

Лекция №14

Динамические данные

Виды памяти

Существует три вида памяти: статическая, стековая и динамическая.

Статическая память выделяется еще до начала работы программы, на стадии компиляции и сборки.

Статическая память

Существуют два типа статических переменных:

Глобальные переменные (определенные вне функций):

```
...  
int max_num = 100;
```

```
...  
void main() {  
...  
}
```

Статические переменные (определенные со словом `static`):

```
void main() {  
...  
static int max = 100;  
...  
}
```

Локальные переменные

Локальные (или стековые) переменные – это переменные определенные внутри функции (или блока):

```
void my_func() {  
    ...  
    int n = 0;  
    ...  
    if (n != 0) {  
        int a[] = {0, 2, 4, 5};  
        ...  
    }  
    ...  
}
```

Память выделяется в момент входа в функцию или блок и освобождается в момент выхода из функции или блока.

Динамическая память

Недостаток статической или локальной памяти:
количество выделяемой памяти вычисляется на
этапе компиляции и сборки.

Использование динамической памяти позволяет
избавиться от данного ограничения.

Выделение и освобождение памяти

Выделение памяти:

```
void* malloc(size_t n);
```

n – размер памяти в байтах

возвращаемое значение: указатель на выделенную
память

Освобождение памяти:

```
void free(void *p);
```

p – указатель на память, которую необходимо
освободить

Динамические массивы

Пример. Ввести с клавиатуры n чисел (n задается пользователем) и вывести их в обратном порядке.

Неправильный способ решения задачи (с использованием локальной переменной массива) :

```
void main() {
    int n,i;
    scanf("%d", &n); /* вводим кол-во чисел */
    int a[n]; /* ошибка. */

    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for (i = n-1; n >=0; i--)
        printf("%5d", a[n]);
}
```

Динамические массивы

Правильный способ решения задачи (с использованием динамической переменной массива):

```
void main() {
    int n,i;
    scanf("%d", &n); /* вводим кол-во чисел */

    /* выделяем память под массив */
    int *a = (int*)malloc(n * sizeof(int));

    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
    for (i = n-1; n >=0; i--)
        printf("%5d", a[n]);

    free(a); /* освобождаем память */
}
```

Динамические структуры

Выделение памяти под структуру:

```
struct <тэг структуры> *<имя переменной> =  
    (struct <тэг стр.>*) malloc(sizeof(<тэг стр.>));
```

Освобождение памяти:

```
free(<имя переменной>);
```

Опишем структуру:

```
struct student {  
    char name[50];  
    int grade;  
    int group;  
};
```

Массивы динамически создаваемых структур

Пример. Формирование массива из динамически создаваемых структур.

```
void main() {
    /* Объявляем массив студентов */
    struct student* a[100] = {NULL};
    int i,n;
    scanf("%d", &n); /* n - количество студентов */

    for (i = 0; i < n; i++) {
        /* резервируем память */
        student[i] = (struct student*)malloc(
            sizeof(student));
        scanf("%50s %d %d", student[i]->name,
            &student[i]->age, &student[i]->grade);
    }
    ...
}
```

Динамические массивы структур

Пример. Формирование динамического массива из структур.

```
void main() {
    /* Объявляем массив студентов */
    struct student* a;
    int i,n;
    scanf("%d", &n); /* n - количество студентов */
    /* резервируем память */
    a = (struct student* a) malloc( n * sizeof(struct
student a));

    for (i = 0; i < n; i++) {
        scanf("%50s %d %d", student[i].name,
            &student[i].age, &student[i].grade);
    }
    ...
}
```

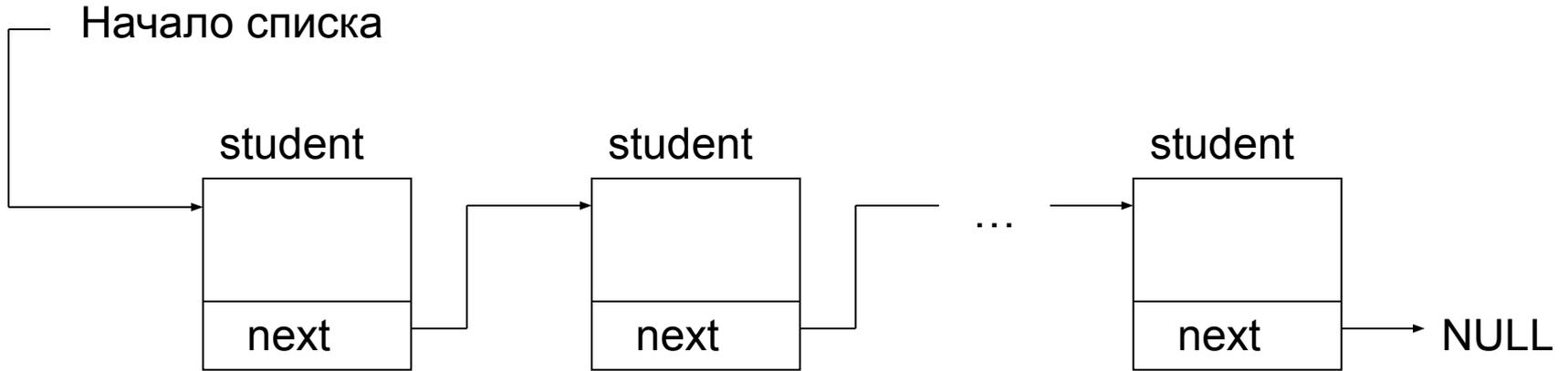
Динамические структуры данных

Использование динамической памяти позволяет создавать динамические структуры данных:

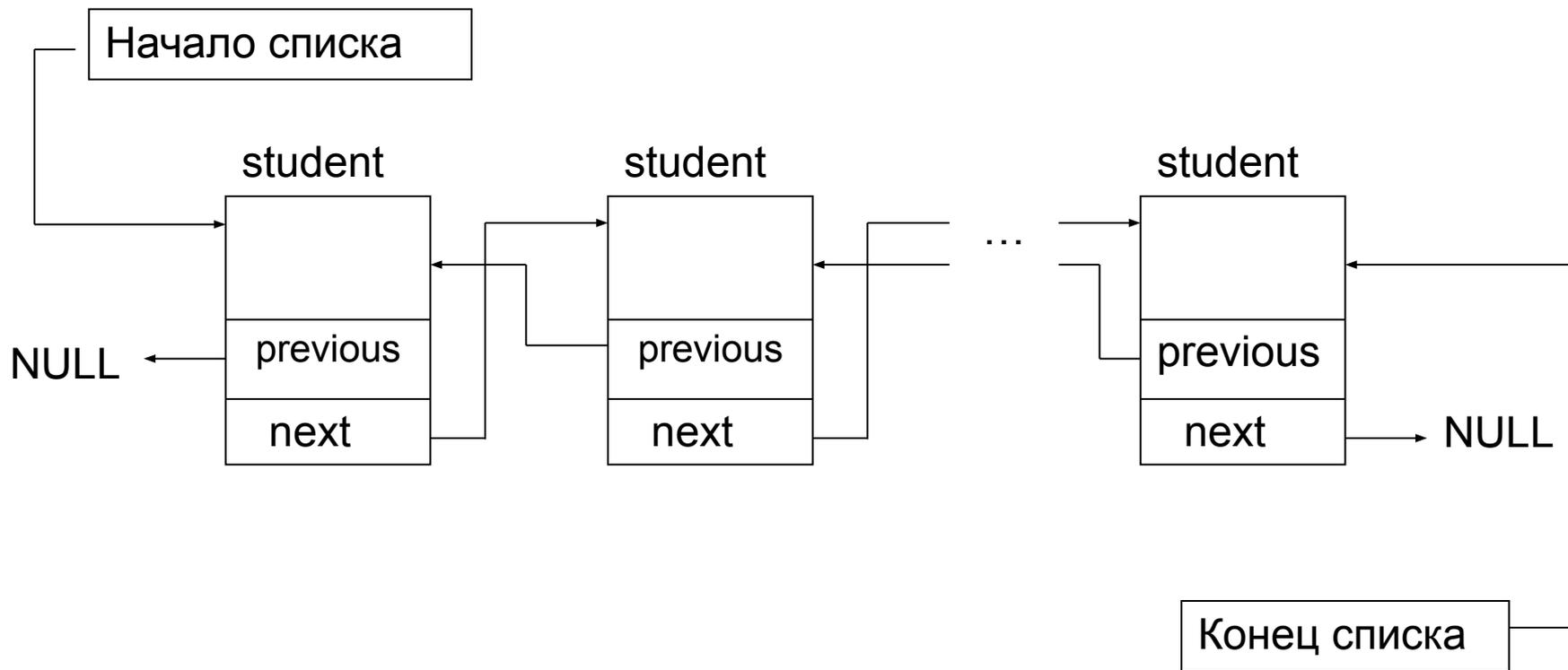
- Списки (однонаправленные и двунаправленные)
- Деревья

```
struct student {  
    char name[50];  
    int grade;  
    int group;  
    struct student* next; /* указывает на следующую  
                           структуру */  
};
```

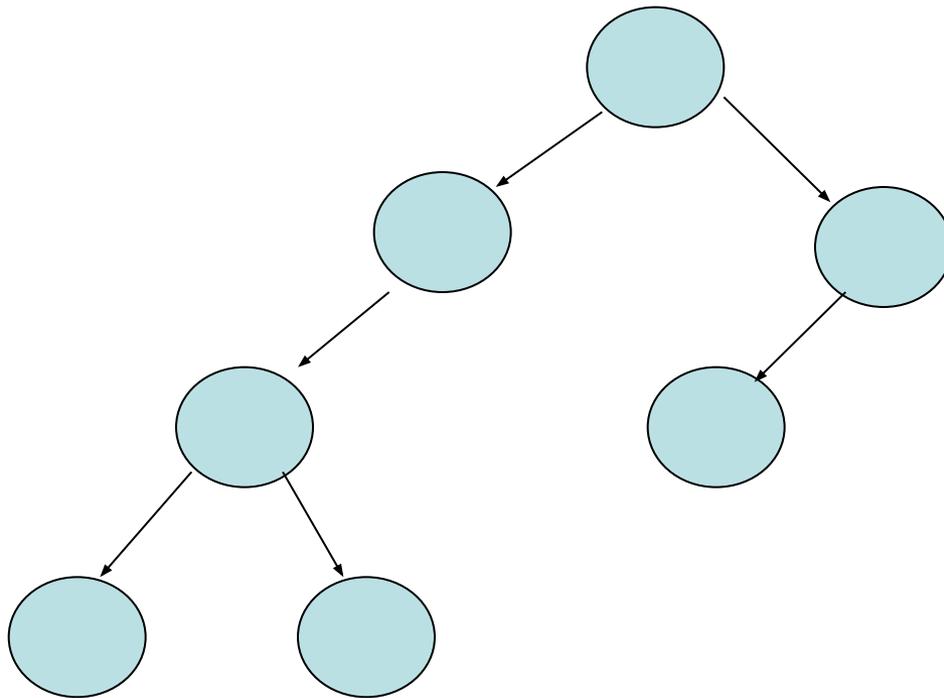
Однонаправленный (односвязный) СПИСОК



Двунаправленный список



Бинарные деревья



«Дорожная карта» Что дальше?

