Структура данных ОЧЕРЕДЬ

Очередь – линейный список, в котором извлечение данных происходит из начала, а добавление – в конец списка.

Очередь организована по принципу ***FIFO*** (*First In, First Out*) – первым вошел, первым выйдет.

Работа с очередью реализуется при помощи динамических структур, для которых необходимо выделение и освобождение памяти.

Простой пример – очередь в кассу, если очереди нет, обслуживаешься сразу, иначе, становишься в ее конец.

Последовательно обслуживаются стоящие в начале очереди.

В течение дня очередь то увеличивается, то уменьшается и может отсутствовать.

Очереди организуются в виде ***односвязных*** или ***двухсвязных*** списков, в зависимости от количества связей (указателей) в адресной части элемента структуры.

Односвязный список (очередь)

Шаблон структуры, информационная часть (ИЧ) которого – целое число:

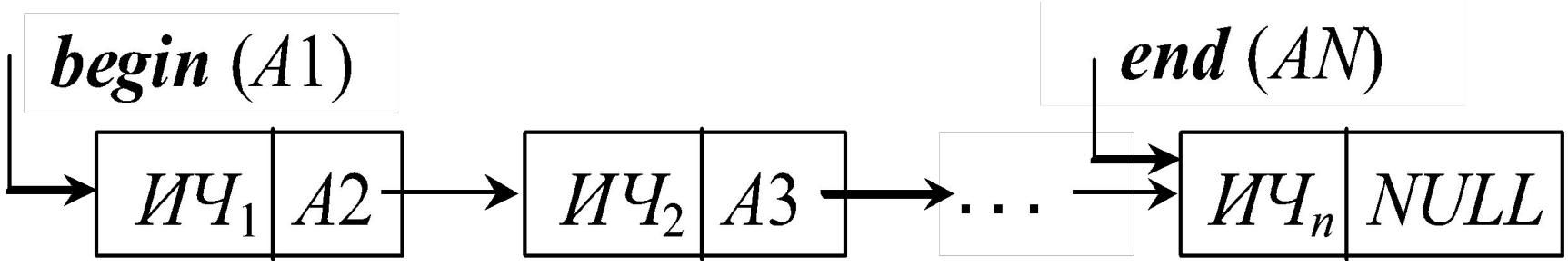
```
struct Spis1 {          // Или TList1
    int  info;
    Spis1  *next;
};
```

При организации очереди обычно используют два указателя

```
    Spis1  *begin, *end;
```

begin и ***end*** – указатели на начало и конец.

При создании очереди организуется структура следующего вида:



Каждый элемент очереди имеет информационную ***info*** ($ИЧ_1$, ..., $ИЧ_n$) и адресную ***next*** (*A*2, *A*3, ..., *AN*) части.

Адресная часть последнего элемента равна ***NULL***.

Основные операции с очередью:

– формирование очереди;
– добавление нового элемента в конец очереди;
– удаление элемента из начала очереди;
– обработка элементов (просмотр, поиск и др.);
– освобождение памяти, занятой очередью.

Формирование очереди состоит из двух этапов: создание первого элемента, добавление нового элемента в конец.

Создание первого элемента

1. Ввод информации для первого элемента (например, целое число ***i***);
2. Захват памяти, используя текущий указатель:

```
t = new Spis1;
```

формируется конкретный адрес (***A1***) для первого элемента;

3. Формирование информационной части:

t -> info = i; (обозначим ***i*1**)

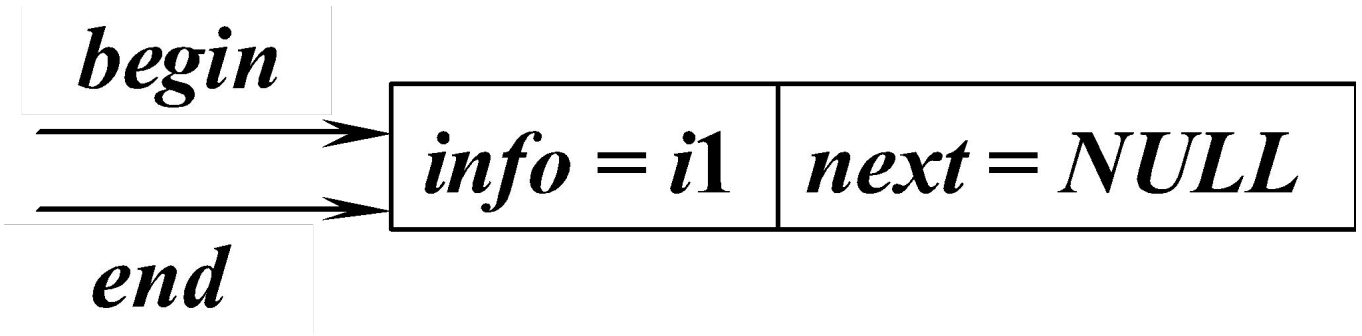
4. В адресную часть записываем ***NULL***:

t -> next = NULL;

5. Указатели начала и конца очереди устанавливаем на созданный элемент ***t*** :

begin = end = t;

На этом этапе получим следующее:



Добавление элемента в очередь

Рассмотрим добавление только для второго элемента.

1. Ввод информации для текущего (второго) элемента – значение ***i*** .
2. Захват памяти под текущий элемент:

   t = new Spis1; (адрес ***A*2**)
3. Формирование информационной части (***i*2**):

   t -> info = i;
4. В адресную часть заносим ***NULL***, т.к. этот элемент становится последним:

   t -> next = NULL;

5. Элемент добавляется в конец, поэтому в адресную часть бывшего последнего элемента ***end*** заносим адрес созданного:

end -> next = t;

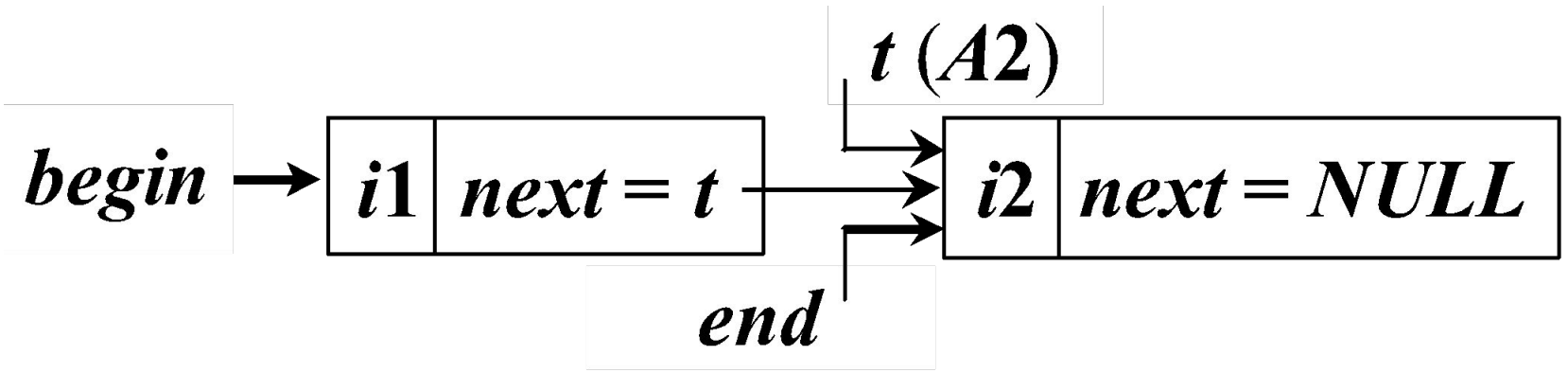
бывший последний элемент становится пред-последним.

6. Переставляем указатель ***end*** последнего элемента на добавленный:

end = t;

Обратите внимание, что пункты 1 – 4 обоих этапов идентичны.

В результате получим



Для добавления в очередь любого количества элементов организуется цикл, включающий пункты 1 – 6 рассмотренного алгоритма.

Завершение цикла реализуется в зависимости от поставленной задачи.

Обобщив оба этапа, функция формирования очереди может иметь следующий вид (***b*** – начало, ***e*** – конец, ***in*** – введенная ранее информация):

```
void Create (Spis1 **b, Spis1 **e, int in)  {
    Spis1 *t = new Spis1;     // Захват памяти
    t -> info = in;           // Формирование ИЧ
    t -> next = NULL;         // Формирование АЧ
// Если указатель на начало равен NULL, то
// формируем первый элемент
    if ( *b == NULL )
            *b = *e = t;
```

```
// Иначе добавляем элемент в конец
    else {                                  (*e) -> next = t;

        *e = t;
    }
}
```

В функцию передаются адреса указателей, чтобы при изменении обеспечить их возврат в точку вызова.

Обращение к данной функции

Create (&begin, &end, in);

Эту функцию можно реализовать иначе:

```
Spis1* Create (Spis1 **b,  Spis1 *e,  int in)  {
    Spis1 *t = new Spis1;
    t -> info = in;
    t -> next = NULL;
    if ( *b == NULL )
        *b = e = t;
    else {                                  e -> next = t;

        e = t;
    }
    return e;
}
```

В функцию передаются:

адрес указателя на начало списка, чтобы при его изменении обеспечить возврат в точку вызова;

значение указателя на конец списка, измененное значение которого возвращается в точку вызова оператором ***return e*** ;

значение ранее введенной ИЧ ***in***.

Обращение к функции в этом случае :

end = ***Create*** (&begin, end, in);

Удаление первого элемента из очереди аналогично извлечению элемента из Стека…

Пусть очередь создана (***begin*** не равен ***NULL***, рекомендуется организовать эту проверку).

1. Устанавливаем текущий указатель ***t*** на начало:

t = begin;

2. Обрабатываем ИЧ первого элемента (например, выводим на экран).

3. Указатель начала переставляем на следую-щий (2-й) элемент

begin = begin -> next;

4. Освобождаем память, занятую бывшим первым:

delete t;

Алгоритмы просмотра и освобождения памяти аналогичны рассмотренным ранее для Стека.

В функциях просмотра ***View*** и освобождения памяти ***Del_All***, реализующих эти алгоритмы, необходимо только изменить типы данных (вместо *Stack* пишем *Spis*1).

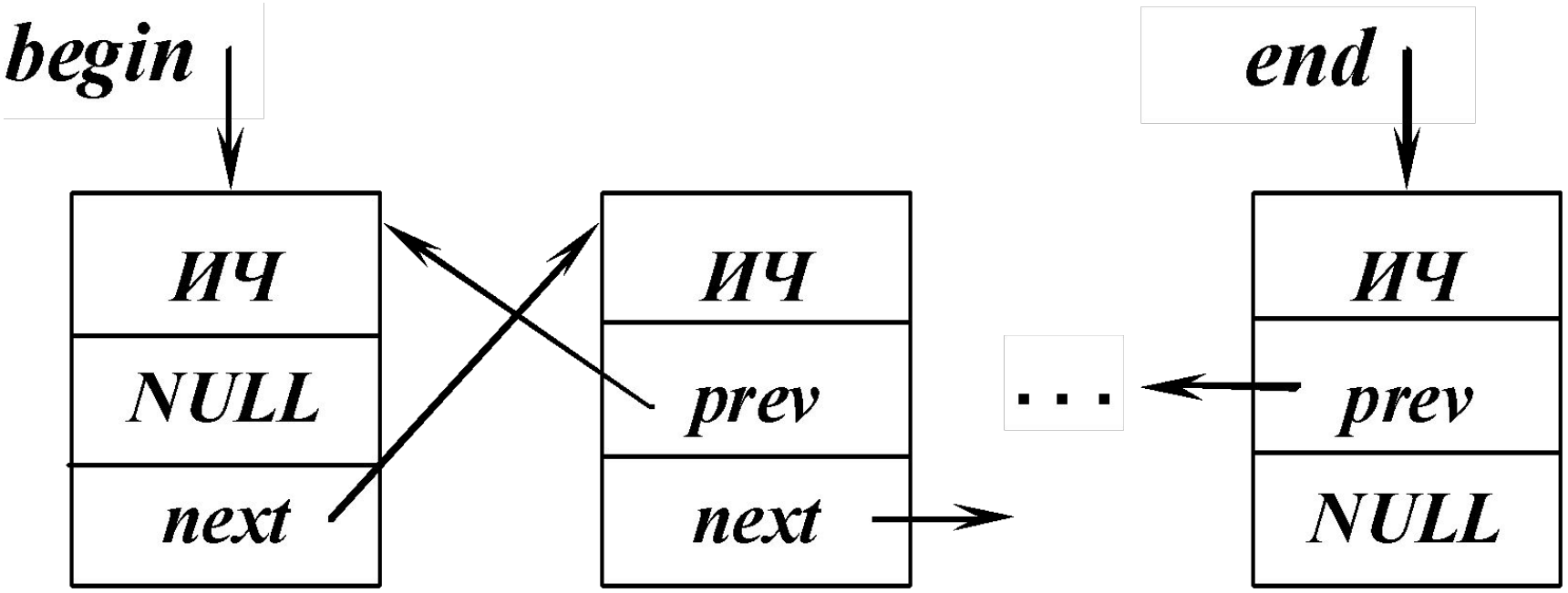
Очередь может быть организована и в виде ***двухсвязного (двунаправленного) списка***, в каждом элементе которого два указателя: на предыдущий (***prev***) и следующий (***next***).

Шаблон такой структуры будет иметь вид:

```
struct Spis2    {          // Или TList2
    Информационная часть
    Spis2  *prev, *next;    – Адресная часть
} ;
```

Для работы с такими списками обычно используются указатели на начало ***begin*** и на конец ***end*** списка.

Графически такой список выглядит следующим образом:



Адреса ***begin -> prev*** (предыдущий у первого) и ***end -> next*** (следующий за последним) равны ***NULL***.

Формирование двунаправленного списка

проводится в два этапа – формирование первого элемента и добавление нового, причем добавление может выполняться как в начало, так и в конец списка.

Введем структуру, информационной частью ***info*** которой будут целые числа:

```
struct Spis2    {          // Или TList2
    int info;
    Spis2  *prev, *next;
} ;
```

Создание первого элемента

1. Захват памяти: ***t = new Spis2***;

   формируется конкретный адрес в ОП.

2. Ввод переменной ***i*** и формирование ИЧ:

   t -> info = i;

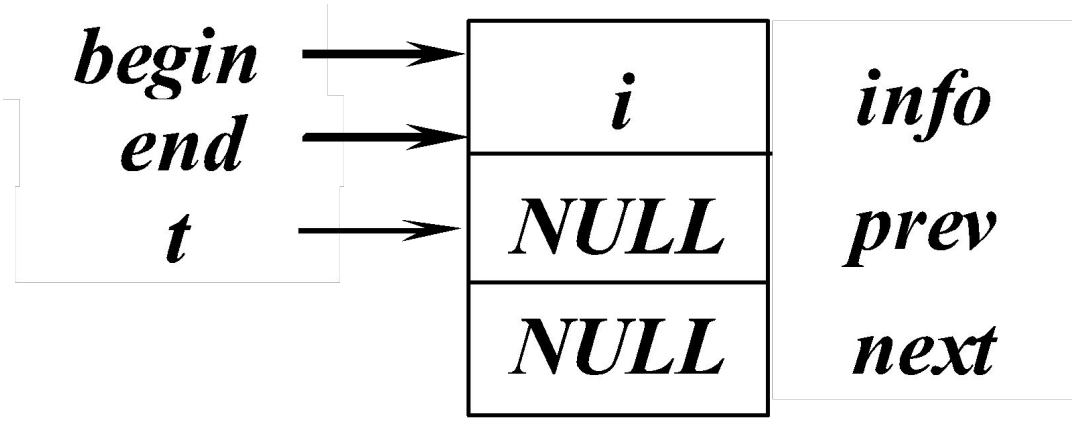
3. Формирование адресных частей (для первого элемента – это ***NULL***):

   t -> next = t -> prev = NULL;

4. Установка указателей начала и конца списка на созданный ***первый*** элемент:

   begin = end = t;

Получили первый элемент списка:



Добавление элемента

Захват памяти и формирование ИЧ аналогичны рассмотренным пунктам 1 – 2.

Добавление в начало списка

3.1. Формирование адресных частей:

а) предыдущего нет: ***t -> prev = NULL***;

б) связываем новый элемент с первым

t -> next = begin;

4.1. Изменяем адреса:

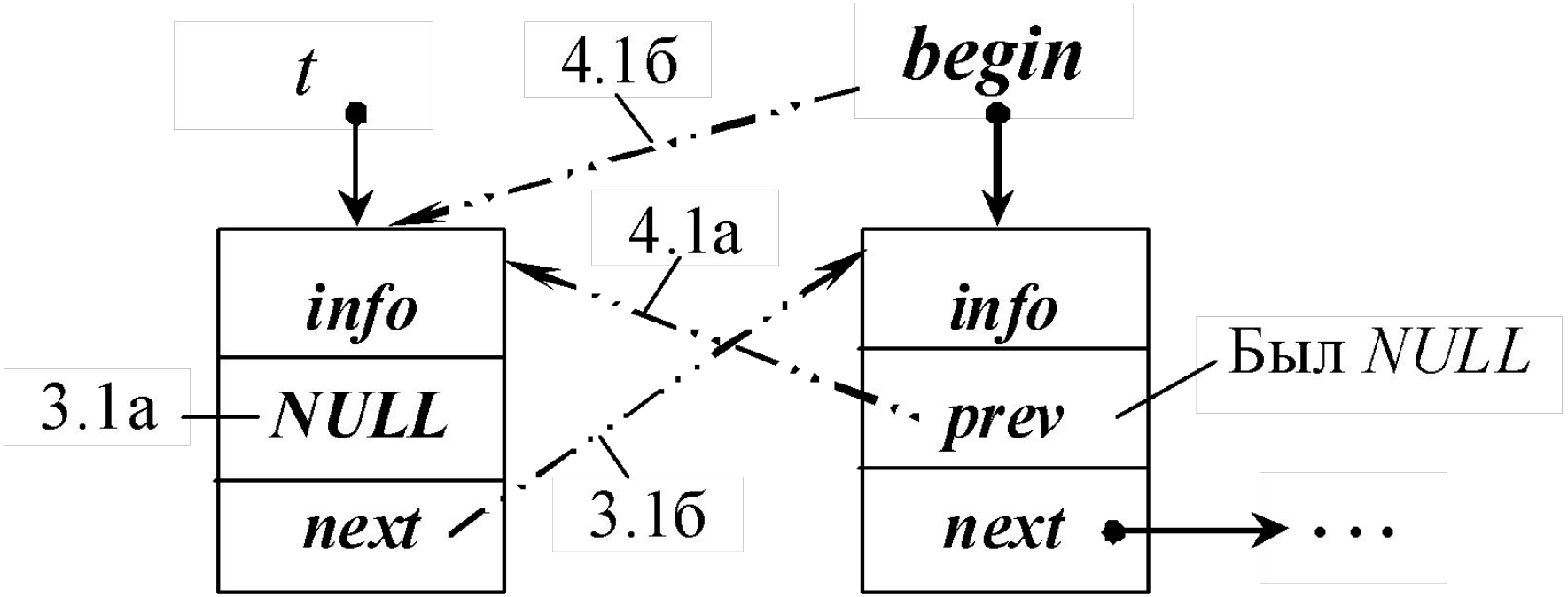
а) изменяем адрес ***prev*** бывшего первого

begin -> prev = t;

б) переставляем указатель ***begin*** на новый

begin = t;

Схема добавления элемента в начало



Добавление в конец списка

3.2. Формирование адресных частей:

а) следующего нет: ***t -> next = NULL***;

б) связываем новый элемент с последним

t -> prev = end;

4.2. Изменяем адреса:

а) изменяем адрес ***next*** бывшего последнего

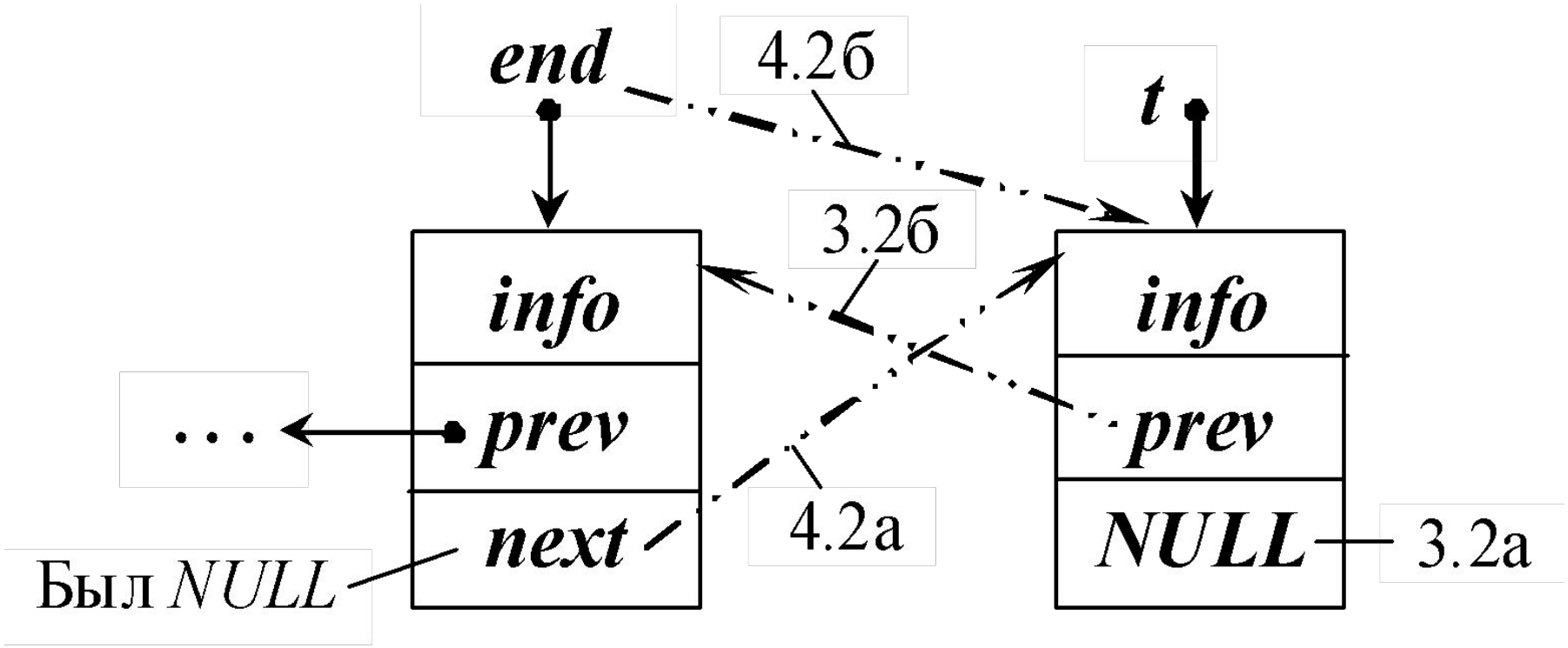
end -> next = t;

б) переставляем указатель ***end*** на новый

end = t;

Внимание, включив пункты 1 – 4 в цикл, можно добавить нужное количество элементов.

Схема добавления элемента в конец



Алгоритм просмотра списка

С начала	***С конца***
1. Текущий указатель устанавливаем на:	
начало списка ***t = begin***;	конец списка ***t = end***;
2. Начало цикла, работающего пока ***t ≠ NULL***	
3. ИЧ текущего элемента ***t -> info*** – на экран.	
4. Текущий указатель переставляем на:	
следующий элемент ***t = t -> next***;	предыдущий элемент ***t = t -> prev***;
5. Конец цикла.	

Алгоритм поиска по ключу

Ключом может быть любое поле структуры, обычно это значение должно быть уникальным (не повторяться). В нашем случае найдем в списке элемент, информационная часть (***ключ***) которого совпадает с введенным с клавиатуры значением (***i_key***).

1. Введем дополнительный указатель:

   Spis2 *key = NULL;

2. Введем значение искомого ключа ***i_key***.

3. Установим текущий указатель на начало:

   t = begin;

4. Начало цикла (выполняем пока ***t* != *NULL***).

5. Сравниваем ИЧ текущего элемента ***t*** с ***i_key***.

5.1. Если они совпадают (***t -> info* = *i_key***):

а) запоминаем адрес найденного элемента:

key* = *t; (выводим ***key -> info*** на экран)

б) т.к. ключи уникальны, завершаем поиск (выход из цикла ***break***).

5.2. Иначе, переставляем текущий указатель ***t***:

t* = *t -> next;

6. Конец цикла.

7. Если ***key* = *NULL***, т.е. искомый элемент не найден, то выводим сообщение о неудаче.

Алгоритм удаления ОДНОГО элемента по ключу

Удалить из списка элемент, *ИЧ* (ключ) которого совпадает с введенным значением.

Решение задачи проводим в два этапа – поиск и удаление.

Первый этап «Поиск» рассмотрен ранее.

Второй этап «Удаление» выполняем, если элемент для удаления найден (***key*** ≠ ***NULL***).

Удаление выполняем в зависимости от расположения элемента с адресом ***key*** в списке.

Удаление

1. Если удаляемый элемент находится в ***начале*** списка, т.е. ***key*** = ***begin***, то первым элементом списка должен стать второй:

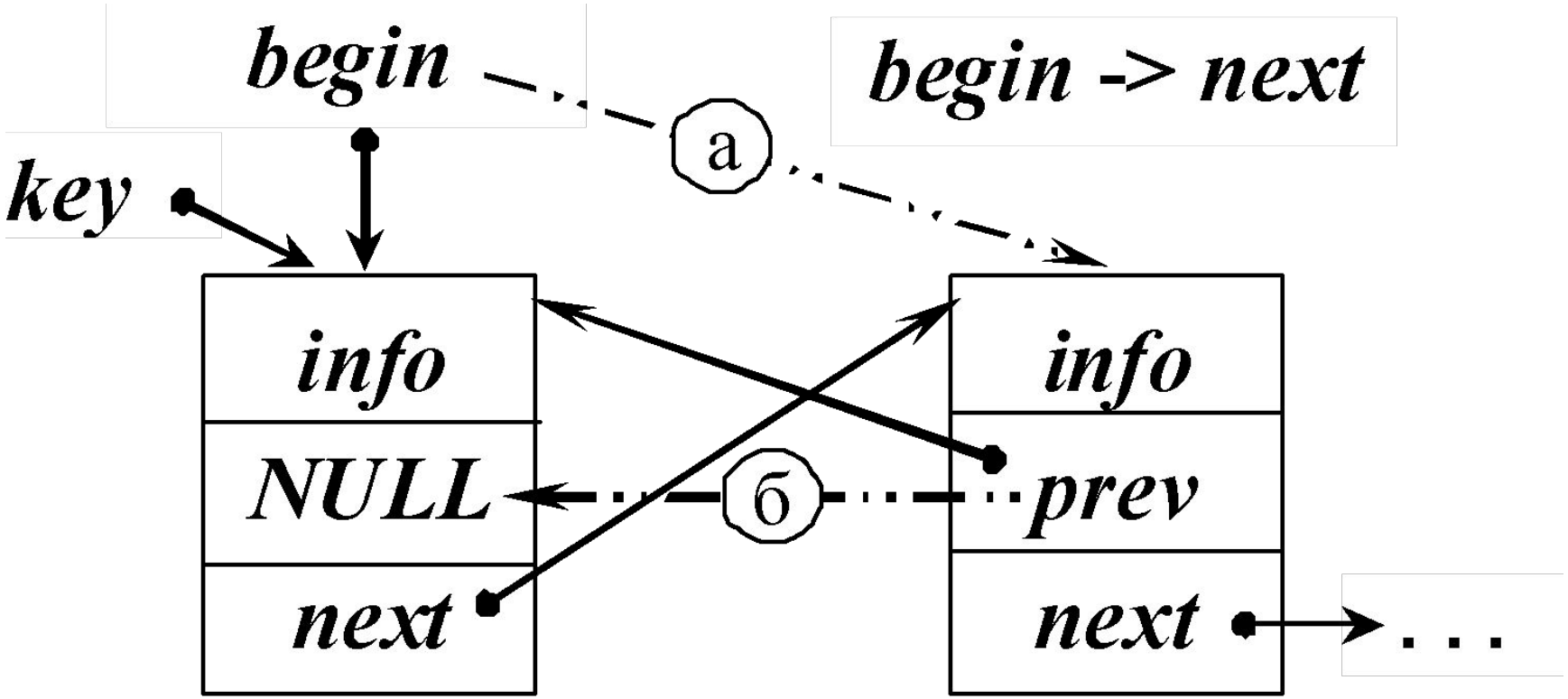
   а) указатель на начало переставляем на следующий (второй) элемент:

   begin = begin -> next;

   б) адрес ***prev*** элемента, который был вторым, а теперь становится первым в ***NULL***, т.е. предыдущего нет, причем исключаем ситуацию, если ***begin*** остался один, т.е. если ***begin != NULL***

   begin -> prev = NULL;

Схема удаления элемента ***key*** из начала списка:



2. ***Иначе***, если удаляемый элемент в ***конце*** списка (***key*** = ***end***), то последним элементом в списке должен стать предпоследний:

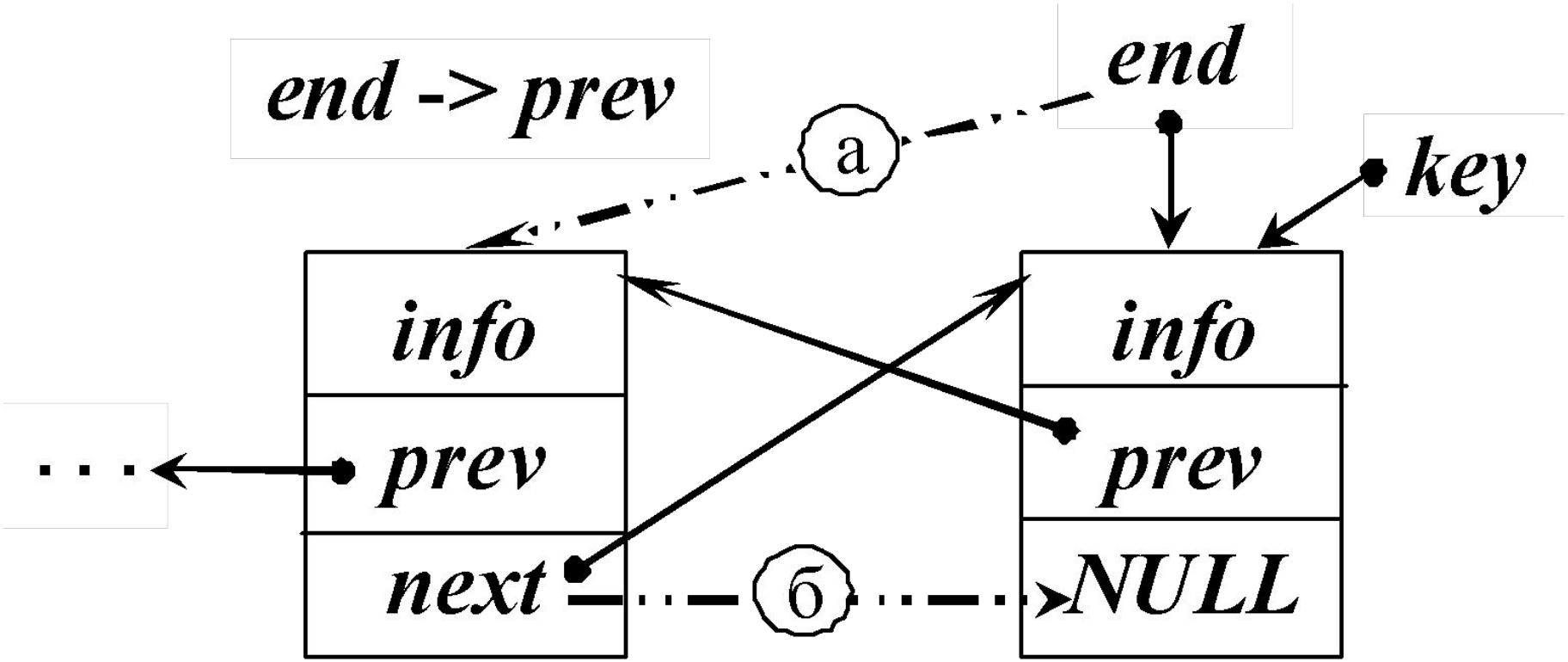
а) указатель конца списка переставляем на предыдущий элемент, адрес которого в поле ***prev*** последнего ***end*** элемента:

end = end -> prev;

б) обнуляем адрес ***next*** нового последнего элемента

end -> next = NULL;

Схема удаления элемента ***key*** из конца списка:



3. Иначе, если элемент ***key*** находится ***в середине*** списка, нужно обеспечить связь предыдущего ***key -> prev*** и следующего ***key->next*** элементов:

а) адрес ***next*** предыдущего элемента ***key -> prev*** переставим на следующий элемент ***key -> next***:

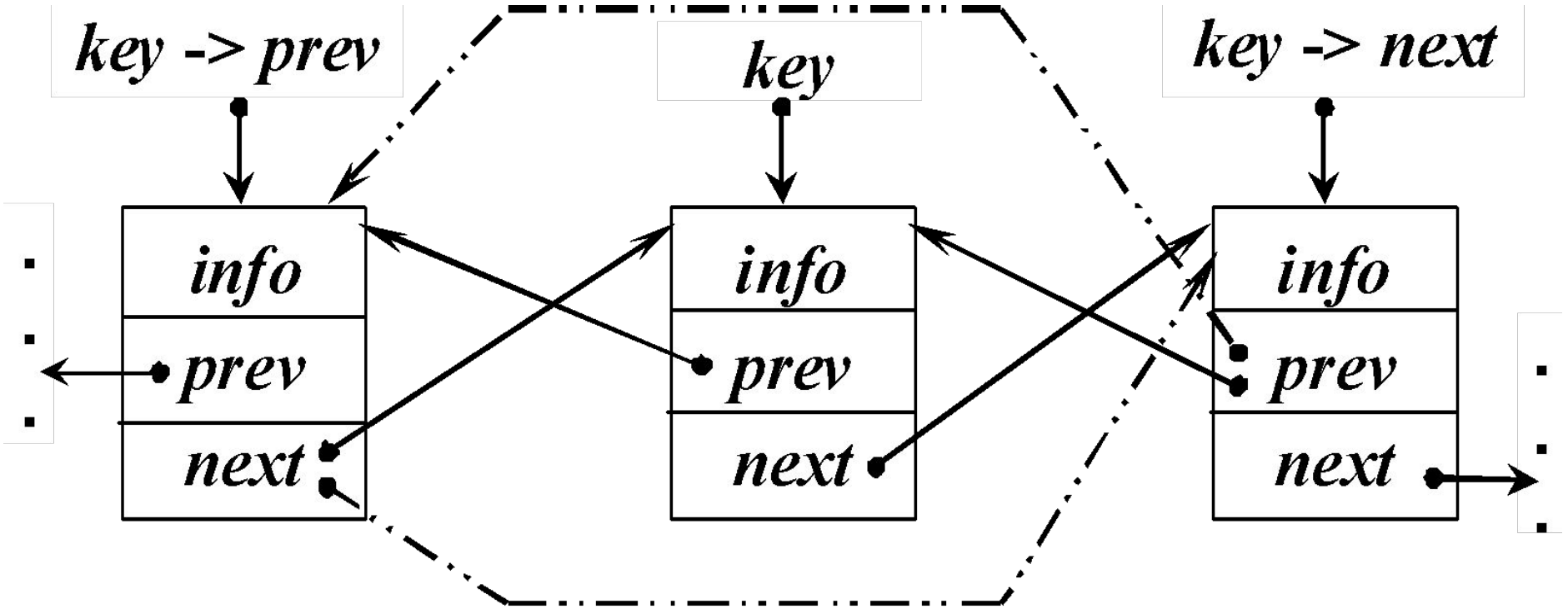
(*key -> prev*) -> *next* = *key -> next*;

б) и наоборот, адрес ***prev*** следующего элемента ***key -> next*** переставим на предыдущий ***key -> prev***:

(*key -> next*) -> *prev* = *key -> prev*;

4. Освобождаем память, занятую удаленным элементом ***delete key***;

Схема удаления ***key*** из середины списка:



Алгоритм вставки элемента после элемента с указанным ключом

Вставить в список элемент после элемента, значение *ИЧ* (***ключ***) которого совпадает с введенным.

Решение данной задачи проводится в два этапа – поиск и вставка.

Поиск аналогичен рассмотренному в алгоритме удаления.

Вставку выполняем, если искомый элемент найден, т.е. указатель ***key*** не равен ***NULL***.

Этап второй – вставка

Найден адрес элемента ***key***, после которого вставляем новый.

1. Захватываем память под новый элемент

   t = new Spis2;

2. Формируем ИЧ (***t -> info***).

3. Связываем новый элемент с предыдущим

   t -> prev = key;

4. Связываем новый элемент со следующим

   t -> next = key -> next;

если *key = end*, то *t -> next = key -> next = NULL*.

5. Связываем предыдущий элемент с новым

key -> next = t;

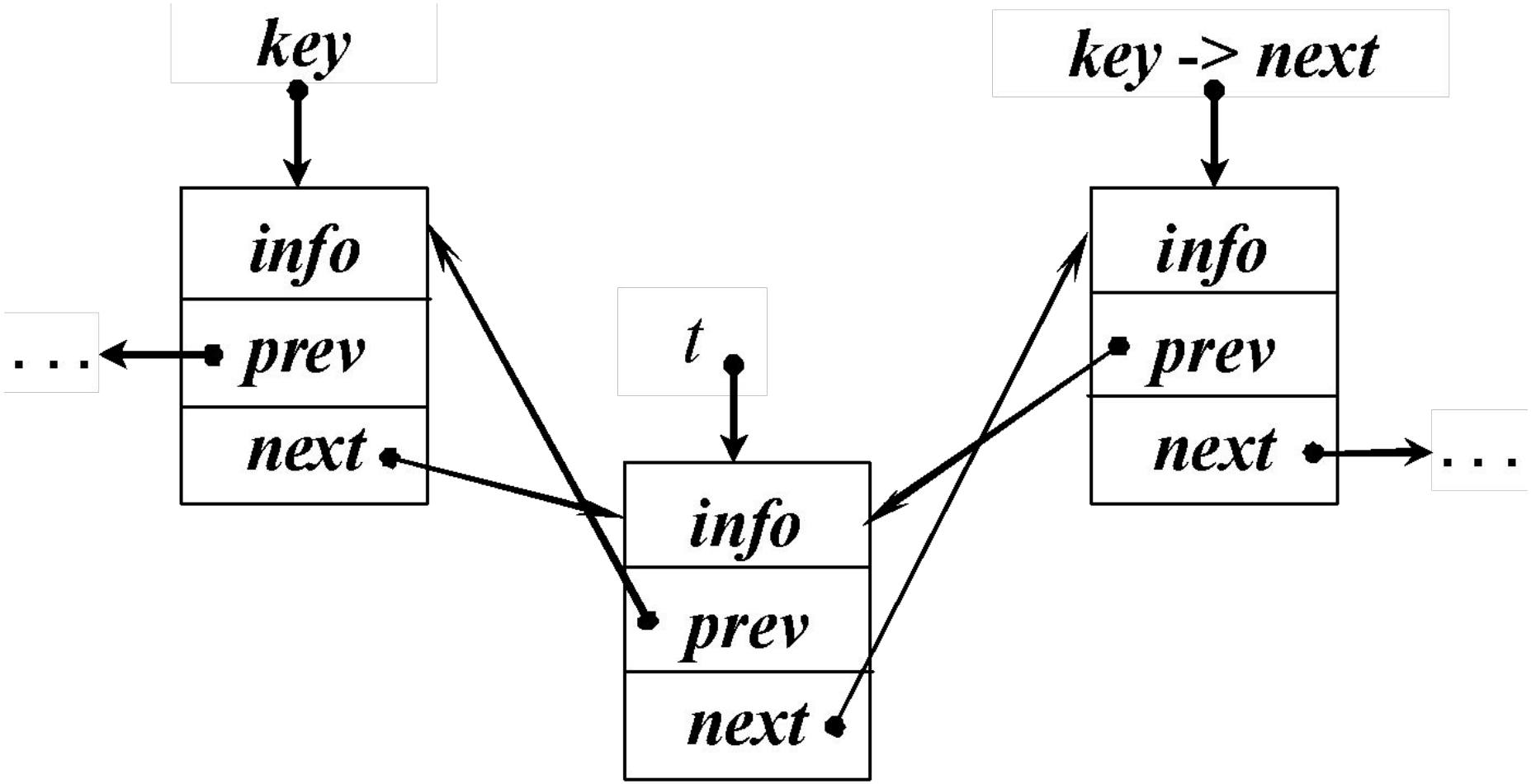
6. Если элемент добавляется не в конец списка (как показано на рисунке), т.е. ***key*** не равен ***end***, то

(t -> next) -> prev = t;

7. Иначе (***key*** = ***end***), то указатель ***key -> next*** равен ***NULL*** (см п. 4) и новым последним становится ***t*** :

end = t;

Общая схема вставки элемента:



Алгоритм освобождения памяти, занятой списком, аналогичен рассмотренному ранее алгоритму для стека.

Только в функции ***Del_All*** необходимо изменить типы данных.