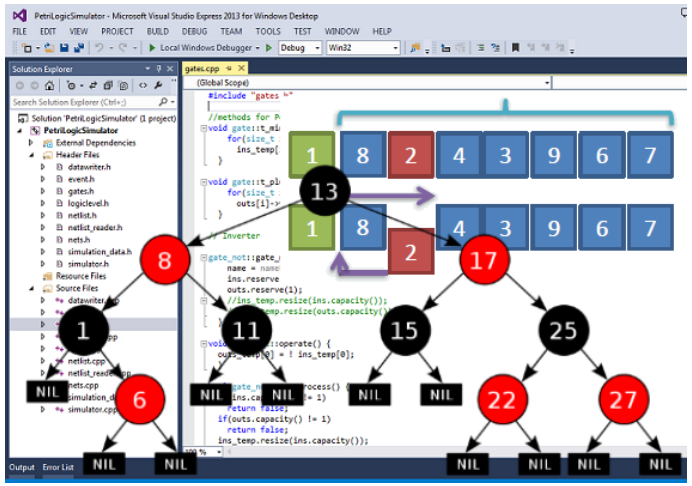
Национальный исследовательский университет «МИЭТ»

Институт интегральной электроники (группы ЭН-24-25, каф. ПКИМС)

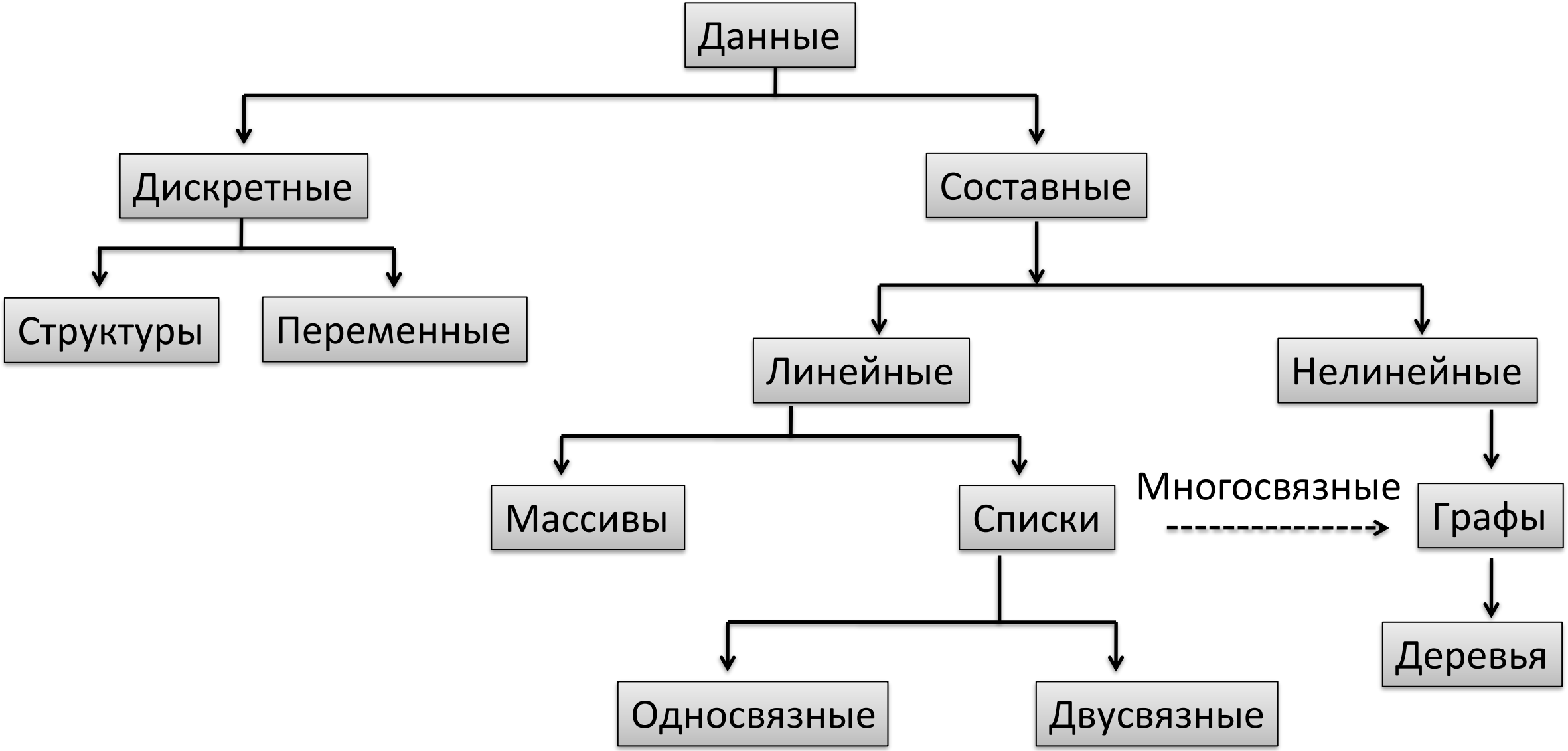
Теория алгоритмов



Лекция 2

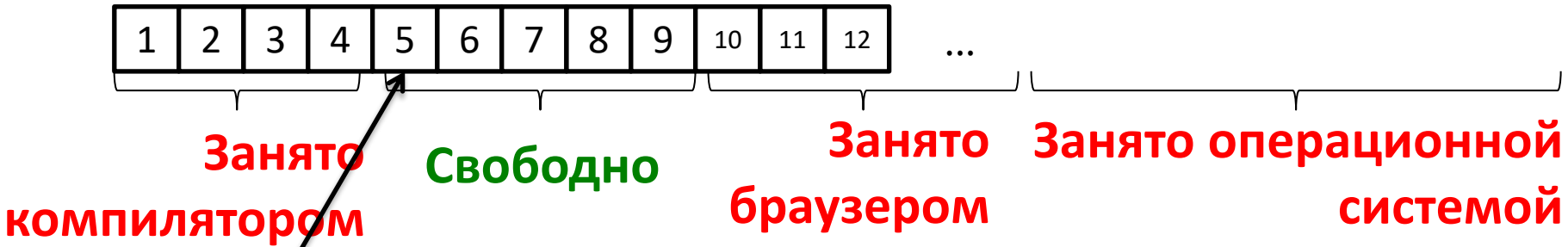
Подходы к организации данных

Общая классификация структур данных



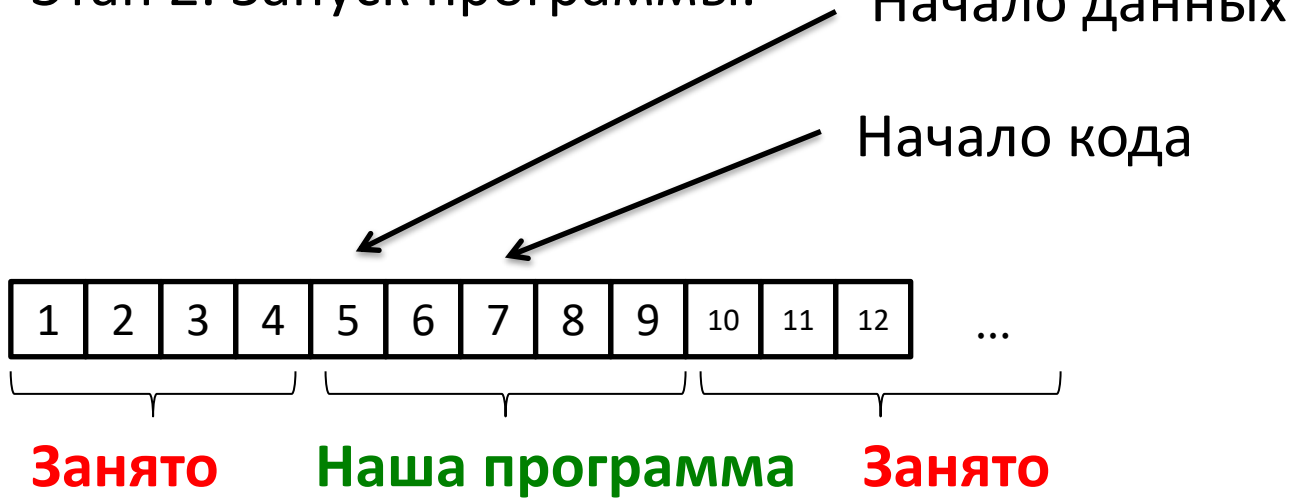
Абсолютная адресация памяти (1)

Этап 1. Компиляция программы.




```
int x = 2;

int main() {
  return 0;
}
```

Этап 2. Запуск программы.



Характеристики абсолютной адресации памяти

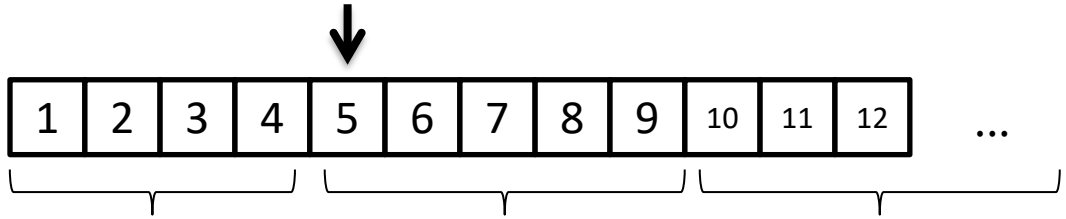
Удобства абсолютной адресации памяти:

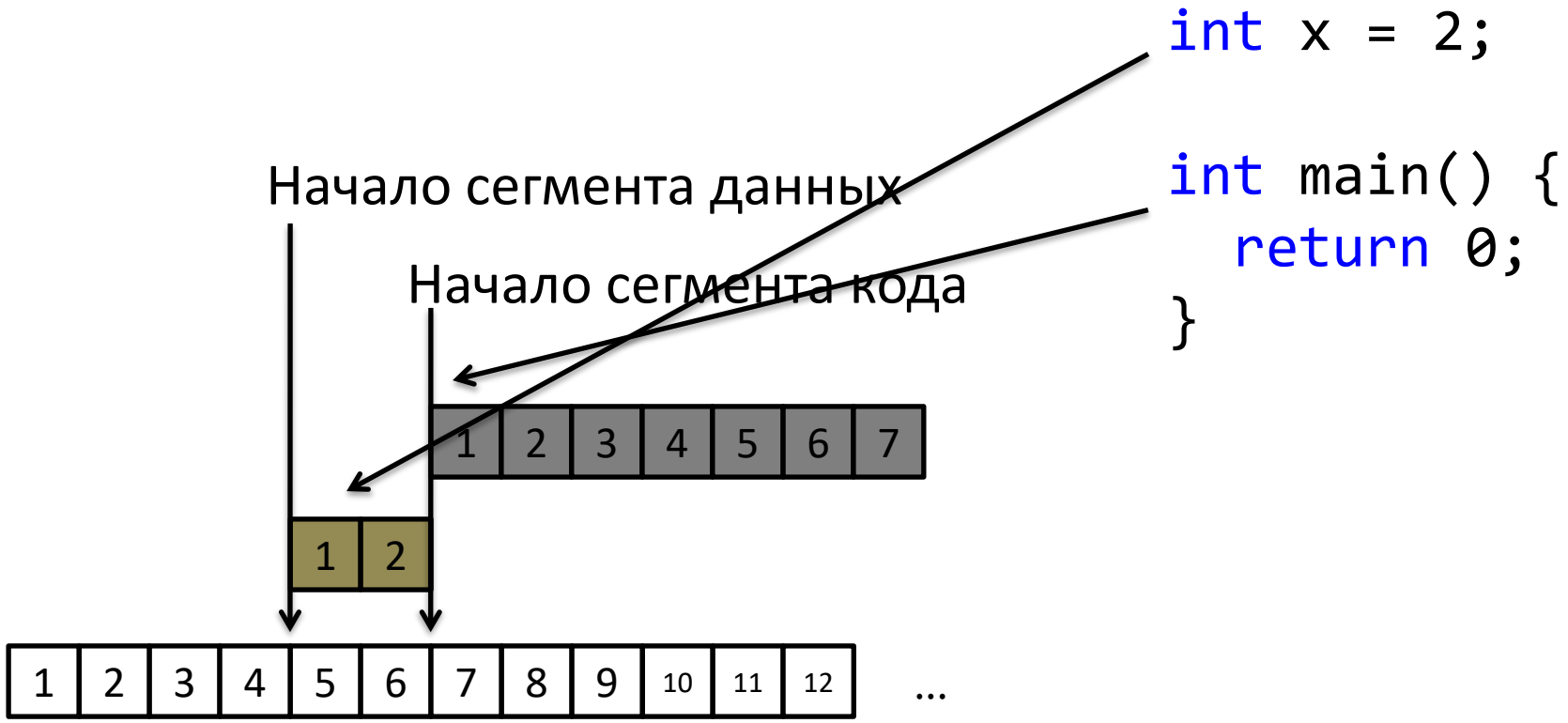
1. Простота реализации компилятора

Неудобства абсолютной адресации памяти:

1. Зависимость от объёма используемой памяти
2. Зависимость от количества запущенных программ
3. Отсутствие надёжности от запуска к запуску
4. Непереносимость исполняемого кода

Сегментная адресация памяти





Характеристики сегментной адресации данных

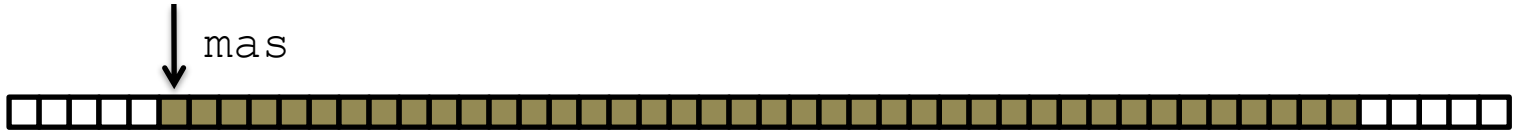
Удобства абсолютной адресации памяти:

1. **Независимость** от объёма используемой памяти
2. **Независимость** от количества запущенных программ
3. **Присутствие** надёжности от запуска к запуску
4. **Переносимость** исполняемого кода (в пределах платформы)

Неудобства абсолютной адресации памяти:

1. Сложность реализации сегментирования данных – реализуется на уровне ОС

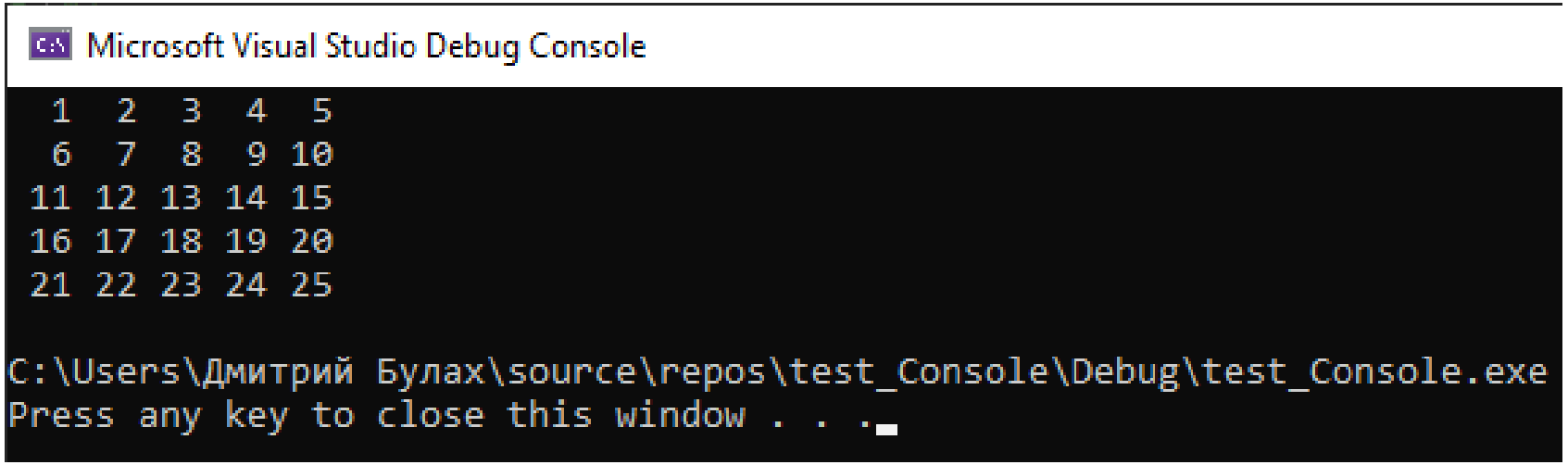
Линейные структуры данных: массивы и матрицы



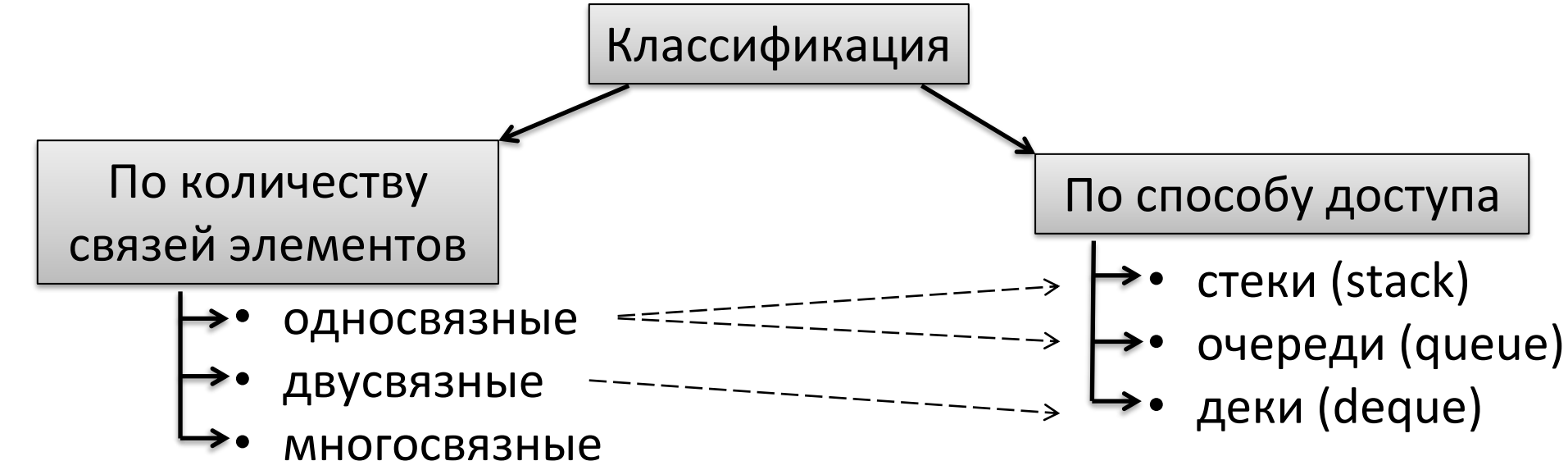

```
int mas[10];


int main() {
  int mas[10];
  for (int i = 0; i < 10; ++i)
    mas[i] = i;

  printf("%d\n", mas[6]);

  return 0;
}
```

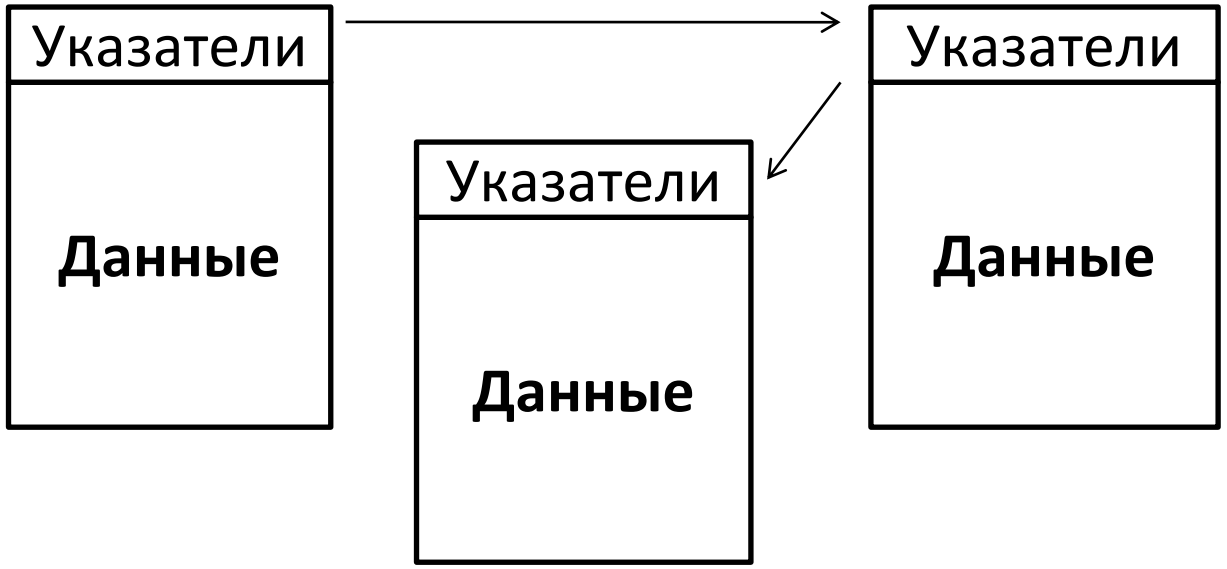
Доступ к элементам матрицы

```
#include <iostream>
#include <iomanip>

int main() {

  int mas[5][5] = {
    { 1,  2,  3,  4,  5},
    { 6,  7,  8,  9, 10},
    {11, 12, 13, 14, 15},
    {16, 17, 18, 19, 20},
    {21, 22, 23, 24, 25}
  };


  for (int i = 0; i < 5; ++i) {
    for (int j = 0; j < 5; ++j)
      std::cout << std::setw(3) << mas[i][j];
    std::cout << std::endl;
  }

  return 0;
}
```



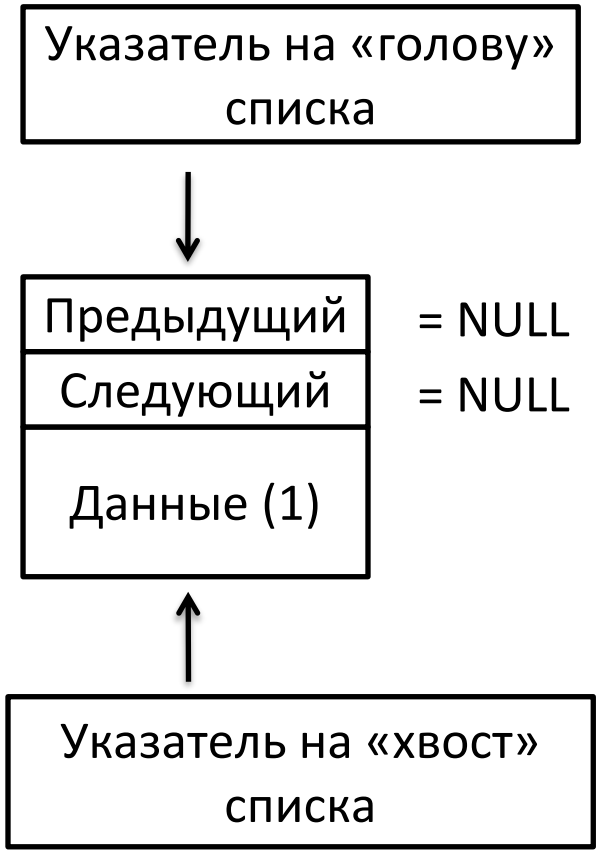
Списочные структуры данных (1)

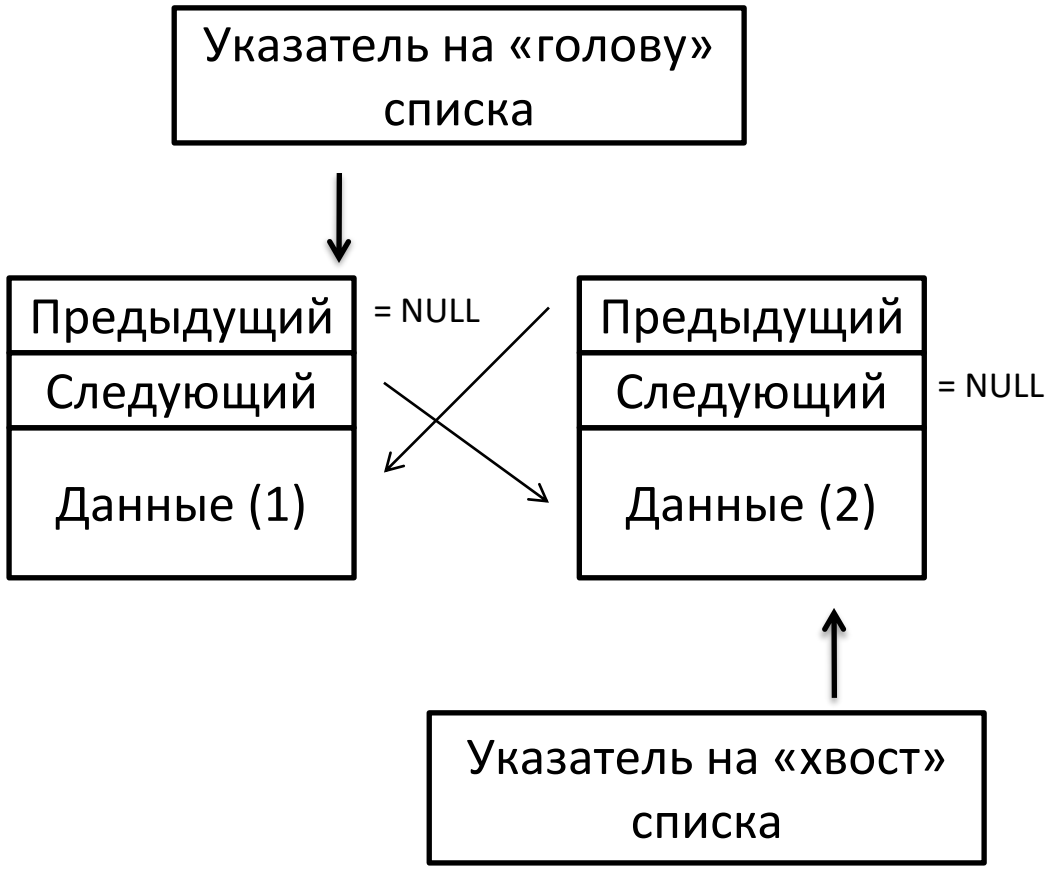





```
struct ListItem {
  ListItem *prev;
  ListItem *next;

  int      x;
  double   y;
  char     z[10];
};
```

Двусвязные списки: деки





Сравнение различных составных типов данных

	Время добавления/удаления элементов	Время доступа к элементу	Занимаемый объём памяти
Односвязный список	время выделения памяти	доступ к элементу списка	данные + указатель
Двусвязный список	время выделения памяти	доступ к элементу списка	данные + 2 указателя
Массив	выделение + копирование + удаление	минимально, «мгновенно»	только данные

Библиотека STL: контейнер std::vector

```
#include <vector>
```
← Подключаем заголовочный файл

```
using namespace std;
```
← Подключаем пространство имён

```
vector <int>  mas;
```
← Объявляем вектор целых чисел

```
mas.push_back(10);
mas.push_back(-4);
```
← Добавляем в вектор числа 10 и -4

```
cout << mas[0];
```
← Обращаемся к элементу вектора

```
mas.clear();
```
← Очищаем массив

Библиотека STL: контейнер std::list

```
#include <list>
```
← Подключаем заголовочный файл

```
using namespace std;
```
← Подключаем пространство имён

```
list <int>  lst;
```
← Объявляем список целых чисел

```
lst.push_back(10);
lst.push_back(-4);
```
← Добавляем в список числа 10 и -4

```
cout << *lst.begin();
```
← Обращаемся к элементу списка

```
lst.clear();
```
← Очищаем список

Библиотека STL: контейнеры std::queue и std::stack

```cpp
#include <iostream>
#include <queue>

using namespace std;

int main() {

  queue <int> q;

  q.push(0);
  q.push(1);

  cout << ' ' << q.front();

  return 0;
}
```

```cpp
#include <iostream>
#include <stack>

using namespace std;

int main() {

  stack <int> stack;

  stack.push(1);
  stack.push(2);

  cout << ' ' << stack.top();

  return 0;
}
```

Библиотека STL: контейнер std::map

```
#include <map>
```
← Подключаем заголовочный файл

```
using namespace std;
```
← Подключаем пространство имён

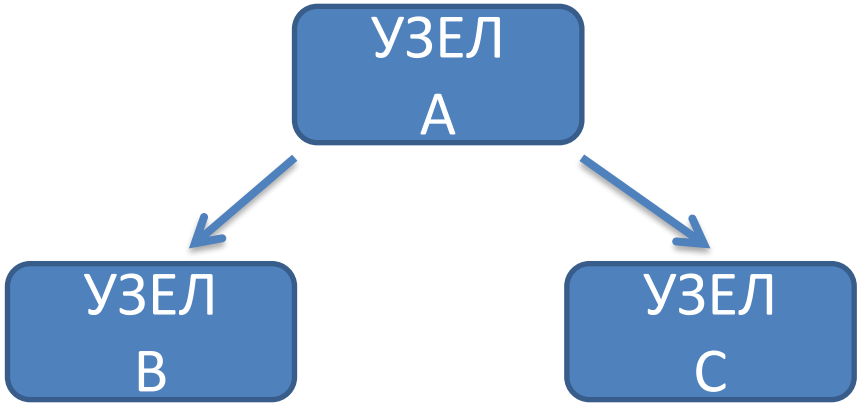
```
map<int, float>  data;
```
← Объявляем пару int, float

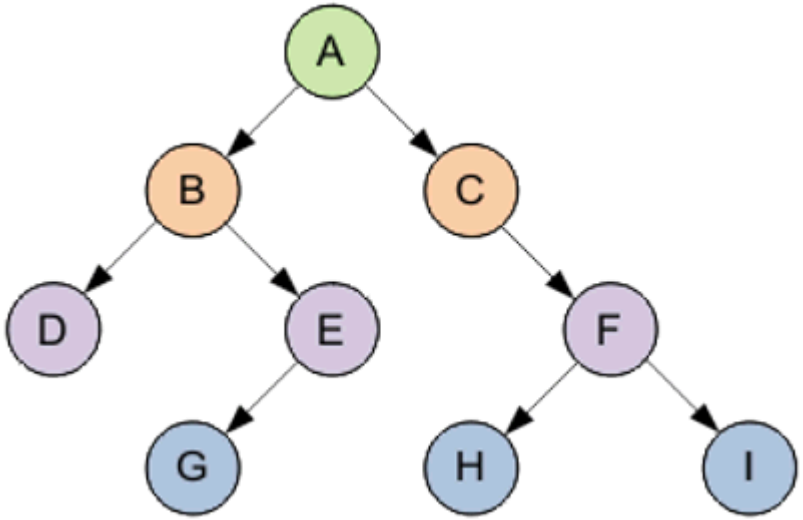
```
data[4] = 1;
data[2] = -3.1415;
data[-8]= 2.71828;
```
← Заносим дробные числа

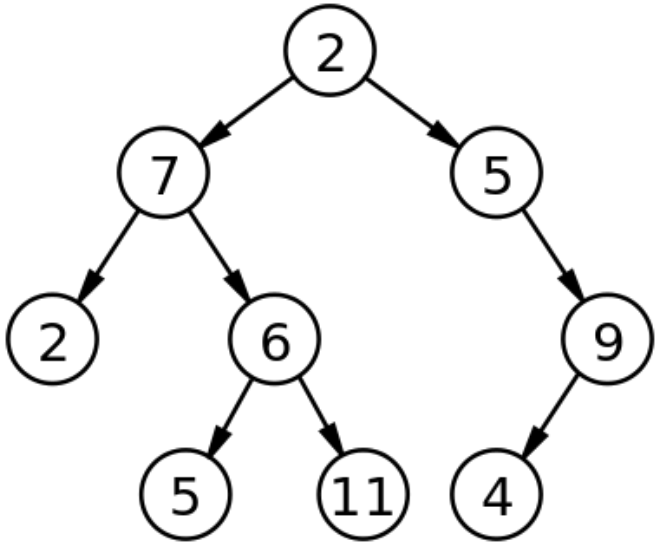
Обращаемся к ключу первого элемента

```
std::cout << (*data.begin()).first << std::endl;
```

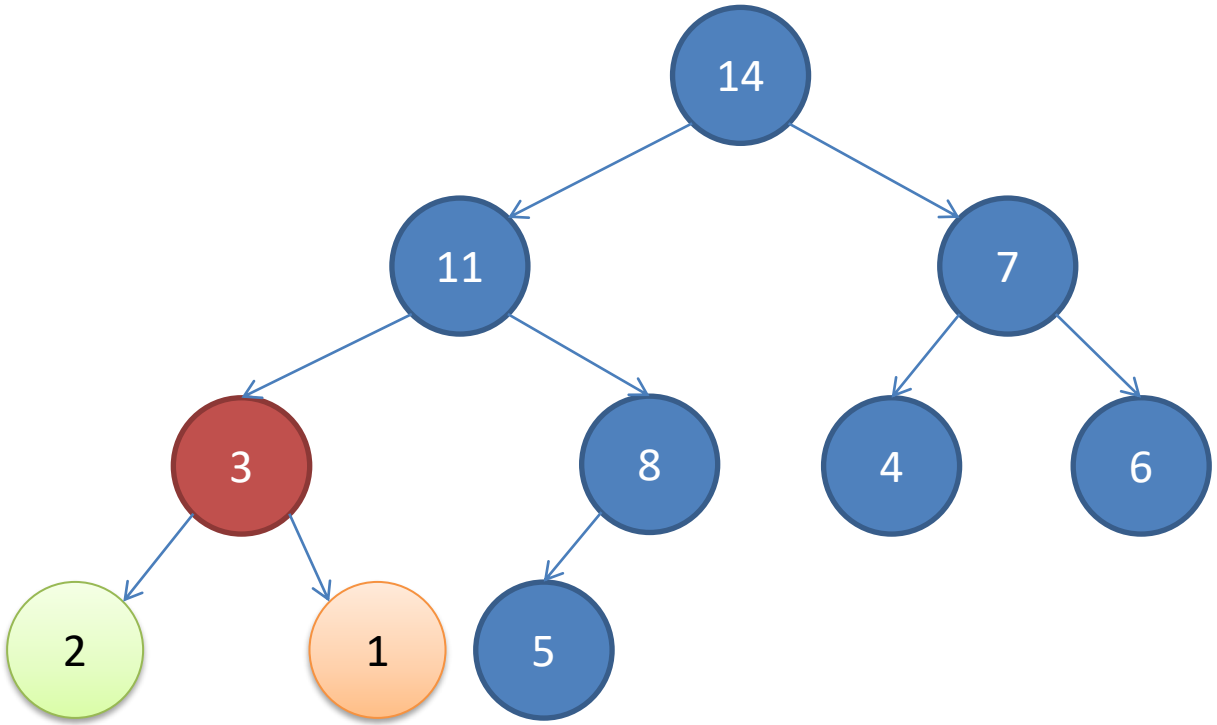
Бинарные (двоичные) деревья

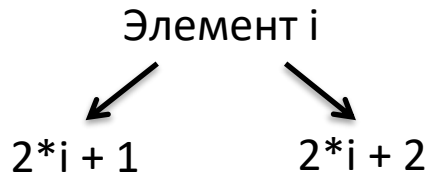


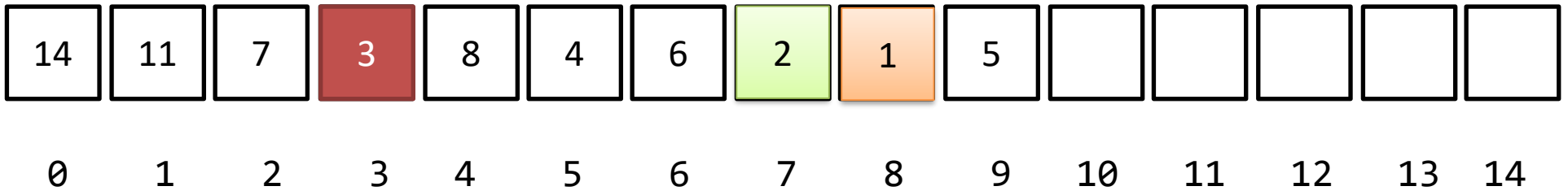




Бинарная куча: реализация на массивах







Бинарные деревья поиска (BST, Binary Search Tree) (1)