Программирование на C++

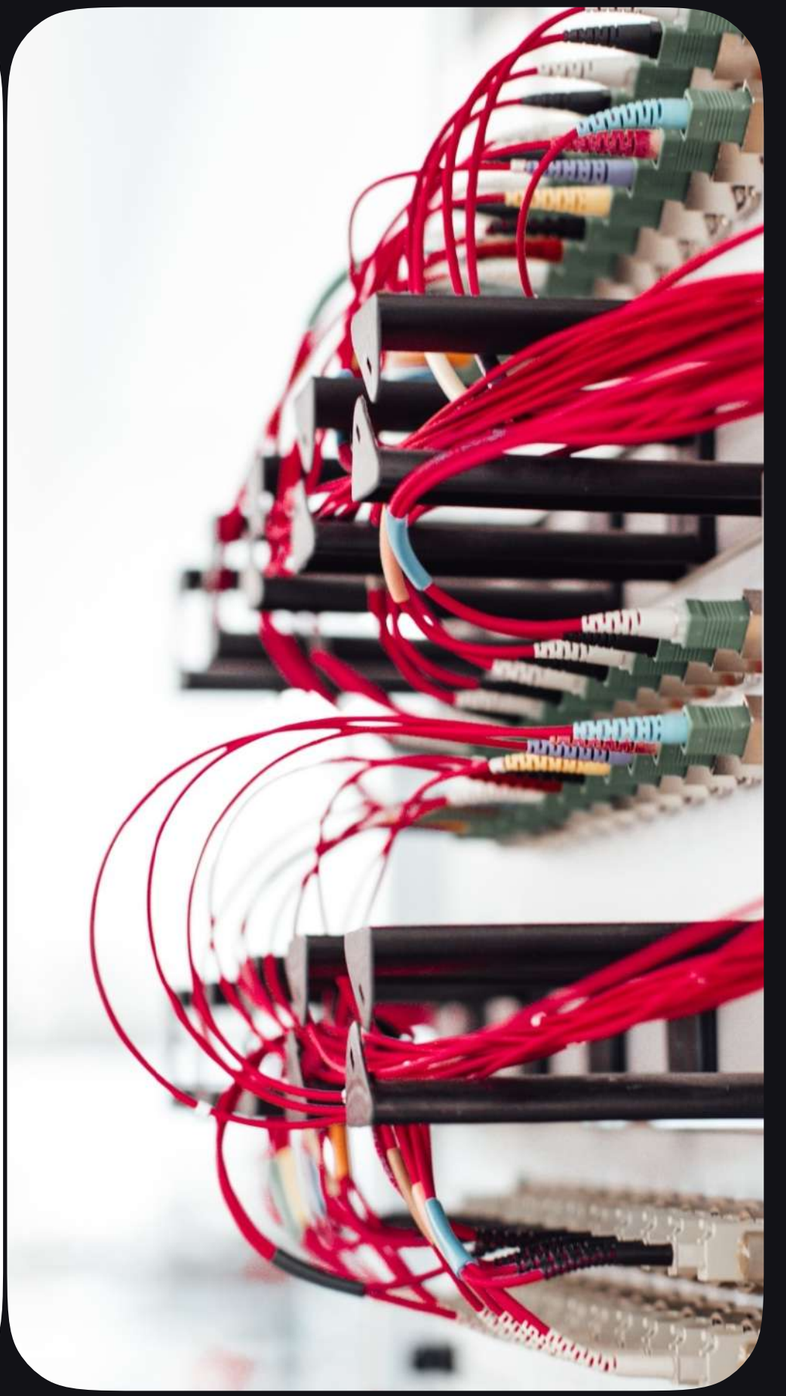
Минцифры России

UCHi DOMA

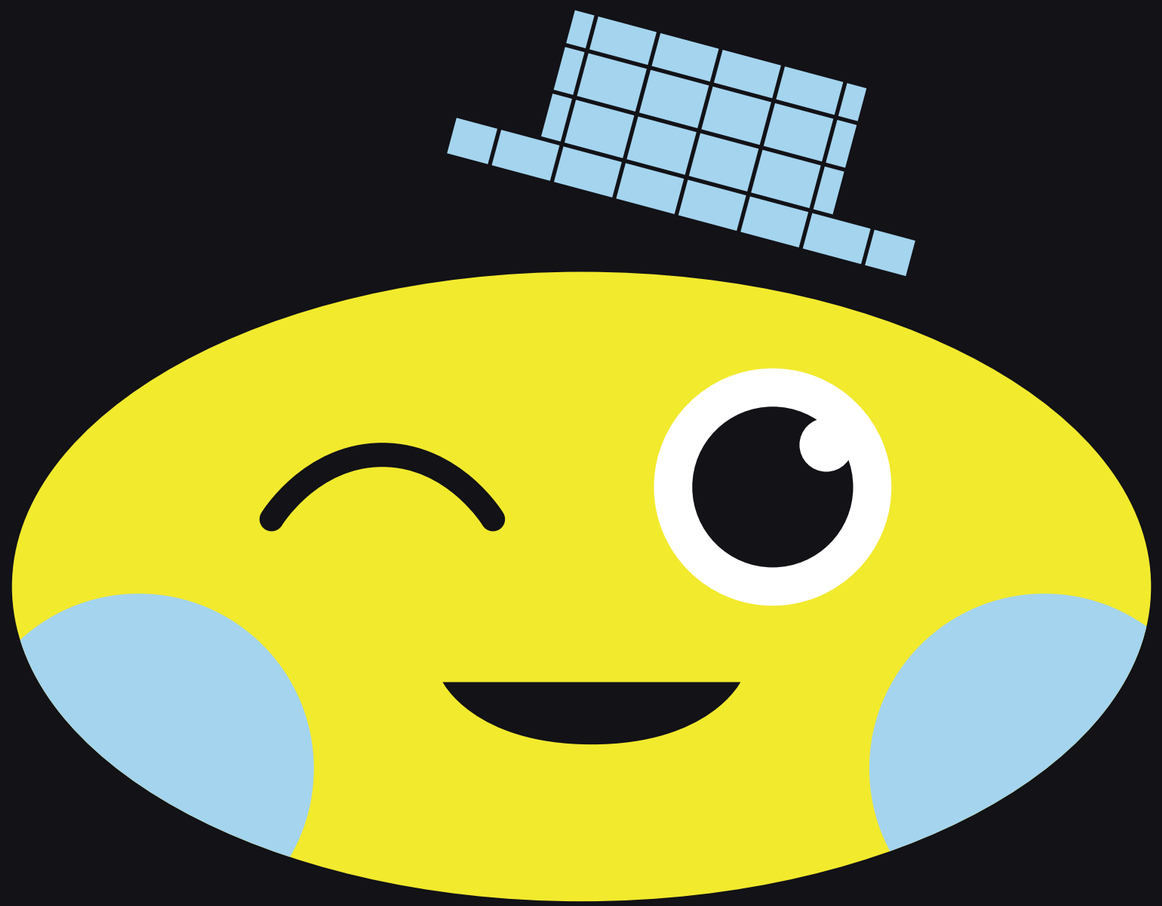
20.35 УНИВЕРСИТЕТ

Модуль 2. Урок 12

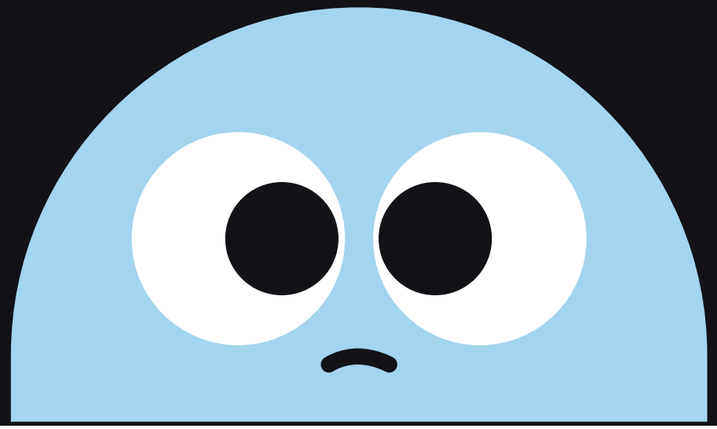
Структуры с ссылками на себя



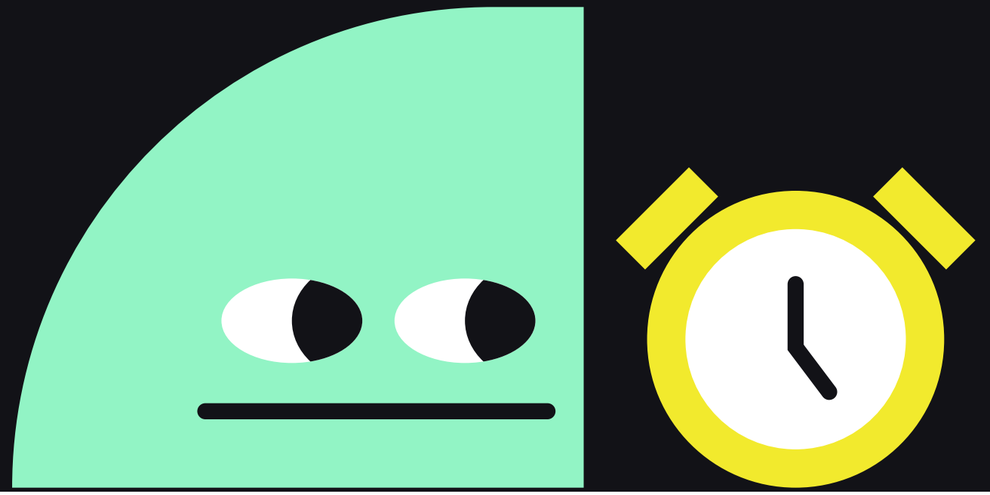
Привет!



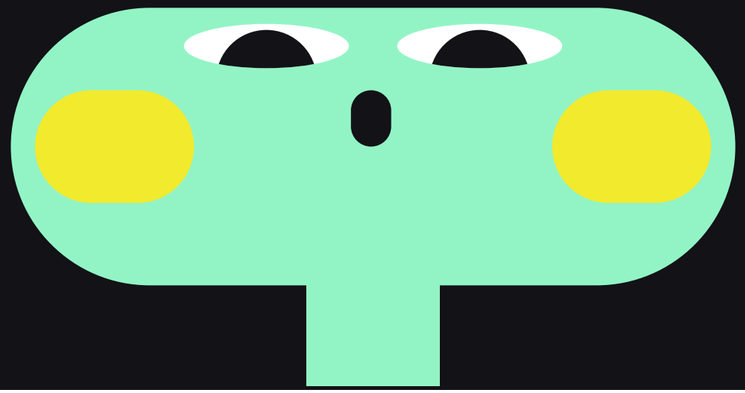
проверка готовности



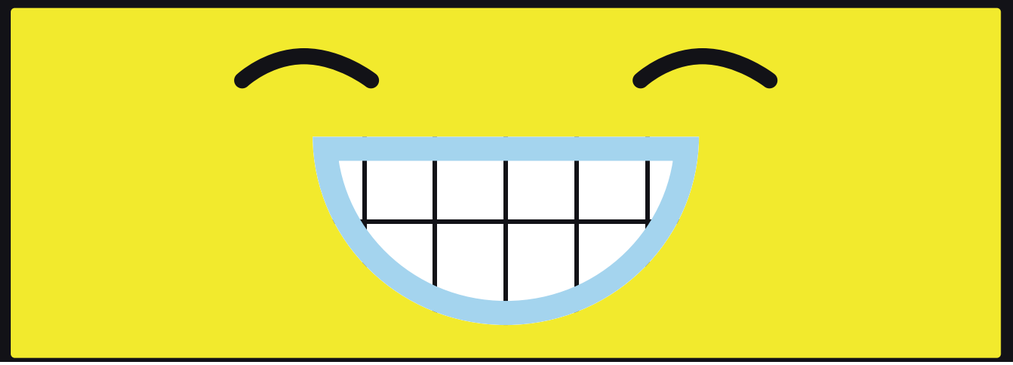
Видим и слышим друг друга без помех



Не опаздываем и не отвлекаемся



Сидим прямо



Улыбаемся, если всё ок

Как домашка?

- Какие были трудности?
- Какие остались вопросы?
- Сколько заданий выполнено?

Разомнёмся

Что будет выведено на экран в результате работы программы?

```c
#include <stdlib.h>
#include <malloc.h>
struct book
{
  char title[15];
  char author[15];
  int value;
};
int main()
{
  struct book lib={"Сказки","Пушкин",100};
  struct book* ptr_lib=&lib;
  printf("%d",ptr_lib->value);
  return 0;
}
```

Разомнёмся

Что будет выведено на экран в результате работы программы?

```c
#include <stdlib.h>
#include <malloc.h>
struct book
{
  char title[15];
  char author[15];
  int value;
};
int main()
{
  struct book lib={"Сказки","Пушкин",100};
  struct book* ptr_lib=&lib;
  printf("%d",ptr_lib->value);
  return 0;
}
```

Результат работы программы:

```
100
```

Вопрос

Предположим, есть массив структур с данными о книгах, массив отсортирован по алфавиту по возрастанию (по Авторам).

Появилась новая книга, как эффективно добавить её в массив?

Цели урока

- изучить структуры с ссылками на себя
- отработать на практике составление алгоритмов со структурами с ссылками на самих себя на Си

Структуры, ссылающиеся на себя

Структуры, ссылающиеся на себя, содержат в качестве элемента указатель, который ссылается на структуру того же типа.

```
struct Student {
  char fio[80];
  int year;
  struct Student *next;
};
```

Структуры, ссылающиеся на себя

Структура типа struct Student содержит указатель next, который указывает на структуру того же типа struct Student.

Указатель next называют связкой, так как его используют для того, чтобы связать структуру типа struct Student с другой структурой того же типа.

```
struct Student {
  char fio[80];
  int year;
  struct Student *next;
};
```

Структуры, ссылающиеся на себя, могут связываться вместе для образования полезных структур данных, таких как списки, очереди, стеки и деревья.

Связанные списки

Структурной особенностью списка является то, что его элементы линейно упорядочены в соответствии с их позицией в списке.

Примеры списков: список студентов группы, список призёров олимпиады, список документов для представления в приёмную комиссию, список литературы для самостоятельного чтения и т.п.

Связанные списки

Связанный список — это структура данных, в которой элементы следуют в некотором порядке. Однако, в отличие от массива, этот порядок определяется указателями, связывающими элементы списка в линейную цепочку.

Стеки и очереди — это специальные разновидности связанных списков.

Список — это динамическая структура данных. Длина списка при необходимости может увеличиваться или уменьшаться. Размер списка может увеличиваться до тех пор, пока имеется свободная память.

Связанные списки

Достоинство организации данных в виде линейного связанного списка в том, что нет ограничения на длину списка и эффективно используется память (используется ровно столько памяти, сколько надо плюс накладные расходы — поля адресов).

Недостатки — необходимость хранить дополнительную информацию (поля адресов) и отсутствие прямого доступа к i-му элементу списка, как в массивах по индексу (для получения доступа к определенному элементу списка надо всегда просматривать список с начала или с конца).

Операции вставки и удаления для связанных списков требуют меньше действий, чем для массивов, так как не требуется перемещать элементы, следующие за вставляемым (или удаляемым).

Использование связанных списков больше подходит для задач, в которых изменения в списках происходят часто.

Связанные списки

Для работы со списками на языке С потребуется их программная реализация

Для этого необходимо:

1. сконструировать средствами С структуру данных, которая будет представлять в программе список (в этой структуре будут храниться элементы списка)
2. описать в виде функций требуемые операции над списками

Связанные списки

Для хранения отдельного элемента списка создается динамический объект — структура, состоящая из двух частей:

- основной, содержащей нужную информацию
- дополнительной, содержащей ссылку на следующий элемент списка

Связанные списки

Для возможности быстро сохранять список в файле, а также вводить из файла производят Группировку смысловых полей в отдельную структуру. Это также упрощает перенос информации между элементами списка.

Связанные списки

Соседние элементы списка располагаются в оперативной памяти произвольно относительно друг друга, в отличие от соседних компонент массива, всегда занимающих смежные участки памяти.

Такое расположение элементов облегчает операции вставки и удаления, так как нет необходимости перемещать элементы, как это делается для массивов.

Пример

Организация хранения информации о книгах в магазине в виде списка:

```
typedef struct
{ // шаблон данных элемента списка
    char name[80]; // название книги
    int kol; // количество книг
} InfoBook;

struct List { // шаблон элемента списка
    InfoBook data;
    struct List *next;
};
```

Связанные списки

- Доступ к связанному списку обеспечивается через указатель на первый элемент списка. Называется такой указатель головой списка (head).
- Поэтому программа должна иметь переменную — указатель на первый элемент списка, который равен NULL, если список пустой.
- Имея доступ к первому элементу списка, без труда можно просмотреть весь список, просто переходя по связям от одного элемента к другому.

```
struct List *head; // голова списка
```

Связанные списки

Для выделения памяти под элементы списка необходимо пользоваться любой из функций:

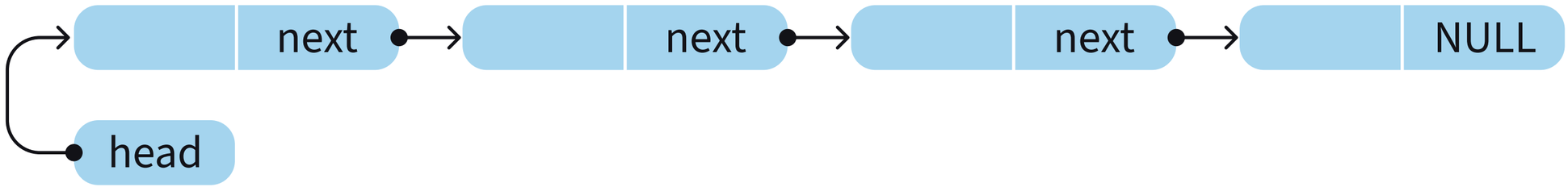
- malloc(sizeof(struct List))
- calloc(l, sizeof(struct List))

Для перехода к следующему элементу списка используется его адрес в памяти, который хранится в указателе next.

В последнем элементе списка указатель на следующий элемент имеет значение NULL — это является признаком конца списка.

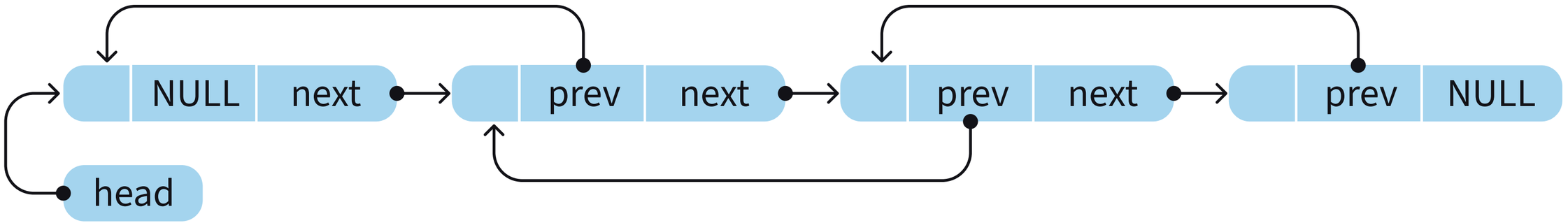
Связь данных в списке

Возможна различная связь данных в списке. Если каждый элемент списка содержит указатель на элемент, следующий непосредственно за ним, то получаемый список называют односвязным (однонаправленным).



Связь данных в списке

Если в дополнение к этому каждый элемент списка содержит указатель на элемент, следующий непосредственно перед ним, то такой список называют **двусвязным** (**двунаправленным**).

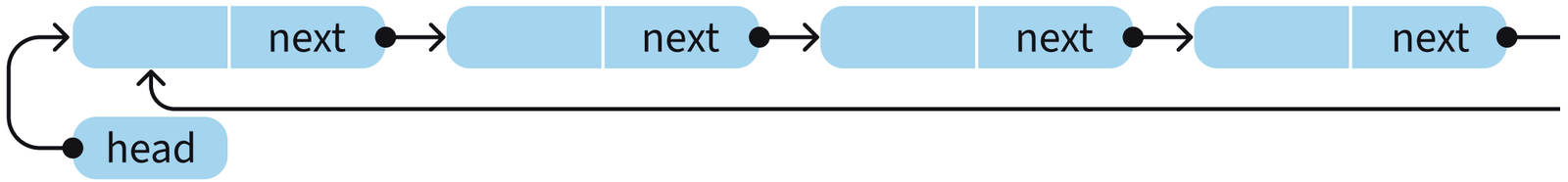


Связь данных в списке

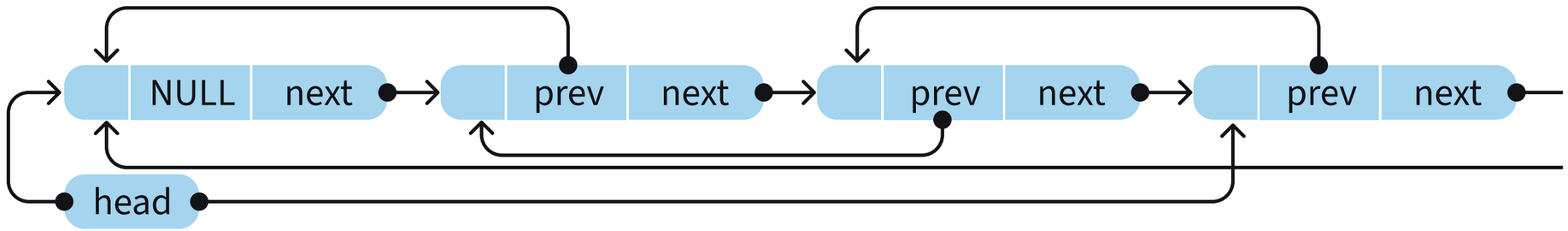
Обычно у последнего элемента двусвязного списка указатель на следующий элемент равен NULL, отмечая конец списка. Но в некоторых списках удобно, чтобы этот указатель показывал на первый элемент списка.

Таким образом, список из цепочки превращается в кольцо. Такие списки (однонаправленные и двунаправленные) называют **кольцевыми**.

Кольцевой однонаправленный список:



Кольцевой двунаправленный список:



Связанные списки

При работе со списками на практике чаще всего приходится выполнять следующие операции:

- просмотреть весь список
- найти один элемент с заданными свойствами
- вставить новый элемент в список
- удалить заданный элемент из списка
- упорядочить список в определенном порядке

Возможны и более сложные операции над линейными списками — соединить два линейных списка в один список, разбить список на два списка, создать копию списка и т.п.

Пример

Вывод всего списка

```
void printList()
{
    struct List *p = head; // стать на первый элемент списка
    while (p != NULL)
    { // пока не дошли до конца списка
    printf( "%20s %6d\n", p->data.name, p->data.kol );
    p = p->next; // перейти к следующему элементу списка
    }
}
```

Пример

Удаление списка с освобождением всей занятой им памяти

В результате получаем head=NULL: список пустой

```
void freeList()
{
struct List *p;
    while(head != NULL)
    {
        p = head->next;
        free(head);
        head = p;
    }
}
```

Пример

Поиск элемента

```
struct List *findNameList()
{
    char s[80];
    printf("Введите название книги для поиска:");
    scanf("%s", &s);
    struct List *p = head; // стали на первый элемент списка
    while (p != NULL)
    { // пока не дошли до конца списка
        if (strcmp(p->data.name, s) == 0) // нашли совпадение
            break; // закончили поиск
        p = p->next; // перешли к следующему элементу списка
    }
    if (p != NULL) // элемент найден
        printf("Такая книга есть в магазине\n”);
    else
        printf("Такой книги нет в магазине\n");
    return p;
}
```

Осуществляется линейный просмотр списка от первого элемента до тех пор, пока не будет найден искомый элемент или же до конца списка, если элемента с заданными свойствами в списке нет.

Пример

Вставка нового элемента

Вставка новых элементов в список может осуществляться:

- всегда в начало списка, всегда в конец списка
- перед заданным элементом, после заданного элемента
- в нужное место в отсортированном списке

Решение данной задачи состоит из двух этапов:

- Во-первых, необходимо создать динамический объект для вставляемого элемента списка и занести в него информационные поля
- Во-вторых, путём изменения указателей, включить новый элемент в список

Пример

Вставка нового элемента в начало списка

```
struct List *addBegibList(InfoBook a)
{
    struct List *add, *p;
    add = (struct List *)malloc(sizeof(struct List));
    if (add)
    { // память выделилась
        add->data = a; // информационная часть
        add->next = head; // новый элемент указывает на прежний первый
        head = add; // вставка в начало списка
    }
}
```

Пример

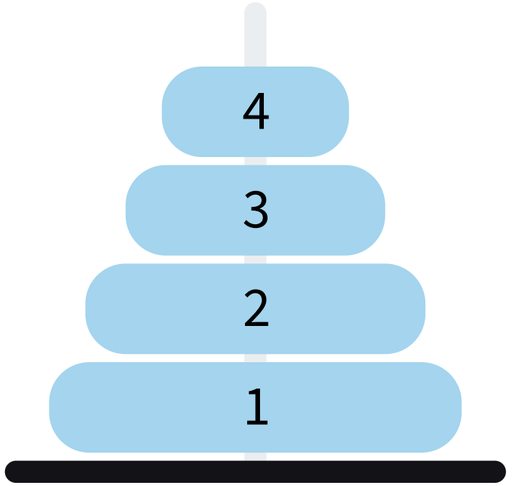
Вставка нового элемента перед элементом с заданным названием

```
struct List * addBeforeList(InfoBook a, char *s)
{
    struct List *add, *prev, *p;
    add = (struct List *)malloc(sizeof(struct List));
    if (add)
    { // память выделилась
        add->data = a; // информационная часть
        prev = NULL; // указатель на элемент перед нужным
        p = head; // поиск нужного элемента
        while (p != NULL)
        {
            if (!strcmp(p->data.name, s))
                break;
            prev = p;
            p = p->next;
        }
        if (p == head)
        { // вставка в пустой список или в начало списка
            add->next = head;
            head = add;
        }
        else
        { // вставка в середину списка или в конец списка
            add->next = p;
            prev->next = add;
        }
    }
}
```

Стек

Стеком называется структура данных, добавление или удаление элементов для которой осуществляется с помощью указателя стека в соответствии с правилом LIFO (last-in, first-out — последним пришел, первым ушел).

«Последний вошел — первый вышел»



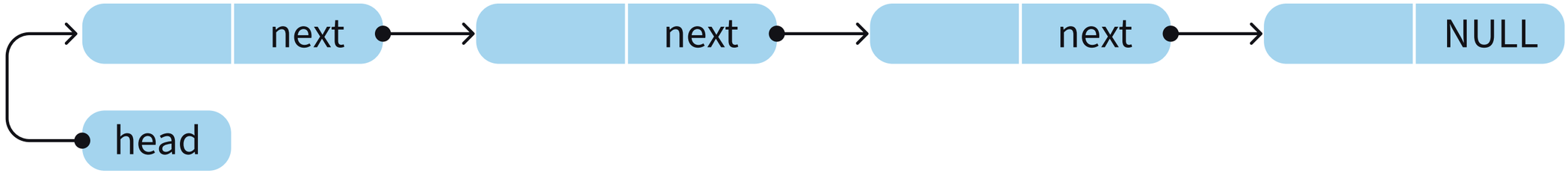
Стеком называется структура данных, добавление или удаление элементов для которой осуществляется с помощью указателя стека в соответствии с правилом LIFO (last-in, first-out — последним пришел, первым ушел).

Стек

Основные операции:

1. Добавление элемента в стек — создать новый элемент (выделить для него память и заполнить данные) и поместить его в вершину стека.

2. Удаление элемента из стека — удалить верхний элемент (на него указывает head) из стека и освободить память, которая была ему выделена.

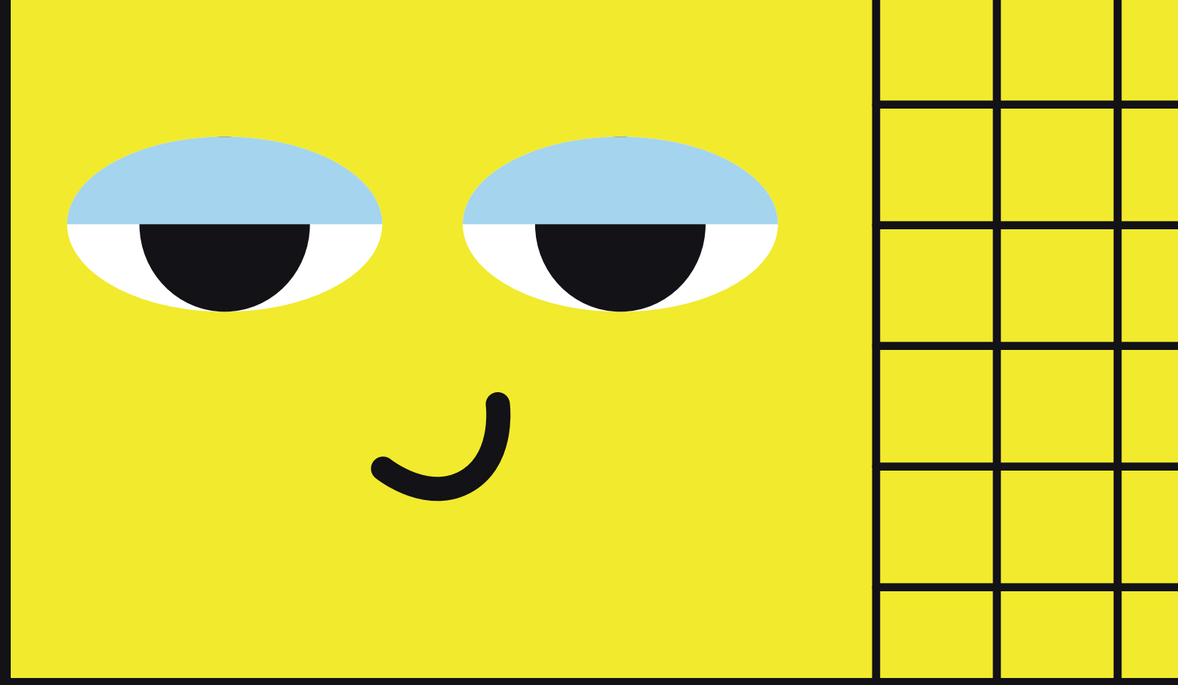


Указатель в последнем элементе стека устанавливается равным NULL, чтобы отметить нижнюю границу стека.

Пример работы со стеком

```c
#include <stddef.h>
#include <stdlib.h>
#include <malloc.h>

typedef struct
{ // шаблон данных элемента стека
    char name[80];
    int kol;
} Info;
struct Node
{ // шаблон элемента стека
    Info data;
    struct Node *next;
};
struct Node *head; // вершина стека
// добавление элемента в стек
struct Node *addStack(Info add)
{
    struct Node *p;
    p = (struct Node *)malloc(sizeof(struct Node));
    if (p)
    { // память выделена успешно
        p->data = add; // заполнение данных
        p->next = head; // ссылка на предыдущий элемент стека
        head = p; // вставка в вершину стека
    }
    return p; // возврат адреса нового элемента или NULL – ошибка добавления
}
// удаление элемента из стека
void delStack()
{
struct Node *p;
if (head != NULL)
{ // стек не пустой
    p = head->next; // запомнить адрес следующего
    free(head); // освобождение памяти
    head = p; // вершина указывает на новый элемент
}
}
// удаление всех элементов из стека (очистка стека)
// в конце работы функции: head = NULL
void freeStack()
{
struct Node *p;
while (head != NULL)
{ // пока в стеке есть элементы
    p = head->next;
    free(head);
    head = p;
}
}
// печать содержимого стека
void printStack()
{
    struct Node *p;
    printf("==================\n");
    p = head; // стать в вершину стека
    while (p != NULL)
    {
        printf("%s %d\n", p->data.name, p->data.kol);
        p = p->next; // переход к следующему элементу
    }
}
void main()
{
    int i, n = 5;
    Info s;
    head = NULL; // стек пустой
    for(i=0; i<n; i++)
    {
        scanf("%s %d", s.name,&s.kol);
        if (!addStack(s))
        {
            printf("Ошибка добавления в стек!\n");
            break;
        }
    }
    printStack(); // в стеке 5 элементов
    delStack();
    printf("Вершина стека: %s %d\n",
    head->data.name, head->data.kol);
    delStack();
    printStack(); // в стеке 3 элемента
    freeStack();
    printStack(); // в стеке 0 элементов
}
```

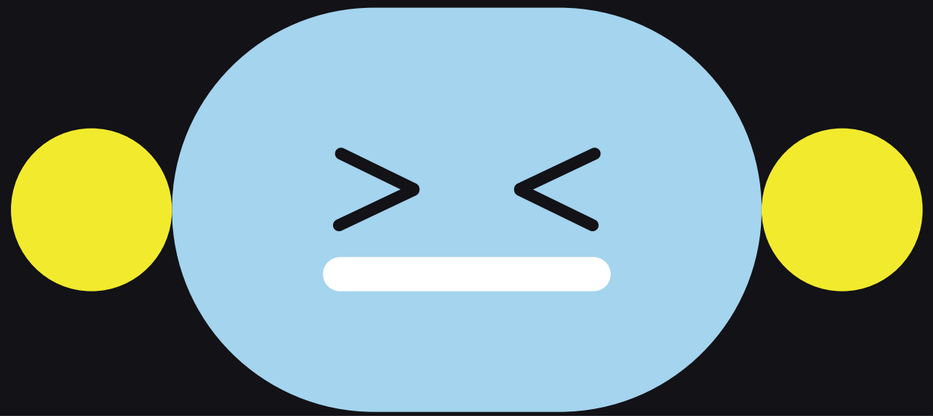


Практика

перерыв физкультминутка

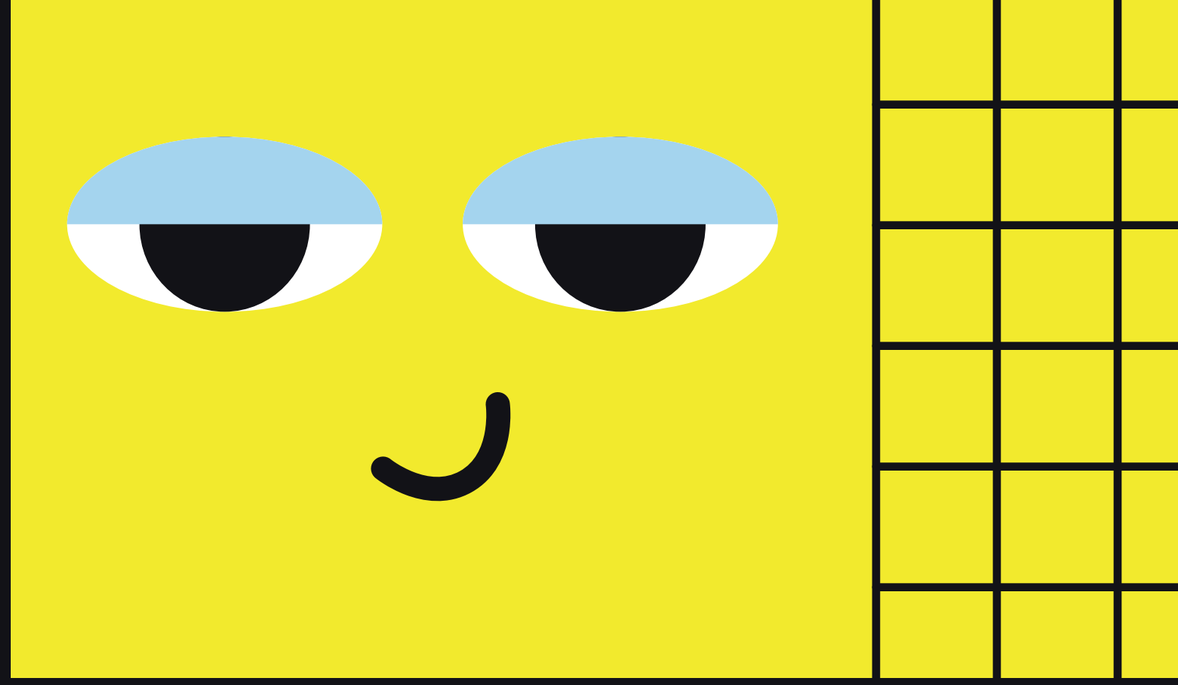
Смотрим вверх–вниз, вправо–влево

Вращаем по кругу туда–обратно



Крепко зажмуриваемся

Быстро моргаем



Практика

Закрепление

Выбери верно описанную структуру с ссылкой на себя

1

```
struct Student {
    char fio[80];
    int year;
    struct Student *next;
};
```

2

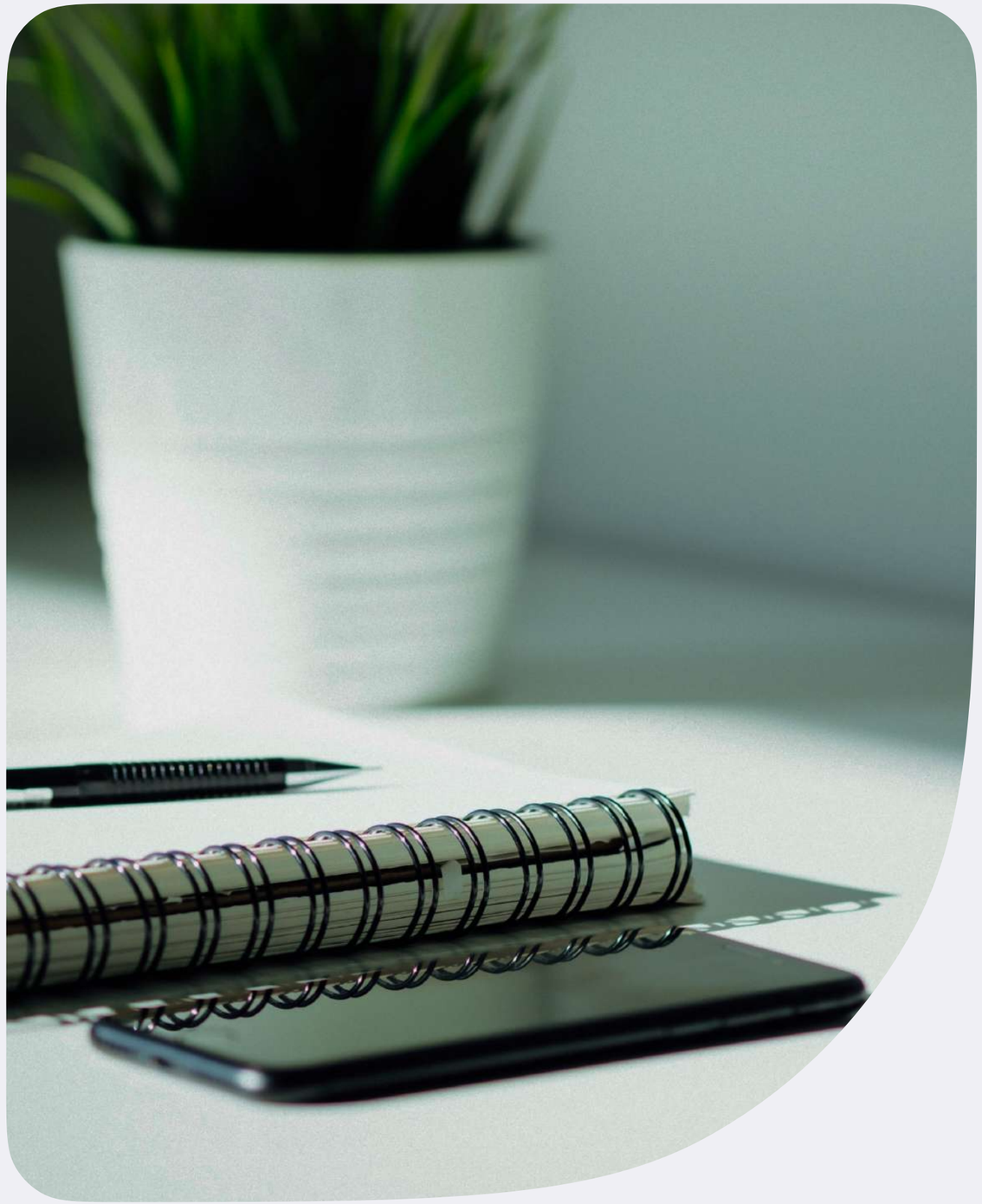
```
struct Student {
    char fio[80];
    int year;
    struct Student next;
};
```

Закрепление

Выбери верно описанную структуру с ссылкой на себя

```
struct Student {
    char fio[80];
    int year;
    struct Student *next;
};
```

```
struct Student {
    char fio[80];
    int year;
    struct Student next;
};
```

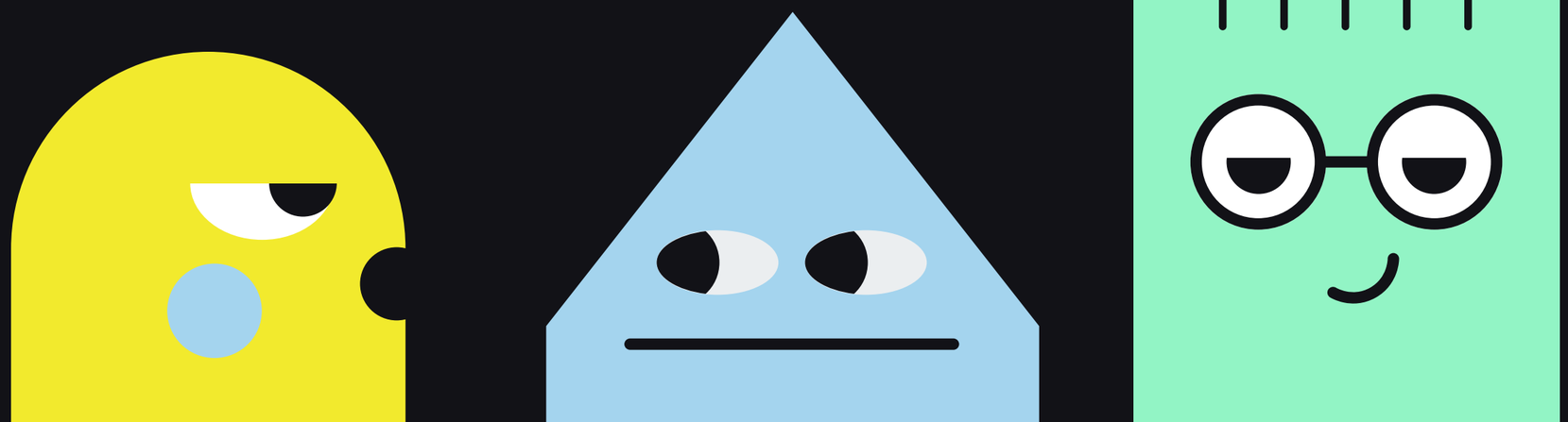


Подведём итоги

- изучили структуры с ссылками на себя
- отработали на практике составление алгоритмов со структурами с ссылками на самих себя на Си

Оцени сложность урока

1	если тебе было совсем просто
2	было достаточно просто, но ты узнал(а) что-то новое
3	было не очень просто, но достаточно комфортно, ты узнал(а) много нового
4	было сложно, ты не знал(а) ничего из материала
5	было слишком сложно, многое осталось для тебя непонятным



Домашнее задание

До встречи!