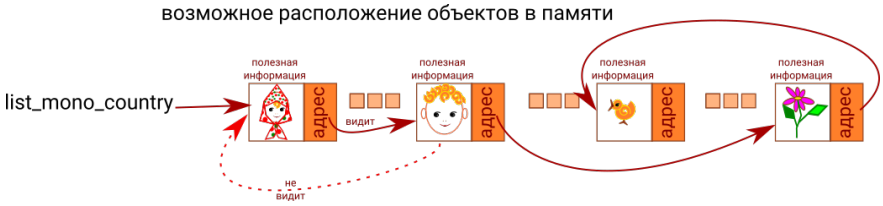
Для хранения данных используются различные структуры данных.

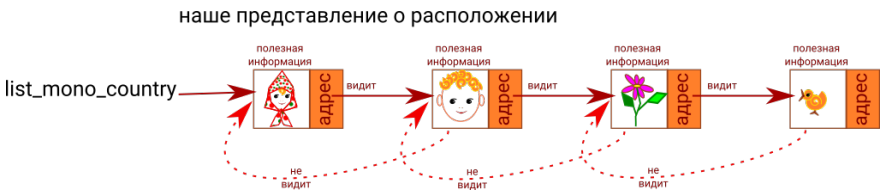
Одной из таких структрур является односвязный список. Односвязный список состоит из элементов, в каждом из которых хранится значение и указатель на следующий элемент списка.

Каждый элемент размещается в области памяти и затем связывается с остальными элементами списка указателями.



То есть элементы могут оказаться далеко не смежных областях памяти.

Однако для работы со списками мы можем себе редставить тот же список так:

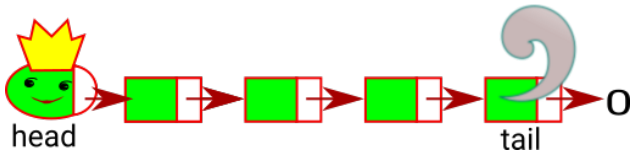


Каждый элемент нашего списка имеет только один указатель на следующий элемент, значит предыдущий элемент не доступен если имеется указатель на текущий элемент.

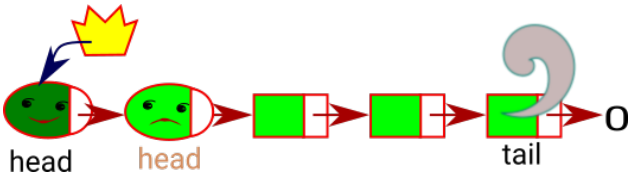
Односвязные списки используются для решения различных задач. Например, для задач, в которых нужен стек, неопределенного размера.

Для удобства работы со списками полезно объявить некоторые структуры и реализовать необходимые функции:

```c
#include <stdio.h>
#include <stdlib.h>
// Описание струткур для работы с односвязным списком:
typedef int Data; // Data может быть любым типом
// один элемент (узел) односвязного списка
typedef struct Nd{
  Data dat; // полезная информация
  struct Nd * next; // указатель на объект типа struct Nd (Node)
}Node;
// Список
// В пустом списке указатели нна голову и хвост равны 0
// в последнем (хвостовом) элементе списка
// указатель next также равен 0
// Это позволит определять состояние списка:
// пуст или нет, и где кончается
typedef struct Lst{
  Node* head; // указатель на начало списка (голова)
  Node *tail; // указатель на последний элемент списка
  int n; // количество элементов
}List;
```

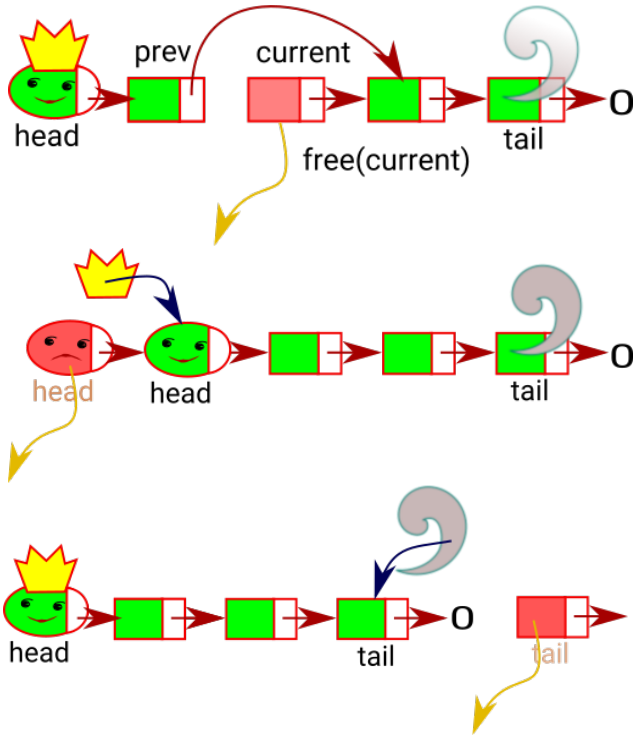



```
// создание списка
List * createList(){
        List *tmp; // временный указатель на  список
// выделяем память под список (указатели на голову, хвост и количество
элементов )
        tmp = (List*)malloc(sizeof(List));
// все указатели обнуляем
        tmp->head = 0;
        tmp->tail = 0;
        tmp->n = 0;
        return tmp;
};
// вставка элемента в голову
void insToHead(List* lst, Data d){
        Node * newObj; // указатель на новый элемент
// выделяем память подд новый элемент
        newObj = (Node* )malloc(sizeof(Node));
// заполняем поля
        newObj->dat = d;
// указатель на следующий элемнт пока 0
        newObj->next = 0;
// если список пуст
        if(!lst->head)
        {
// новый элемнт становится и головой, и хвостом
                lst->head = newObj;
                lst->tail = newObj;
                lst->n = 1;
        }
// если список не пуст
// меняем голову, старую голову:
// цепляем к новому объекту старую голову и
// объявляем новый объект головой
        else
        {
                newObj->next = lst->head;
                lst->head = newObj;
// увеличиваем количество элементов в списке
                lst->n++;
        }

};

// печатаем содержимое списка с головы
void printList(List lst){
        Node *current;
```




```
        current = lst.head;
// если список пуст, то указатель на голову 0
// цикл работоать не будет
        while(current && current->next != lst.head){
// печать хвоста
                if(current == lst.tail){
                        printf("(%p: %d)", current, current->dat);
                }else{
// печать остальных элементов
                        printf("(%p: %d)->", current, current->dat);
                }
// переход к следующему элементу
                current = current->next;
        }
//      printf("(%p: %d)\n", lst.tail, lst.tail->dat);
};
// удаление списка
void destroyList(List *lst){
        int i = 0;
// Будем получать адрес следующего элемента,
// а удалять текущий
        Node *current, *prev;
        current = lst->head;
        prev = current;
// пока не добрались до 0
        while(current /*&& current != lst->tail->next*/){
                current = current->next;
                free(prev);
                i++;
                prev = current;
        }
// в конце удаляем память под сам список
        free(lst);
        lst = 0;
};

Node* findData(List lst, Data d){
        Node *current;
        current = lst.head;
// ищем адрес элемента, где есть d
// пока  не добрались до 0 - барьер
        while(current){
                if(current->dat == d)
                        return prev;
// переход к следующему
```

```
                current = current->next;
        }
// если ничего не нашли, возвращаем 0
        return 0;
}
```



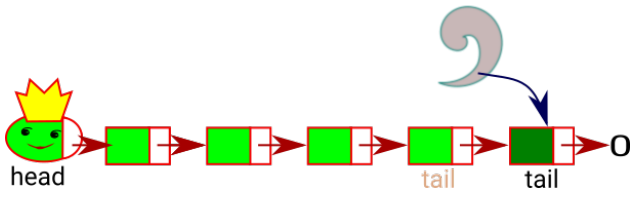

```
List* deleteData(List* lst, Node* p){
        Node *current, *prev;
        current = lst->head;
        prev = current;
// если удалаяем голову, то головой становится
// следующий за ней элемент
        if( p == lst->head){
                lst->head = current->next;
                free (p);
                return lst;
        }
// необходимо помнить указатель на предыдыущий элемент
// чтобы сообщить указателю next предыдущего элемента
// адрес следующего за удаленным
        while(current && current->next != lst->head){
                if(current == p){
                        if(p == lst->tail){
// если удалаяем хвост, то предыдущий элемент становится хвостом
                                lst->tail = prev;
                                lst->tail->next = 0;
                        }
// если удаляем рядовой элемент, то next предыдущего элемента
// получает указатель на следующий за current
```

```
                        prev->next = current->next;
                        free(p);
                        return lst;
                }
                prev = current;
                current = current->next;
        }
};

int main(){
        List *country;
        country = createList();
        Data d;
        int i, n;
        scanf("%d", &n);
// вставка n элементов в голову
        for ( i = 0; i < n; i++){
                scanf("%d",&d);
                insToHead(country, d);
        }
// печать списка
        printList(*country);
        scanf("%d", &d);
// найти первый элемент в списке, равный d
        Node *fnd = findData(*country, d);
// если элемент найден (не 0), то
// удалить его
        if(fnd){
                printf("find:(%p:␣%d)", fnd, fnd->dat);
                country = deleteData(country, fnd);
                printList(*country);
        }else{
                printf("not␣found\n");
        }
// печать списка
        printList(*country);
// список больше не нужен, удаляем
        destroyList(country);
        return 0;
}
```

Задача .1. Вставка в хвост.

Реализовать функцию вставки элемента в конец списка:

```
void insToTail(List* lst, Data d);
```



Задача .2. Удаление элементов Написать функцию, которая удаляет из списка все элементы, равные d

```
void removeData(List* lst, Data d);
```

Задача .3. Вставка элемента Написать функцию

```
void insertDataAfter(List* lst, Data find, Data ins);
```

, которая вставляет элемент ins после элемента find

Задача .4. Вставка элемента Написать функцию

```
void insertDataBefore(List* lst, Data find, Data ins);
```

, которая вставляет элемент ins перед элементом find

Задача .5. Сортировка Написать функцию

```
void sortIns(List* lst, FILE *f);
```

, которая считывает элементы из файла и вставляет из в список, сортируя.