Информатика (основы программирования)

Лекция 14.

Алгоритмизация обработки абстрактных структур данных типа списков

Автор: Бабалова И.Ф.

Доцент, каф.12

2023 г.

Структуры данных (1)

- Структура – это составной объект, в который входят элементы любых типов, за исключением функций. В отличие от массива, который является однородным объектом, структура может быть неоднородной.
- Тип структуры определяется записью вида:

```
struct { список определений };
```

- В структуре обязательно должен быть указан хотя бы один компонент.
- Определение структур имеет следующий вид:

```
struct <Имя> {
    Тип имя переменной;
    Тип имя переменной;
  …
};
```

Поля данных

Структуры данных (3)

```
typedef struct data {
    char *name;
    int year;
    int month, day;
} data_1, data_2
```

В этом объявлении закладывается возможность использования только имени переменой в записи действий с компонентами структуры, без описателя `struct`:

```
data_1.year = 1976;
data_2.month = 4;
```

Абстрактные типы данных. Структуры данных (1)

- Структура – это составной объект, в который входят элементы любых типов, за исключением функций. В отличие от массива, который является однородным объектом, структура неоднородна.
- Тип структуры определяется записью вида:

```
struct { список определений };
```

- В структуре обязательно должен быть указан хотя бы один компонент.
- Определение структур имеет следующий вид:

```
struct <Имя> {
    Тип имя_переменной;
    Тип имя_переменной;
  ...
};
...
struct <Имя> имя_стр._переменной, имя_стр._переменной;
```

Поля данных

Абстрактные типы данных. Структуры данных (2)

```
typedef struct {
    double x, y;
} Point;
...
Point p1, p2, pm[10];
```

```
struct MyDate {
    int year;
    char month, day;
};
...
struct MyDate data1, data2;
```

- Переменные p1, p2 определяются как структуры, каждая из которых состоит из двух полей: x и y.
- Доступ к полям структур: p1.x или p1.y
- Переменная pm определяется как массив из десяти структур.
- Каждая из двух переменных date1, date2 состоит из трех компонентов year, month, day.
- Имя – это имя типа структуры, а не имя переменной.
- Структурные переменные – это список имен переменных, разделенный запятыми.
- В блоках программы составное имя date1.year или date1.month и т.д позволяет обрабатывать переменные

Абстрактные типы данных. Массивы структур(1)

- Для обработки элементов массивов структур используется операция доступа к элементу массива (квадратные скобки). Массивы структур широко используются в организации данных в прикладных программах и системном программном обеспечении.

```
#define NUM_EMPS 100
struct emp {
    char name[21]; char id[8]; double salary;
};
void fillarray(struct emp *, int);
int main() {
    struct emp staff[NUM_EMPS], double sum_tot = 0;
     /* Считаем некоторую сумму одного поля данных этой структуры:
При обращении к полям структуры используется знак точка:
staff.name  или  staff.id */
    for (i = 0; i < NUM_EMPS; i++) sum_salary += staff[i].salary;
```

Абстрактные типы данных. Массивы структур(2)

Другой способ объявления структур.

- Ключевое слово **typedef** позволяет объявить новый тип данных:

```
typedef struct emp {
    char name[21]; char id[8]; double salary;
} emp;
```

- В этом случае при создании переменной не надо писать слово `struct`:

```
emp staff[NUM_EMPS];
```

Организация работы с полями структуры

```
#include <stdio.h>
#include <stdlib.h>
char buf[127]; /*объявляем переменную для ввода имени*/
typedef struct emp { char name[21]; int id[8]; double salary; };
int main() {
    int size;
    scanf("%d", &size);
    /* Для одной компоненты структуры определяем все поля данных*/
    scanf("%f",prgm.salary);
    printf("salary: %.2f\n", prgm.salary);//Напечатать число в заданном формате
    scanf("%127c", buf);     // Ввод имени
    prgm.name = (char*) malloc ((strlen(buf)+1) * sizeof(char));
    strcpy(prgm.name, buf); //Из буфера в поле структуры записываем строку
    printf("Name: %s\n", prgm.name);
    scanf("%d", prgm.id); // Ввести id
    // …
}
```

Абстрактные типы данных. Линейные списочные структуры

Логическая организация линейных структур данных:

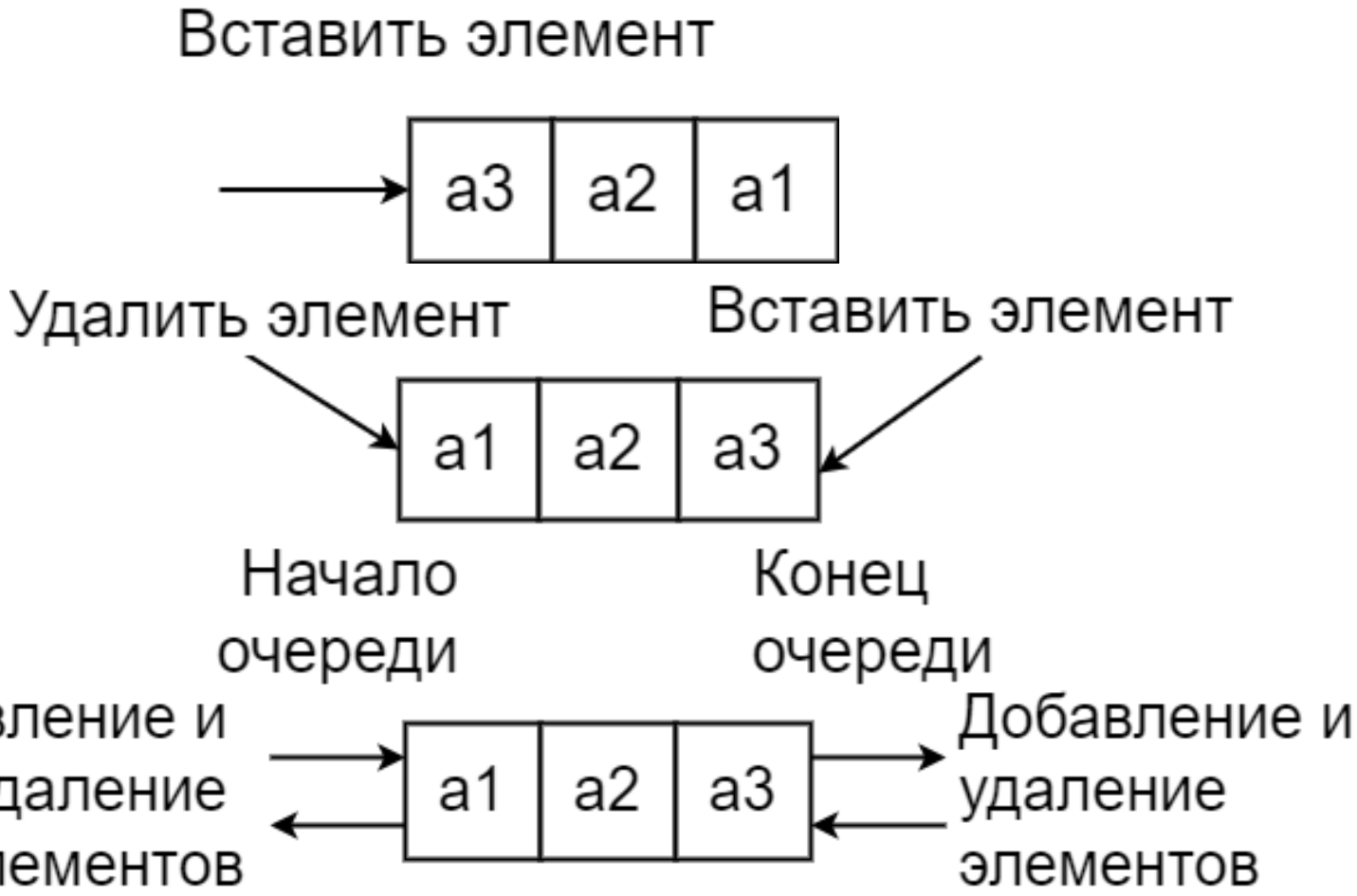
- Стеки
- Очереди
- Деки

Физическая организация линейных структур данных:

- Односвязные списки
- Двусвязные списки

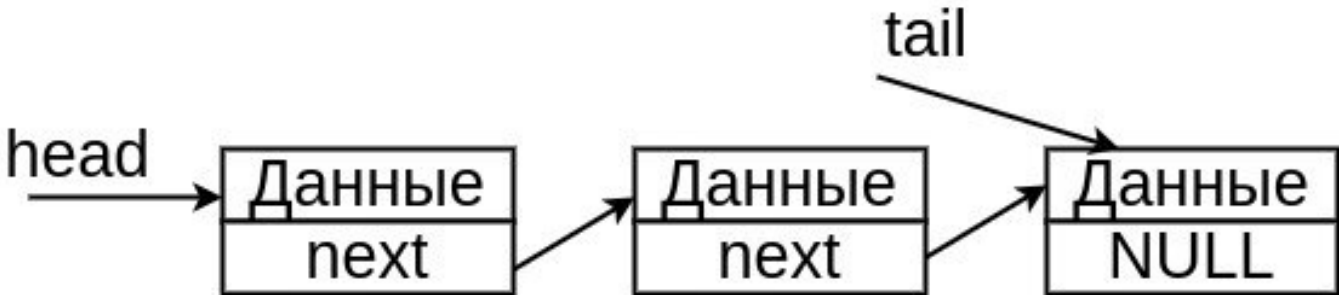
Абстрактные типы данных. Правила формирования списков

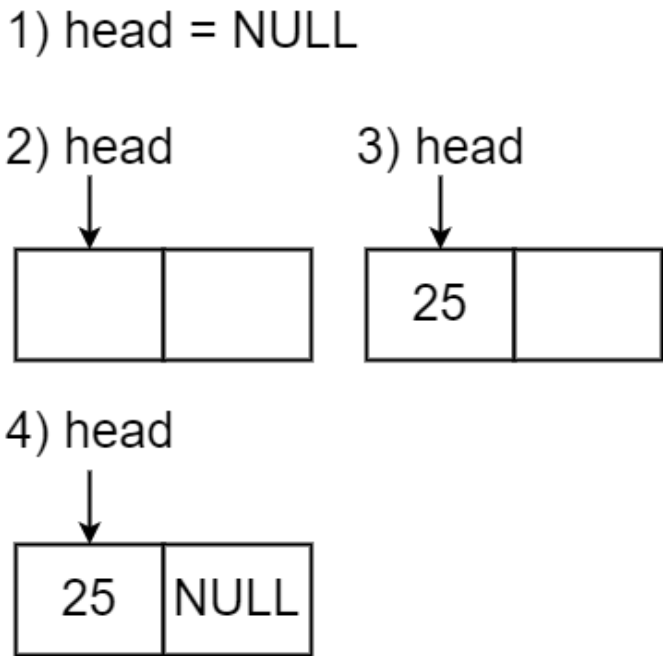
- Стек (LIFO)
- Очереди (FIFO)
- Деки



Линейные односвязные списки

- В последнем элементе хранения указатель на следующий элемент имеет значение NULL.





Правила формирования элемента списка:

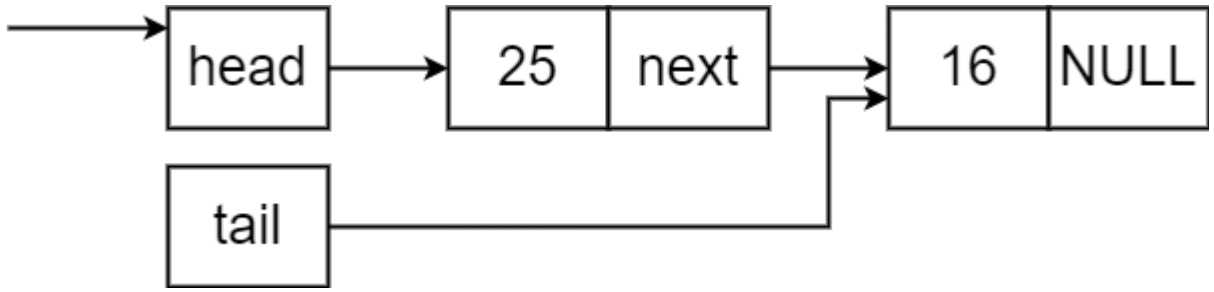
1. Формируется исходное состояние списка
2. Запрашивается память для записи элемента списка
3. Внесение значения в список
4. Занесение NULL в качестве адреса следующего элемента списка

Линейные односвязные структуры

- Использование типа данных ***структура*** становится необходимым при описании элементов списков.

```
typedef struct Item {
    int data;                 // Элемент хранения
    struct Item *next;        // Указатель next ссылается на
                              // следующий элемент списка
} Item;
```

```
typedef struct List {
   Item *head;
   Item *tail;
} List;
```



Операции над списками

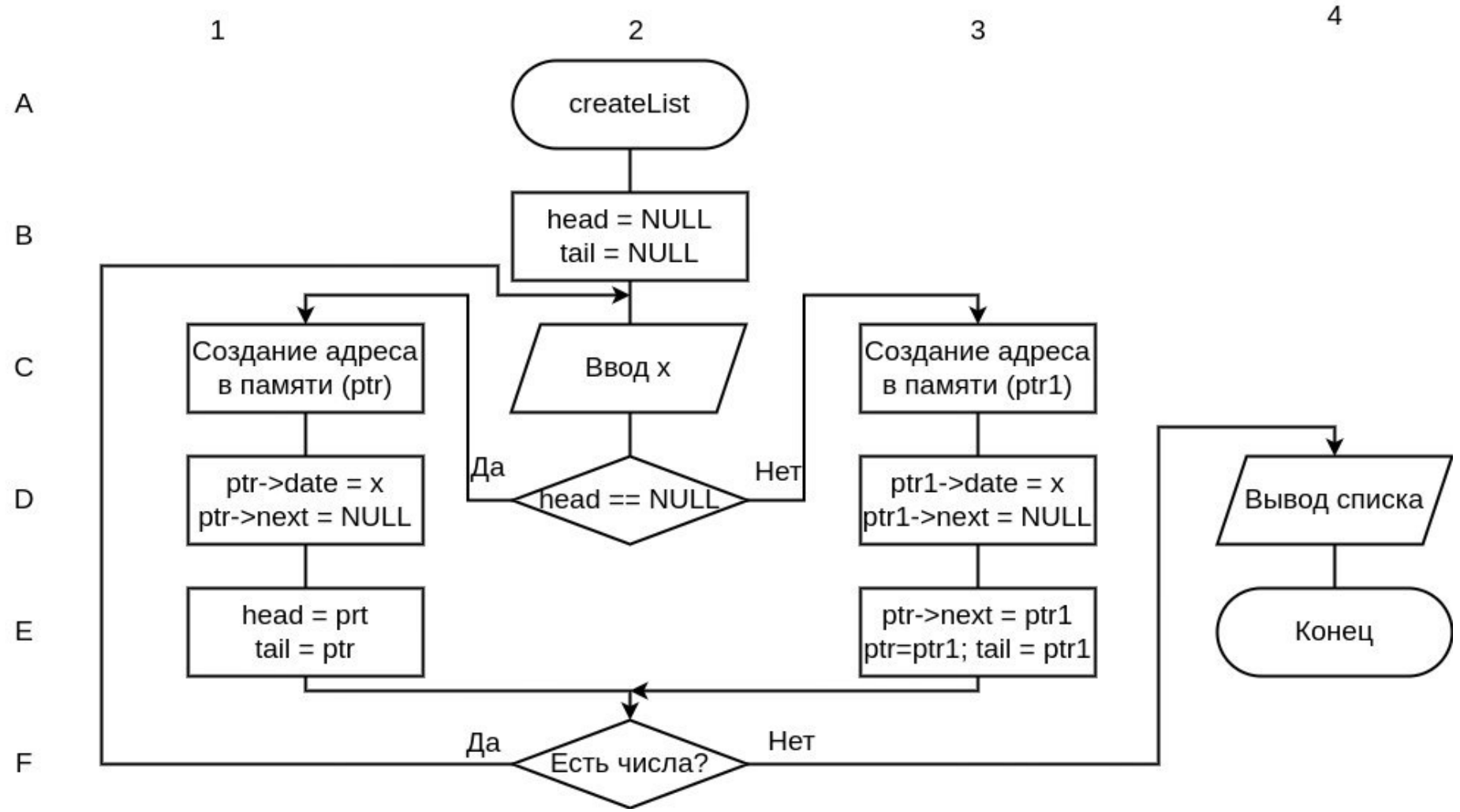
- Создать список
- Удалить элемент из списка
- Добавить элемент в список
- Вставить элемент на заданное место
- Найти в списке требуемое значение
- Сортировать список
- Удалить весь список
- Сохранить данные из списка в файле
- Прочитать данные из файла в список
- Прочитать список и вывести на экран

Алгоритм создания списка

- При связанном описании в качестве элементов хранения используются структуры, связанные по одной из компонент в цепочку, на начало которой (первую структуру) используется указатель List.

```
int list_push_back(List *list, int data) {
    Item *ptr = (Item *) malloc(sizeof(Item)); // Выделить память для эл-та списка
    if (!ptr) return -1; // не удалось выделить память
    ptr->data = data;
    ptr->next = NULL;
    if (!list->head) {
        list->head = ptr; list->tail = ptr;
    } else {
        list->tail->next = ptr;  list->tail = ptr;
    }
    return 0;
}
// Выделение памяти для указателей на список
List *list_new()  { return (List *) calloc(1, sizeof(List)); }
```

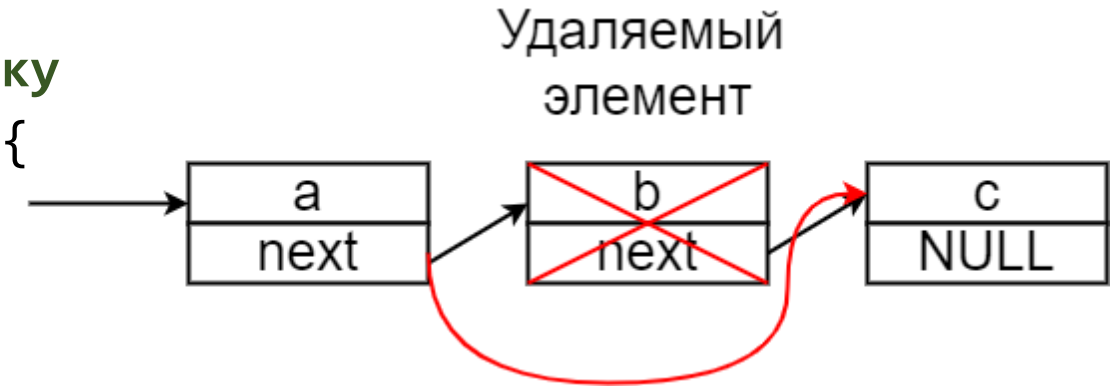
Алгоритм создания списка



Удаление элемента в списке (1)

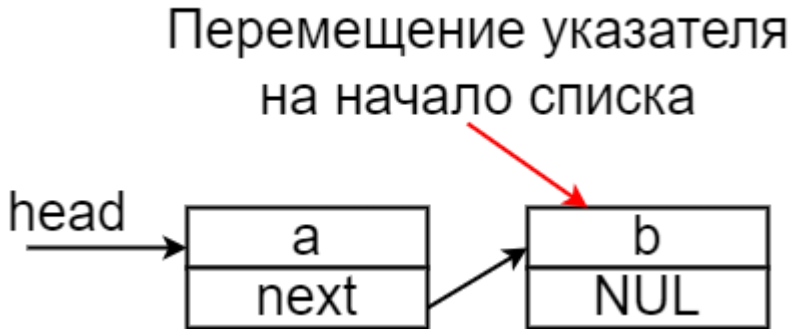
Удаление элемента из середины списка

```
*lp=List->head;
// Это только движение по списку
while (lp && (lp->data != b)) {
    ls = lp;
    lp = lp->next;
}
// Удаление найденного значения
if (lp != NULL) {
    ls->next = lp->next;
    free(lp);
}
// Прочие элементы списка остались на своих местах
```



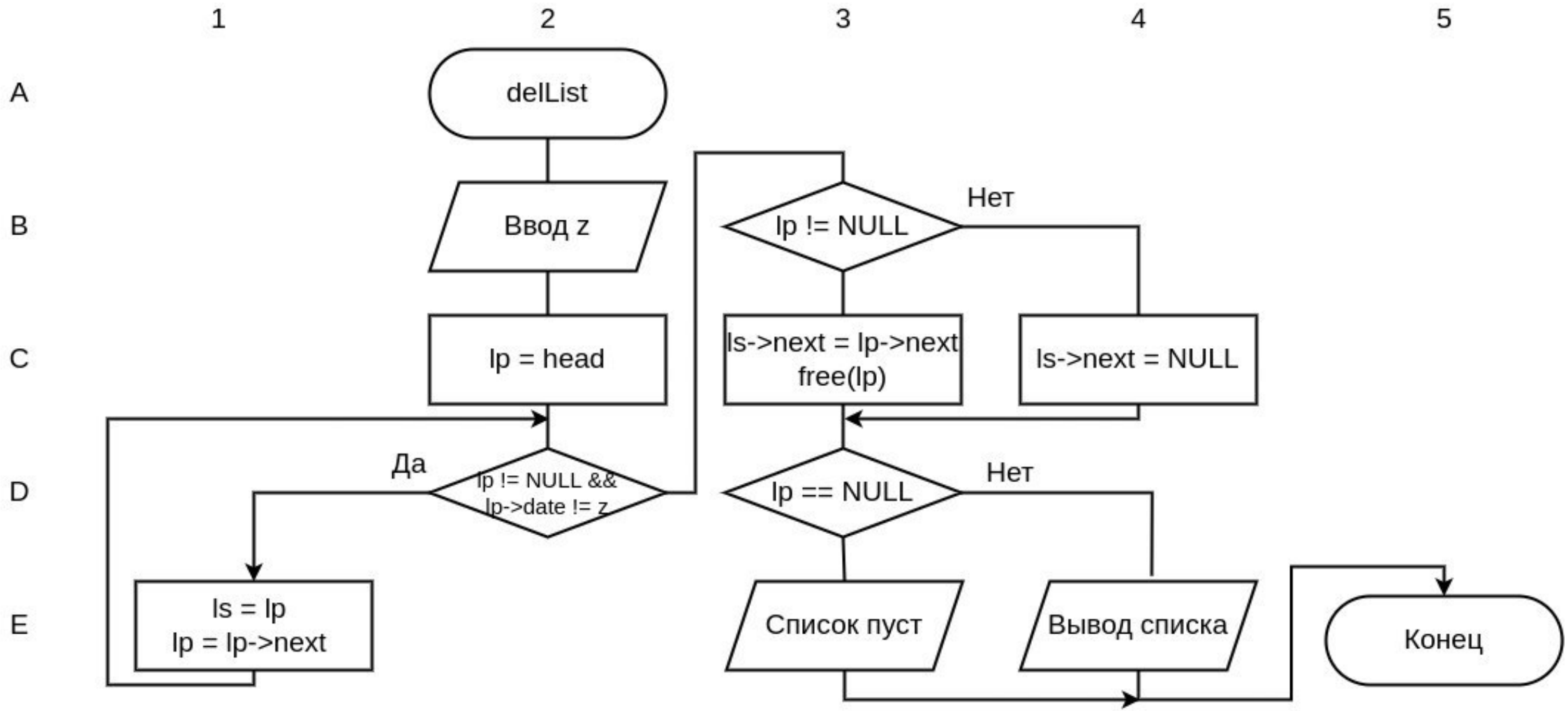
Удаление элемента в списке (2)

Удаление первого элемента




```
*lp=List->head;
if (!lp) { /* список пуст*/ }
else {
  if ((lp == List->head) && (lp->x == b)) {
    List->head = lp->next;
    free(lp);
  }
}
```

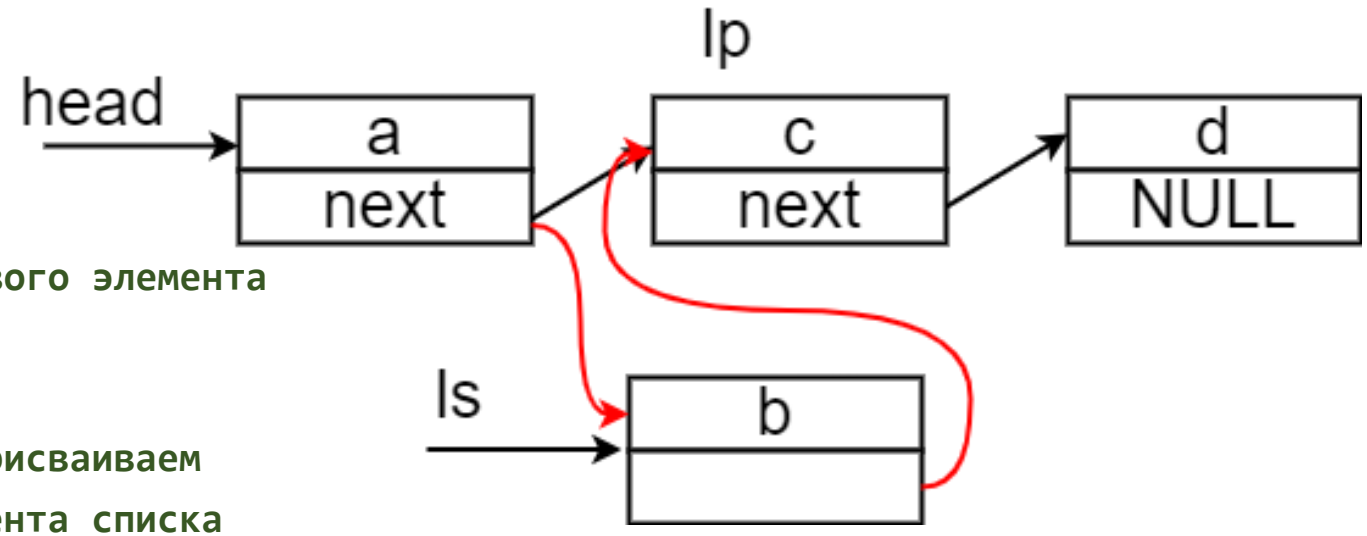
Алгоритм удаления элемента из списка



Добавление элементов в ранее созданный список (пример)

- Задан упорядоченный список. Вставить элемент на правильное место. Вставка должна быть выполнена перед найденным элементом.

```
while (lp != NULL) {
    if (lp -> x < newelem)
        lp = lp -> next;
    else {
        // Выделение памяти  для нового элемента
        ls = malloc(...);
        *ls = *lp;
        // Новому элементу списка присваиваем
        // значение выбранного элемента списка
        lp -> data = newelem;
        lp -> next = ls;  // Это вставляемое значение
    }
}
```



Работа со стеком

- Дисциплина обработки данных: добавление и удаление элементов стека выполняются только через его вершину
- Стек и список можно создавать с помощью рекурсивных функций
- Для вставки элемента на правильное место в стек будет работать алгоритм, рассмотренный для линейных списков


```
typedef struct Item {
    int data;
    struct Item *next;
} Item;

typedef struct Stack {
    Item *head; // Есть только один адрес – вершина стека
} Stack;

Stack *stack_new() {
    return (Stack *) calloc(1, sizeof(Stack));
}
```

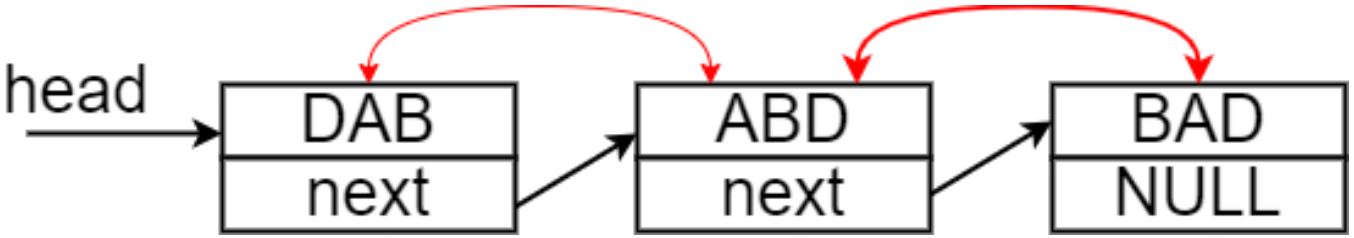
Вставка нового элемента в стек и печать

```
// Вставка в стек
int stack_push(Stack *stack, int data){
    Item *new = (Item *) malloc(sizeof(Item));
    if (!new) {
        return -1; /* если не нашлось памяти*/
    }
    new->data = data;
    new->next = stack->head;
    stack->head = new;
    return 0;
}
```

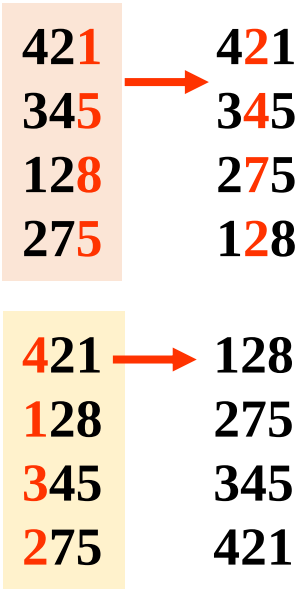
```
// Печать содержимого стека
void stack_print(const Stack *stack) {
    Item *ptr = stack->head;
    while (ptr) {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
}
```

Сортировка списков

- Сортировка в списках ни чем не отличается от сортировки массивов чисел или строк. Все методы сортировки применимы для сортировки списков. Все перемещения выполняются только для информационной части структуры данных



Поразрядная сортировка. Алгоритм:



Выводы

Выводы по работе со списочными структурами данных

1. Алгоритмы сложнее, чем для обработки массивов
2. Дополнительный расход памяти на указатели
3. Размещение списочных структур только в динамической памяти
4. Реализация алгоритмов со списками использует столько памяти, сколько нужно для хранения текущего списка
5. Удаление, вставка, сортировка элементов требуют меньших затрат времени