Лекция №12. Динамические структуры данных

- Понятие динамических структур данных
- Функции выделения/освобождения памяти
- Динамический массив
- Стандартные контейнеры
- Операции над контейнерами

Понятие динамических структур данных

Динамические структуры данных – это структуры данных, память под которые выделяется и освобождается по мере необходимости

Примеры статических и динамических структур данных

Статические структуры данных:

- массив гласных букв
- таблица умножения (матрица 9x9)

Динамические структуры данных:

- список сообщений в Internet-форуме
- дерево предков и потомков

Сравнение статических и динамических структур данных

Статические структуры данных:

- память выделяется на этапе компиляции
- размер памяти остается неизменным в процессе работы программы

Динамические структуры данных:

- память выделяется в процессе работы программы (на этапе компиляции размер занимаемой памяти практически равен нулю)
- размер памяти меняется в процессе работы программы

Преимущества и недостатки динамических структур данных

Преимущества:

- размер занимаемой памяти всегда соответствует объему хранимой информации (память то выделяется, то освобождается)
- объем хранимой информации практически не ограничен

Недостатки:

- более сложный способ работы с динамическими структурами данных

Функции выделения/освобождения памяти, используемые при работе с динамическими структурами данных

`void *malloc(`
`  size_t size`
`);`
	Выделяет непрерывный блок памяти размером **size** байт. Возвращает либо адрес выделенного блока памяти, либо **NULL**, если память не выделена

`void *calloc(`
`  size_t num,`
`  size_t size`
`);`
	Выделяет массив из **num** элементов размером **size** байт каждый. Возвращает либо адрес выделенного блока памяти, либо **NULL**, если память не выделена

`void *realloc(`
`  void *memblock,`
`  size_t size`
`);`
	Расширяет или сокращает блок памяти **memblock** до новых размеров **size**. Возвращает либо адрес вновь выделенного блока памяти, либо **NULL**, если память не выделена. Если новый блок памяти выделяется успешно, то старый освобождается автоматически. Содержимое блока памяти сохраняется, на сколько это возможно.

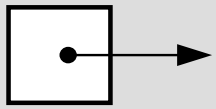
`void free(`
`  void *memblock`
`);`
	Освобождает память, занимаемую выделенным блоком памяти.

Динамический массив

Динамический массив – это массив, *память* под который *выделяется во время выполнения* программы

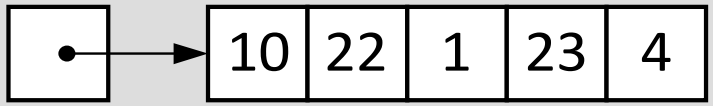
Одномерный массив на этапе компиляции

dynamicArray



Одномерный массив на этапе выполнения программы

dynamicArray



Операции над динамическим массивом

- Выделение памяти под массив
- Обращение к элементам массива
- Освобождение памяти

Выделение памяти под динамический массив

- На этапе компиляции определяется только указатель на первый элемент массива
- Выделение памяти происходит на этапе выполнения программы, когда известно, сколько элементов должно храниться в массиве
- Под массив выделяется непрерывный участок памяти
- Для выделения памяти используются библиотечные функции **`calloc()`** или **`malloc()`**

Задание

Запросить у пользователя кол-во элементов **N** массива **arr**. Выделить память под динамический массив **arr**.

Выделение памяти под динамический массив

```
int N;              // размер массива
int *arr;           // указатель на первый элемент
                    // динамического массива

// Запрашиваем размер массива
scanf("%d", &N);

// Выделяем память под массив
arr= (int *)calloc(N, sizeof(int));
```

Обращение к элементам динамического массива

Обращение к элементам динамического массива выполняется таким же образом, как и к элементам статического массива , т.к. в динамическом массиве:

- имеется указатель на первый элемент массива;
- элементы массива располагаются в памяти последовательно друг за другом.

Задание

Запросить у пользователя два целых числа и записать их в первый и последний элементы динамического массива **arr**. Использовать операцию **[]**.

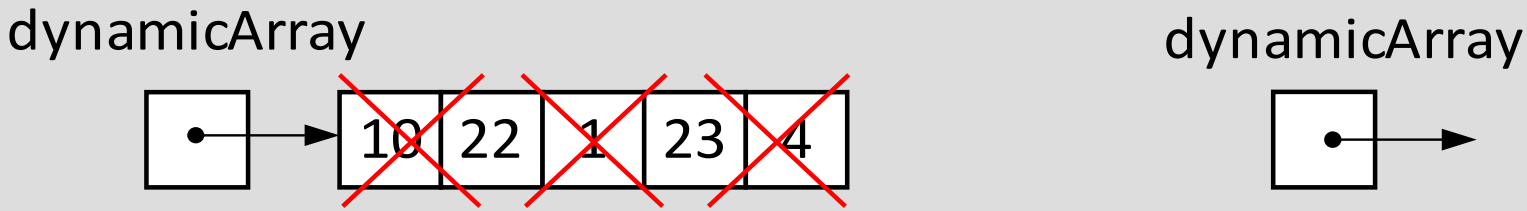
Поменять местами первый и последний элементы массива. Использовать операцию *****.

Обращение к элементам динамического массива

```
// Запрашиваем элементы массива
scanf("%d", &arr[0]);       // первый
scanf("%d", &arr[N-1]);     // последний

// Меняем местами элементы массива
int tmp= *arr;
*arr= *(arr+N-1);
*(arr+N-1)= tmp;
```

Освобождение памяти, занимаемой динамическим массивом

- Память, выделенная под динамический массив, доступна в течение жизни всей программы
- Если динамический массив становиться ненужным, то его следует уничтожить
- Для освобождения памяти, занимаемой массивом, используется библиотечная функция **`free()`**



Задание

Освободить память, занимаемую динамическим массивов **arr**.

Уничтожение динамического массива

```
// Освобождаем память
free(arr);

// Предотвращаем обращение к массиву
arr= NULL;
```

Стандартные контейнеры

Во многих языках программирования имеются готовые динамические структуры различного назначения. Их еще принято называть контейнерами.

Операции над контейнерами

- Добавление данных в контейнер
- Получение информации о кол-ве элементов в контейнере
- Обращение к данным, хранящимся в контейнере
- Извлечение данных из контейнера

Пример операций над контейнером

```
// Студент
struct TStudent
{
  char FIO[51];       // ФИО
  float avg_rating;   // средний рейтинг
};

// Добавить студента в группу
int addStudent(int group, struct TStudent *student);

// Вернуть кол-во студентов
int getCount(int group);

// Вернуть студента
struct TStudent *getStudent(int group, int student);

// Извлечь студента из группы
struct TStudent *removeStudent(int group, int student);
```

Добавление данных в контейнер

- Добавление данных по значению
- Добавление данных по адресу

Добавление данных в контейнер

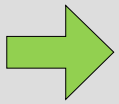
Добавление данных по значению:

- программист передает в контейнер значение для хранения
- контейнер самостоятельно выделяет память под новый элемент
- контейнер самостоятельно копирует в новый элемент переданное значение

Иванов	Петров	...	Сидоров
23.7	33.3	...	21.0

+

Козлов
41.6



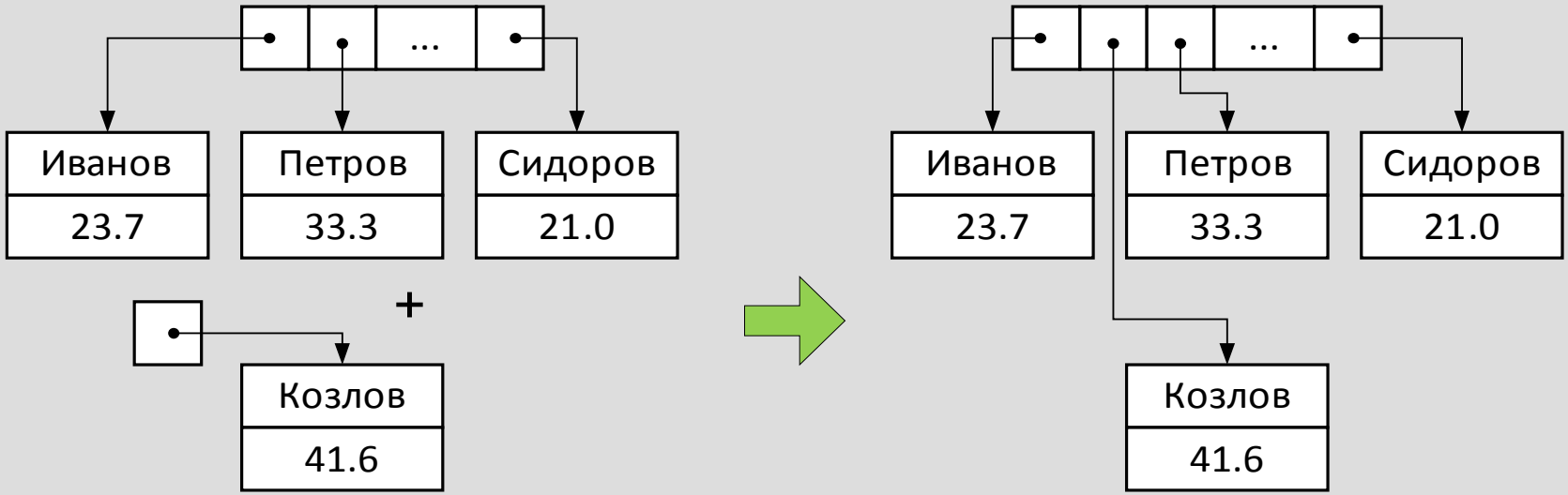
Иванов	Козлов	Петров	...	Сидоров
23.7	41.6	33.3	...	21.0

Козлов
41.6

Добавление данных в контейнер

Добавление данных по адресу:

- программист динамически выделяется память и записывает в нее необходимые данные
- в контейнер передается адрес выделенной памяти (фактически в контейнере хранятся только указатели)



Задание

Добавить в группу **160** студента **Филипова А.Б.**, у которого средний рейтинг равен **22.2**.
Для добавления использовать функцию, которая добавляет данные в контейнер по адресу

```
// Добавить студента в группу
// Входные данные:
// group – номер группы студента
// student – указатель на добавляемого студента
// Выходные данные:
// индекс добавленного студента; если студент не
// добавлен, то возвращается -1
int addStudent(int group, struct TStudent *student);
```

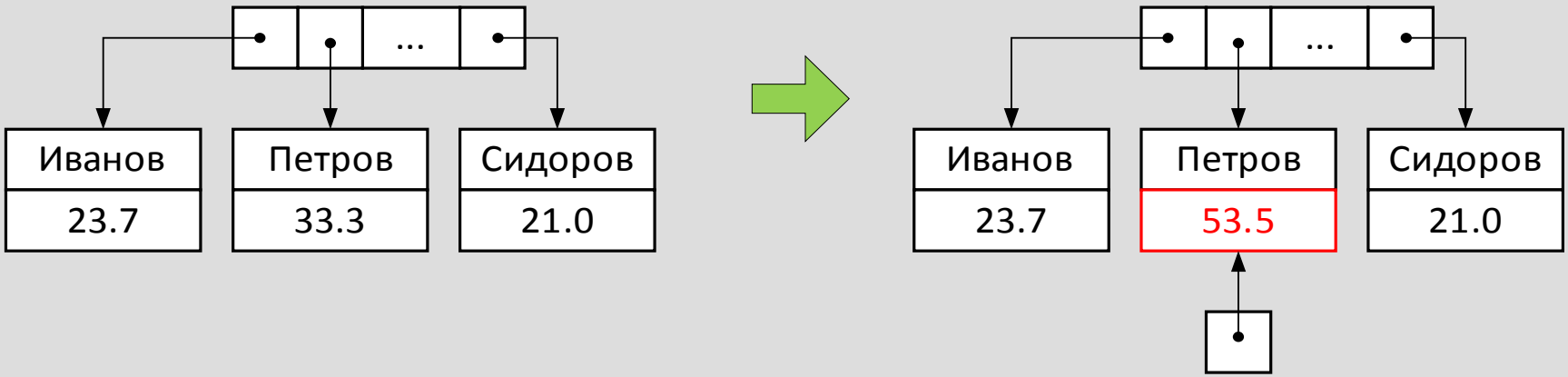
Добавление данных в контейнер по ссылке

```
// Выделяем память под нового студента
struct TStudent *new_student=
          (struct TStudent *)malloc(sizeof(struct Tstudent));

// Заполняем студента
strcpy(new_student->FIO, "Филипов А.Б.");
new_student->avg_rating= 22.2;

// Добавляем студента в группу
addStudent(160, new_student);
```

Обращение к данным, хранящимся в контейнере

Обращение к элементу в контейнере:

- в контейнер передается индекс искомого элемента
- контейнер возвращает указатель на хранимый элемент
- по полученному указателю можно читать или изменять содержимое элемента



Задание

Подсчитать кол-во студентов в группе 160, имеющих рейтинг меньше 20 баллов.

Использовать функции:

```
// Вернуть кол-во студентов
// Входные данные:
// group – номер группы
// Выходные данные: кол-во студентов в группе
int getCount(int group);

// Вернуть студента
// Входные данные:
// group – номер группы
// student – индекс студента в группе
// Выходные данные: студент с заданным индексом, возвращет
// NULL, если студент не найден
struct TStudent *getStudent(int group, int student);
```

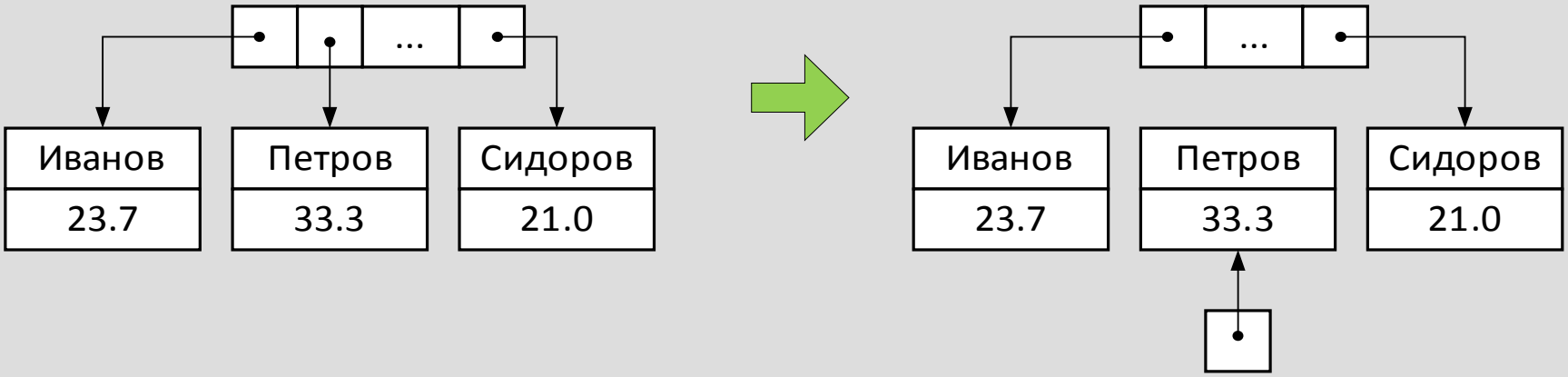
Чтение данных, хранящихся в контейнере

```
struct TStudent *student;          // текущий студент
int studentCount= getCount(160);// кол-во студентов в группе
int low_rating;                    // кол-во студентов с низким
                                   // рейтингом

low_rating= 0;
for(int i= 0; i < studentCount; i++)
{
  if(getStudent(160, i)->avg_rating < 20)
  { low_rating++; }
}
```

Извлечение данных из контейнера

Извлечение элемента из контейнера:

- в контейнер передается индекс извлекаемого элемента
- контейнер удаляет указатель из контейнера и возвращает его копию (элемент фактически удаляется из контейнера)
- через полученный указатель можно обратиться к данным или уничтожить их



Задание

Отчислить из группы 160 последнего студента.

Использовать функцию:

```
// Извлечь студента из группы
// Входные данные:
// group - номер группы
// student - индекс студента в группе
// Выходные данные: студент с заданным индексом, возвращет
// NULL, если студент не найден
struct TStudent *removeStudent(int group, int student);
```

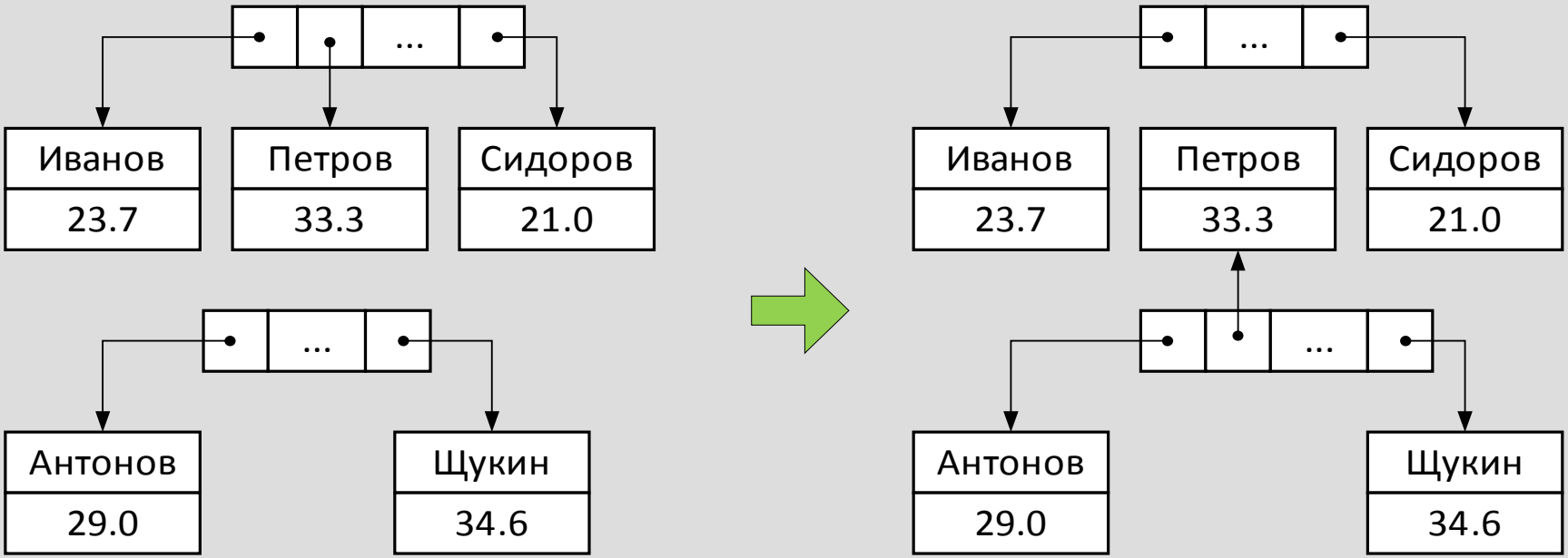
Извлечение данных из контейнера

```
struct TStudent *student;       // отчисляемый студент

// Удаляем последнего студента из группы
student= removeStudent(160, getCount(160)-1);

// Освобождаем память, занимаемую студентом
if(student != NULL)
{
  free(student);
  student= NULL;
}
```

Перемещение данных между контейнерами

Перемещение элемента из одного контейнера в другой:

- элемент извлекается из первого контейнера, возвращается указатель на этот элемент
- элемент добавляется во второй контейнер, используя полученный указатель



Задание

Перевести из группы **160** в группу **161** второго студента

Перемещение данных между контейнерами

```
struct TStudent *student;       // студент для перевода

// Удаляем второго студента из группы 160
student= removeStudent(160, 1);

// Добавляем студента в группу 161
if(student != NULL)
{ addStudent(161, student); }
```