Young Programmer

www.inceptio.org.pl

Organizator:

fundacja INCEPTIO

Partnerzy:

Wydział Fizyki Politechnika Warszawska

Stowarzyszenie KLATRAT

Dofinansowanie:

Miasto Stołeczne Warszawa

Zakochaj się w Warszawie

Young Programmer

www.inceptio.org.pl

Zajęcia nr 6

Aglorytmy i struktury danych
Lista dwukierunkowa

dr inż. Łukasz Graczykowski
mgr inż. Leszek Kosarzewski
Wydział Fizyki Politechniki Warszawskiej

Organizator:

Partnerzy:

Dofinansowanie:

MIASTO STOŁECZNE WARSZAWA

Plan

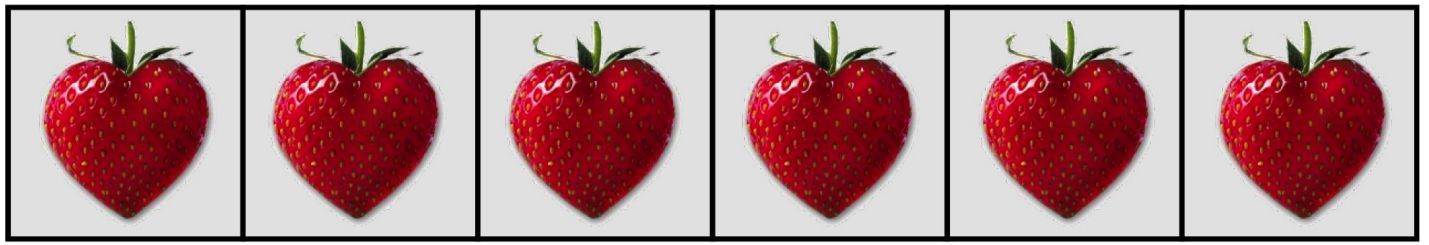
- Wykorzystanie zdobytej wiedzy do napisania listy dwukierunkowej – zaawansowanej struktury danych typu tablicowego o znacznie większych możliwościach
- Spróbujemy to zadanie napisać wspólnie
- Obrazki pożyczone z wykładu dr. inż T. Pietrzaka “Podstawy programowania” dla studentów I roku Wydziału Fizyki PW

Dobra rada

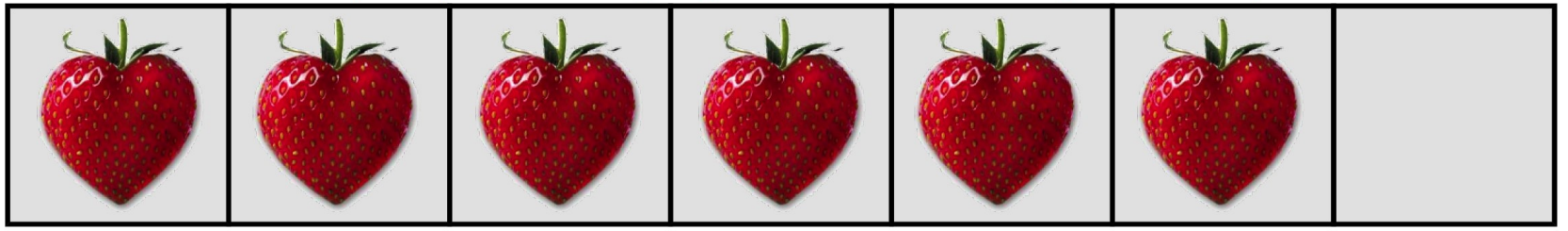
ZAWSZE zaczynajcie pisanie swojego programu od NAJMNIEJSZEJ możliwej kompilującej się wersji.

Problem 1: dodanie elementu do tablicy

Cel: Wstawić kaczora w środek (tablicy) truskawek

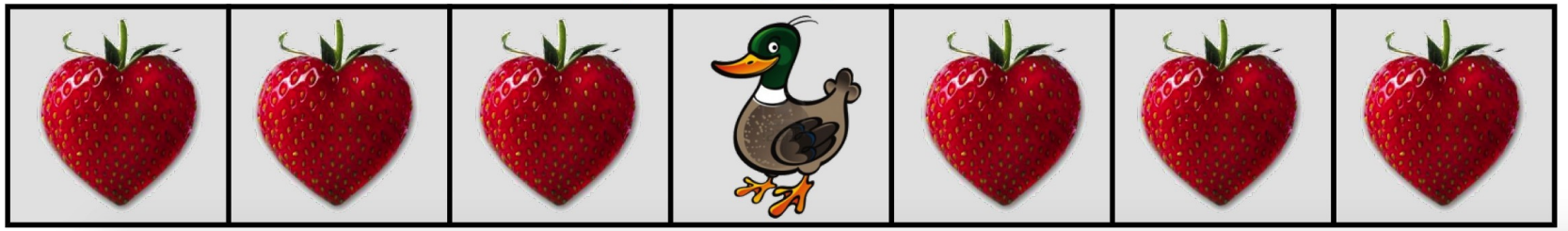


Krok #1: Powiększyć tablicę o jeden element



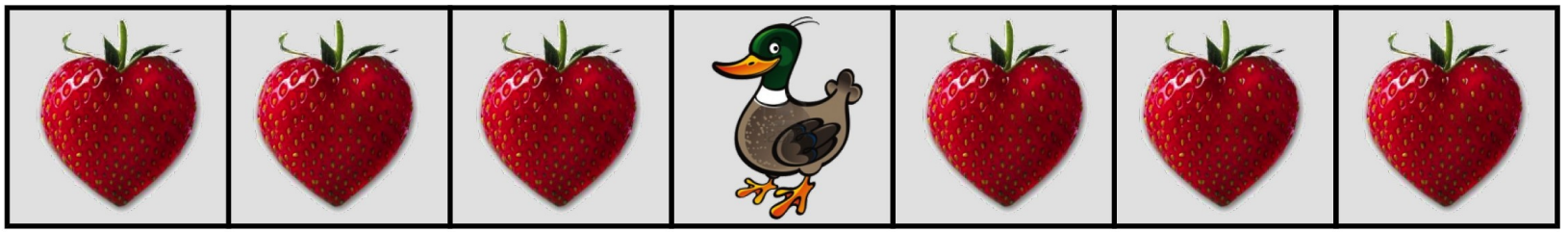
Krok #2: Przesadzić 3 truskawki o jedno pole dalej

Krok #3: Wstawić kaczora pomiędzy truskawki

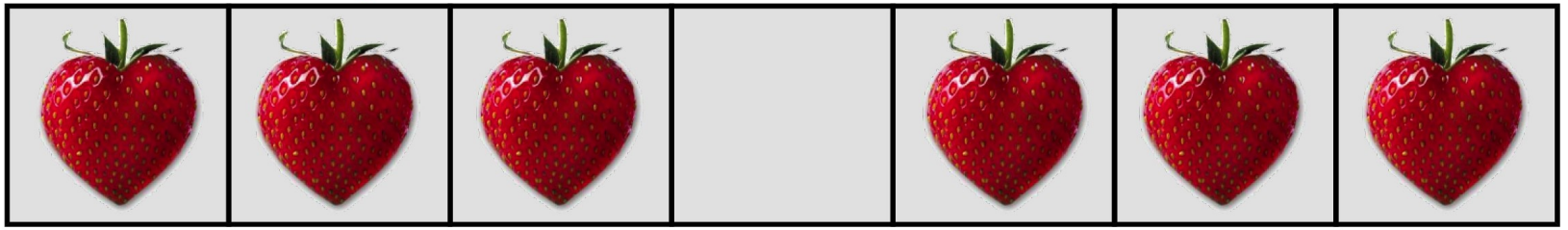


Problem 2: usunięcie elementu z tablicy

Cel: Usunąć kaczora spośród (tablicy) truskawek

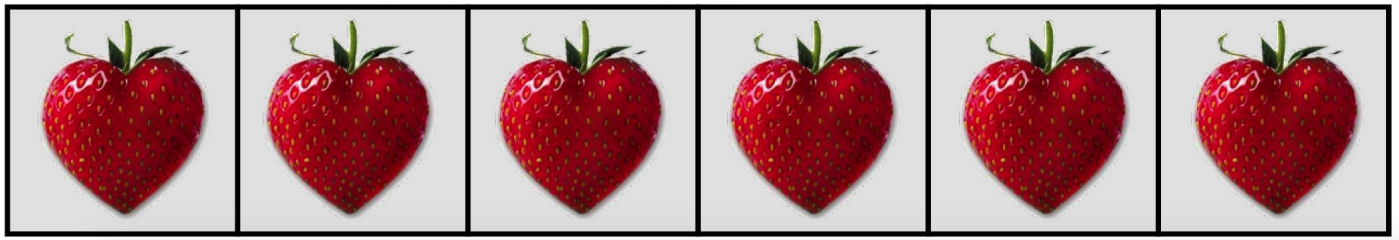


Krok #1: Usunąć kaczora

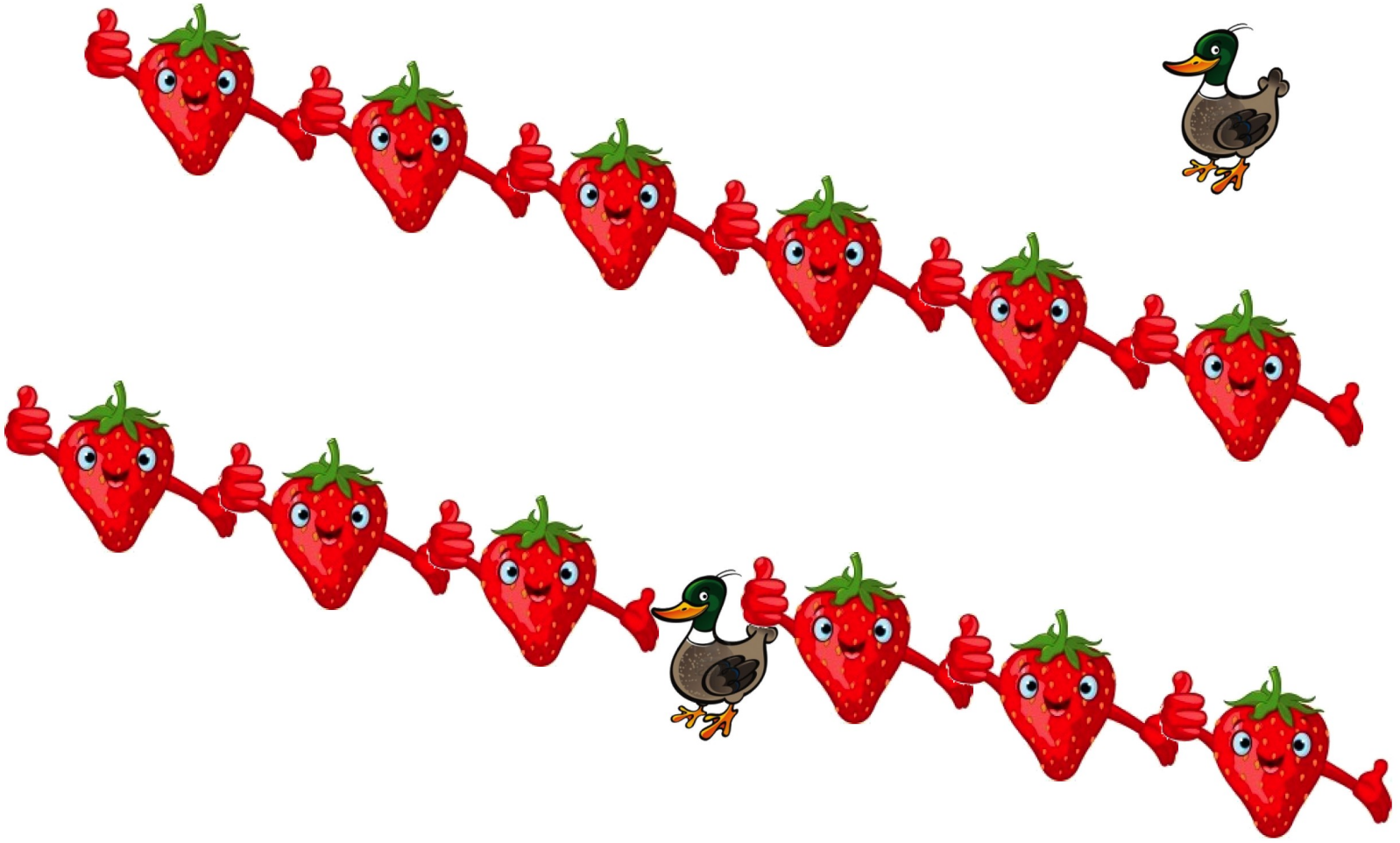


Krok #2: Przesadzić 3 truskawki o jedno pole w lewo

Krok #3: Zmniejszyć tablicę o jeden element

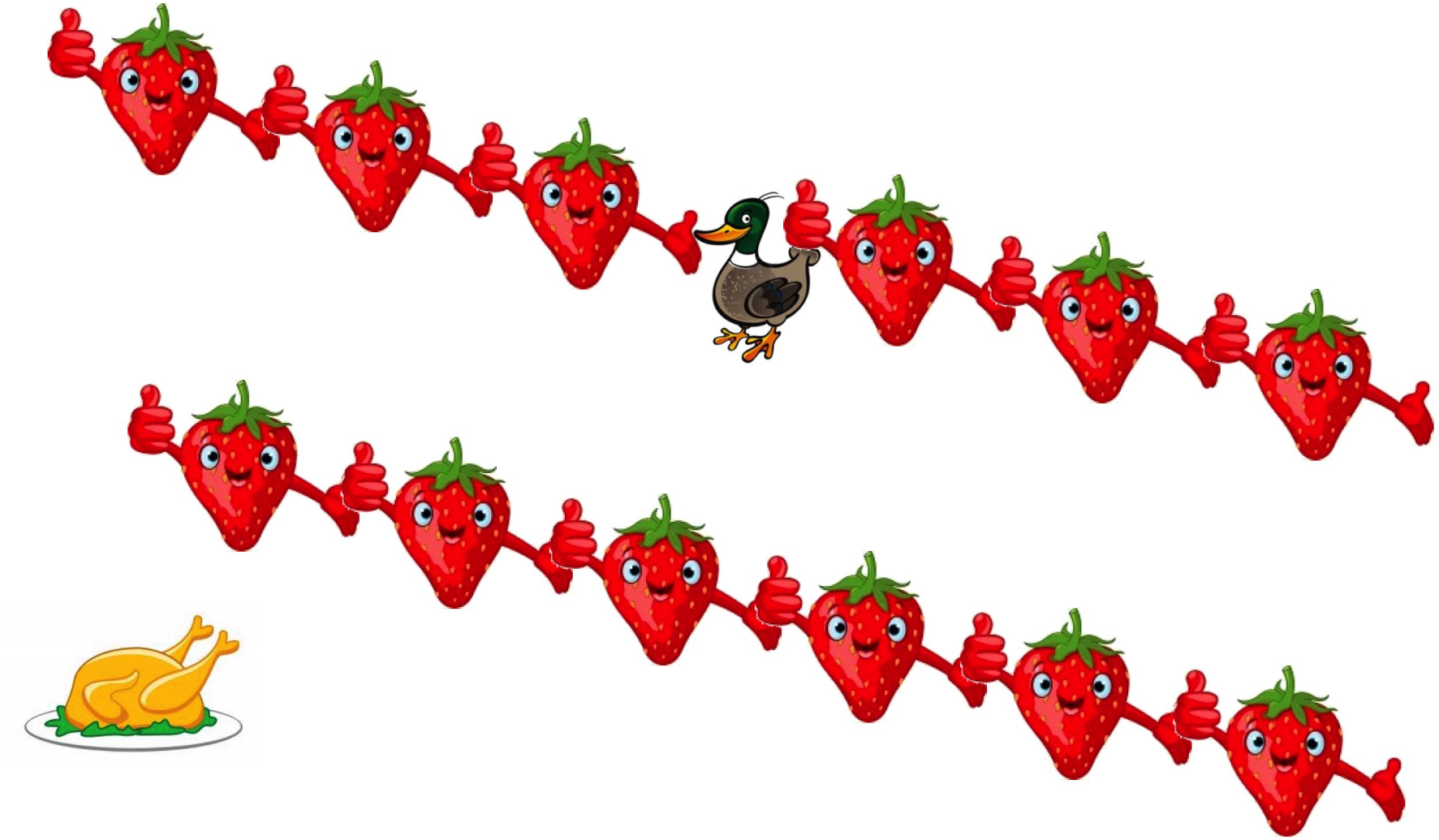


Rozwiązanie: lista!



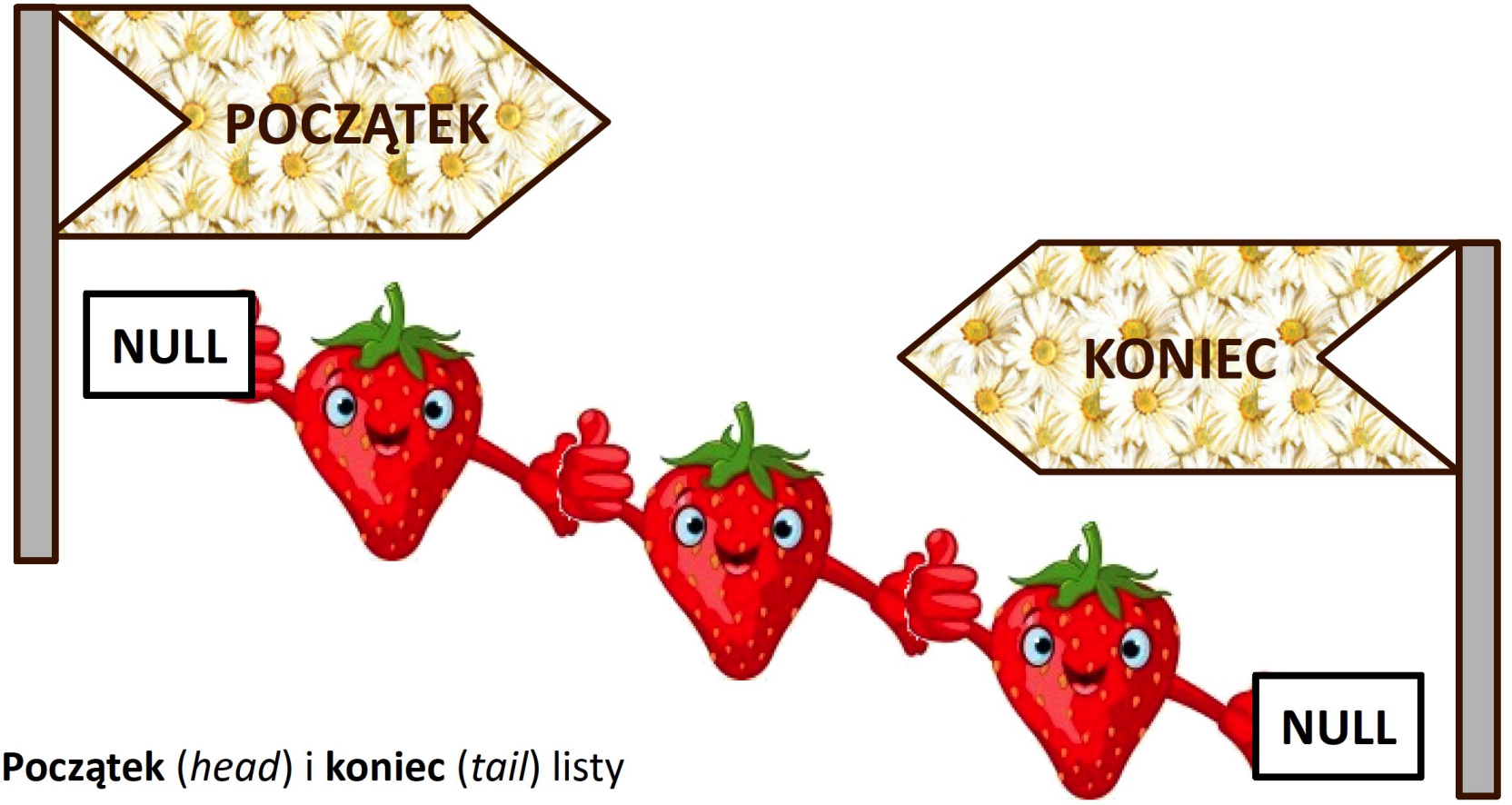
Łatwe i szybkie wstawianie w środek listy!

Rozwiązanie: lista!



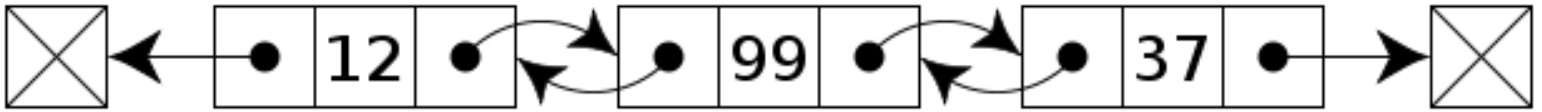
Łatwe i szybkie usuwanie z dowolnego miejsca listy!

Cechy listy dwukierunkowej



- **Początek** (*head*) i **koniec** (*tail*) listy
- Każdy element listy posiada wskazanie na **poprzedni** (*prev*) i **następny** (*next*) element listy; końcowe elementy wskazują na NULL

Tworzymy listę



A doubly linked list whose nodes contain three fields: an integer value, the link to the next node, and the link to the previous node.

Struktura opisująca każde “ogniwo” listy

```
struct Element
  {
    double value; //nasze dane – wartosc typu double
    Element *prev; //wskaznik na element poprzedni
    Element *next; //wskaznik na element nastepny
  };

class List
{
  int fCapacity; //rozmiar listy
  Element *firstElement; //wskaznik na pierwszy element w liscie
  Element *lastElement; //wskaznik na ostatni element w liscie

 public:
  List(); //konstruktor domyslny

  void AddLast(double val); //dodaje element na koncu listy
  void PrintList(); //wypisyje liste na ekran
};
```

Young Programmer

www.inceptio.org.pl

Zajęcia nr 5

Algorytmy i wskaźniki

Zadania

Organizator:

fundacja INCEPTIO

Partnerzy:

Wydział Fizyki Politechniki Warszawskiej

Stowarzyszenie KLATRAT

Dofinansowanie:

Miasto Stołeczne Warszawa

Zakochaj się w Warszawie

Zadanie 1: Tworzymy listę

W pierwszej kolejności w pliku **List.cpp** tworzymy konstruktor, metodę `void AddLast(double val)`, oraz `void PrintList()`. W pliku **main.cpp** tworzymy obiekt typu `List` używając operatora `new`, dodajemy dwa dowolne elementy i wypisujemy listę i jej rozmiar na ekran.

Jak te dwie metody działają, dopisujemy kolejne 3:

```
void AddFirst(double val); //dodaje element na poczatku listy
void RemoveLast(double val); //usuwa ostatni element z listy
void RemoveFirst(double val); //usuwa pierwszy element z listy
```

Wskazówka: należy stworzyć, używając operatora `new`, obiekt typu `Element`, przyporządkować jego polu `value` podaną wartość `val`, zaś wskaźnik na element nstępny `next` ustawiamy na `NULL`. Następnie sprawdzamy, czy lista jest pusta, czy nie (czy dostawiamy pierwszy element, czy już są jakieś elementy w liście). Jeśli lista **nie jest pusta**, wskaźnik `prev` naszego nowego obiektu typu `Element` ustawiamy na `lastElement`. Na końcu musimy ustawić (przesunąć) wskaźnik `lastElement` na nasz nowy obiekt. Jeśli lista **jest pusta**, wskaźnik `prev` ustawiamy na `NULL`, zaś `lastElement` i `firstElement` na stworzony obiekt typu `Element` (mamy tylko jeden element, więc wskaźniki na pierwszy i ostatni muszą na niego wskazywać). Analogicznie realizujemy wszystkie inne metody!

Zadanie 2: Rozbudowujemy listę

Dodajemy kolejne metody do listy:

```
void AddAt(double val); //dodanie elementu na zadane miejsce w liscie
void RemoveAt(double val); //usuniecie elementu z dowolnego miejsca w liscie
```

Dodajemy również destruktor:
~List(); //usuwa wszystkie elementy z listy (element po elemencie)

Więcej szczegółów

Patrz zadanie:
http://www.if.pw.edu.pl/~lgraczyk/JP2015/lab05/plus/zadanie5_10.11.2015.pdf