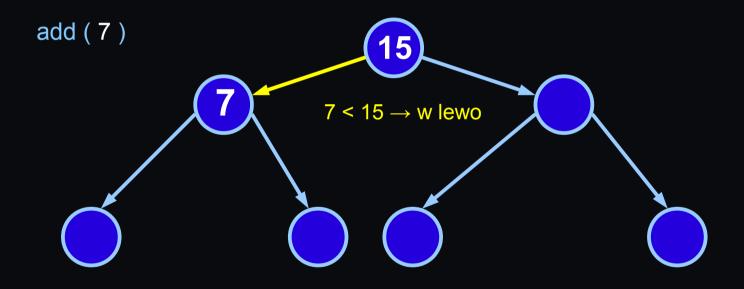


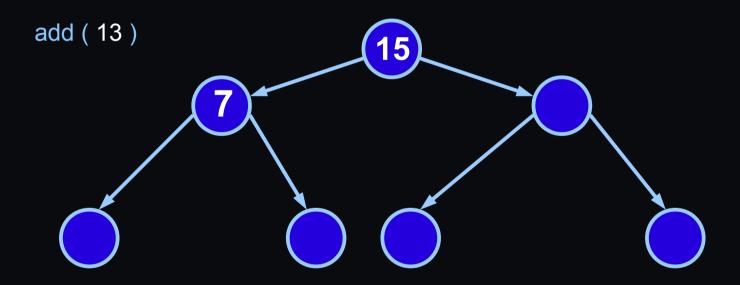
Aby wstawić nowy element do drzewa należy począwszy od korzenia porównywać daną liczbę z węzłem i jeżeli jest ona mniejsza od wartości przechowywanej w tym węźle to poruszać się w lewo po drzewie, w przeciwnym wypadku poruszać się w prawo. Wędrujemy tak długo, aż dojdziemy do pustego miejsca.

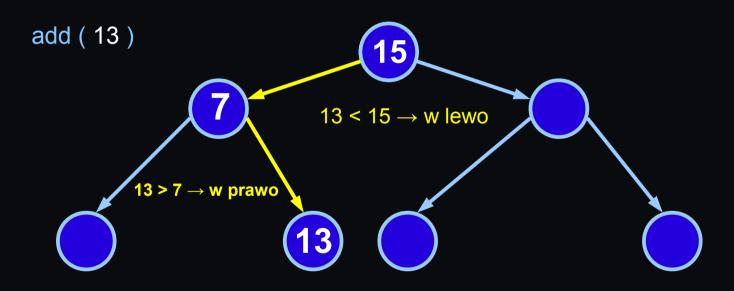
add (15)

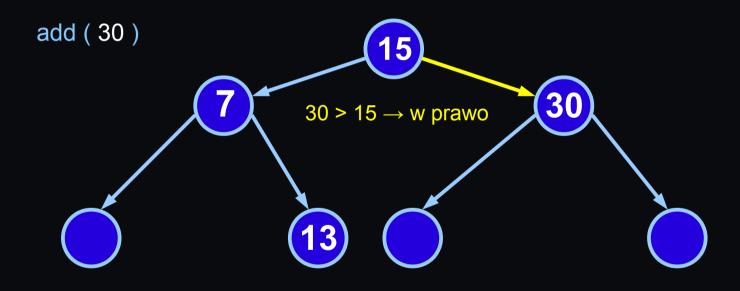


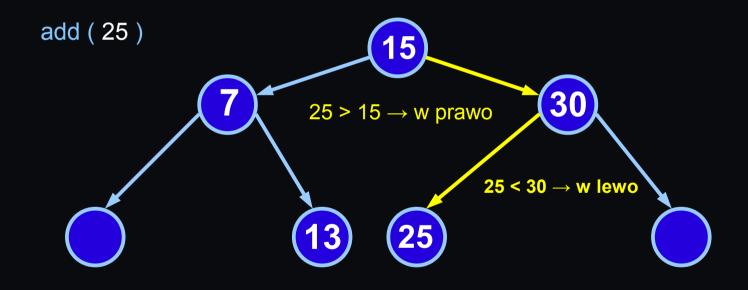


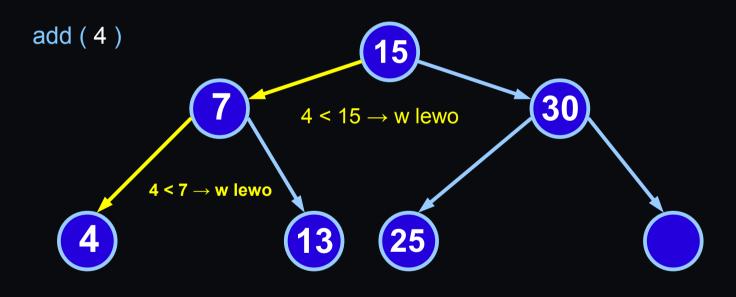


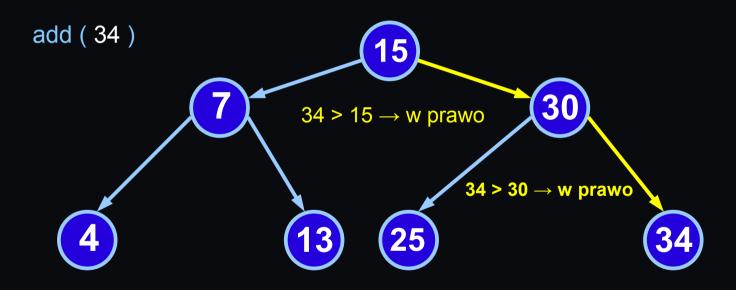




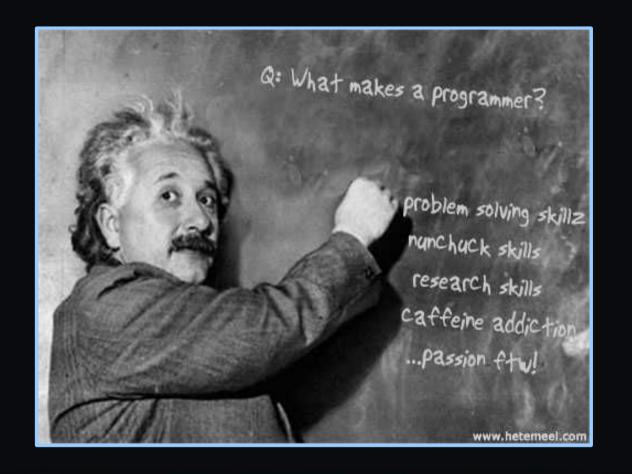






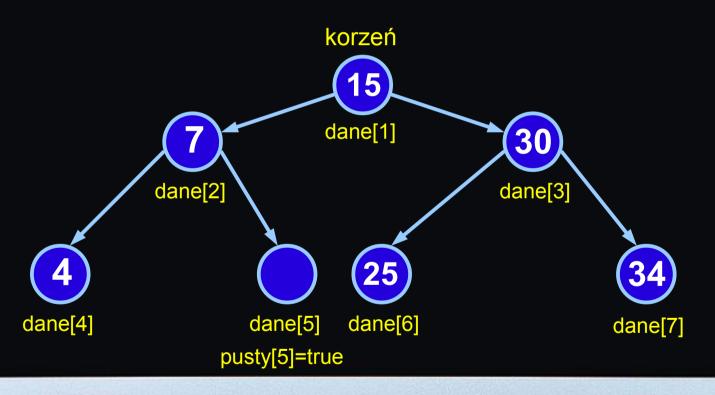


#### IMPLEMENTACJA DRZEWA



#### IMPLEMENTACJA TABLICOWA DRZEWA

Indeks lewego potomka węzła k-tego wynosi 2k, zaś indeks prawego potomka węzła k-tego wynosi 2k+1. Jeżeli k-ty węzeł nie posiada liczby, to pusty[k]=true



#### IMPLEMENTACJA WSKAŹNIKOWA DRZEWA

Do lewego potomka węzła k-tego dostajemy się lewym wskaźnikiem ojca, zaś do prawego potomka węzła k-tego dostajemy się prawym wskaźnikiem ojca. Jeśli węzeł potomny nie istnieje, to odpowiedni wskaźnik ojca wskazuje na NULL.





#### KOPIEC JAKO STRUKTURA DANYCH

Kopiec to drzewo binarne, w którym wszystkie węzły spełniają tzw. warunek kopca:

Węzeł nadrzędny jest większy lub równy węzłom potomnym (w porządku malejącym relacja jest odwrotna - mniejszy lub równy)

Konstrukcja kopca jest nieco bardziej skomplikowana od konstrukcji drzewa binarnego, ponieważ po wstawieniu liczby musimy dodatkowo zatroszczyć się o zachowanie warunku kopca.

# WYGLĄD PRZYKŁADOWEGO KOPCA korzeń = ojciec o największej wartości 10 ojciec większy lub równy potomkom ojciec większy lub równy potomkom



add() dodaje element do drzewa

find() znajduje element w drzewie i zwraca go

remove() usuwa dane element

print() wydruk zawartości drzewa



Dzięki tym programom lepiej zrozumiesz działanie poszczególnych struktur danych:

http://bit.ly/zrozum-struktury

Zawartość archiwum ZIP:

stos.exe kolejka.exe lista.exe drzewo-binarne.exe