# Оглавление

# Односвязный список

Односвязный список является динамической структурой данных, реализующей формирование и обработку набора элементов. Элементы односвязного списка могут быть отсортированы или не отсортированы.

Односвязный список может сохраняться:

- в оперативной памяти;
- в файле.

В односвязном списке выделяют отдельный элемент (Element), который называется узел (Node).

Каждый элемент (узел) односвязного списка состоит из двух частей:

- данные. Это может быть переменная любого примитивного типа (int, double, char, …), объект класса или сложная структура. Данные могут состоять из нескольких полей (переменных);
- адрес или позиция следующего элемента. Если односвязный список хранится в оперативной памяти, то это указатель (адрес) на такой же элемент (узел). Если односвязный список хранится в файле, то это позиция следующего элемента в файле.

Реализовать узел можно с помощью класса или структуры. Также можно создавать односвязный список на основе шаблонного класса, оперирующего некоторым обобщенным типом **T**.

Совокупность элементов (узлов), в которых указатель предыдущего элемента указывает следующий элемент, образует односвязный список.

Для списка можно выделить следующие основные операции:

- формирование списка;
- добавление элемента в конец списка;
- вставка элемента в заданную позицию списка;
- удаление элемента из списка из заданной позиции;
- очистка списка;
- замена элемента в списке;
- поиск элемента в списке по индексу или некоторому критерию.

При желании перечень операций можно расширить, добавив собственные идеи (например, вставка группы элементов в список, сортировка элементов в списке, создание отсортированного списка и т.п.).

## Общий вид списка. Структура, описывающая один элемент списка

Общий вид линейного односвязного списка изображен на рисунке 1.

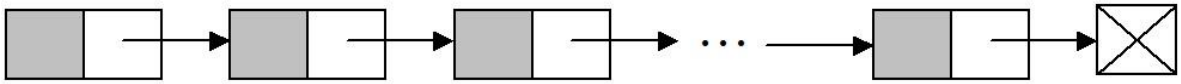

Рисунок 1. Общий вид списка

На рисунке 2 изображена структура Element, используемая как базовый элемент линейного односвязного списка. В каждом элементе хранятся данные некоторого типа **T**.

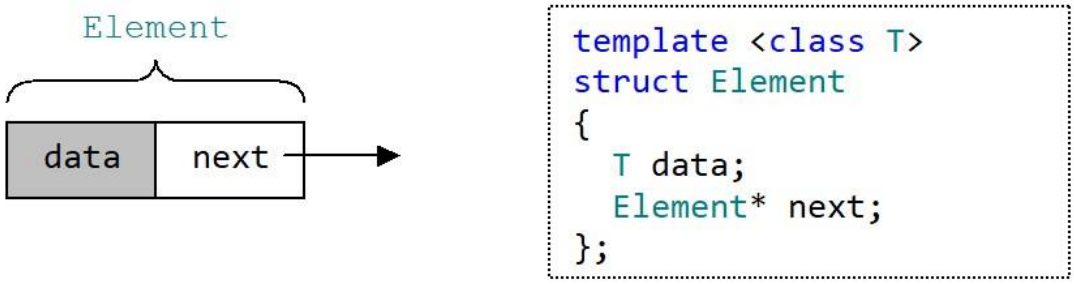


Рисунок 2. Структура Element, описывающая один элемент списка с программным кодом на языке C++

## Формирование списка

### Создать пустой список

Создание списка включает следующие этапы.

1. Объявление указателей на начало и конец списка.
2. Установка нулевых значений указателям.
3. Установка количества элементов в значение 0.

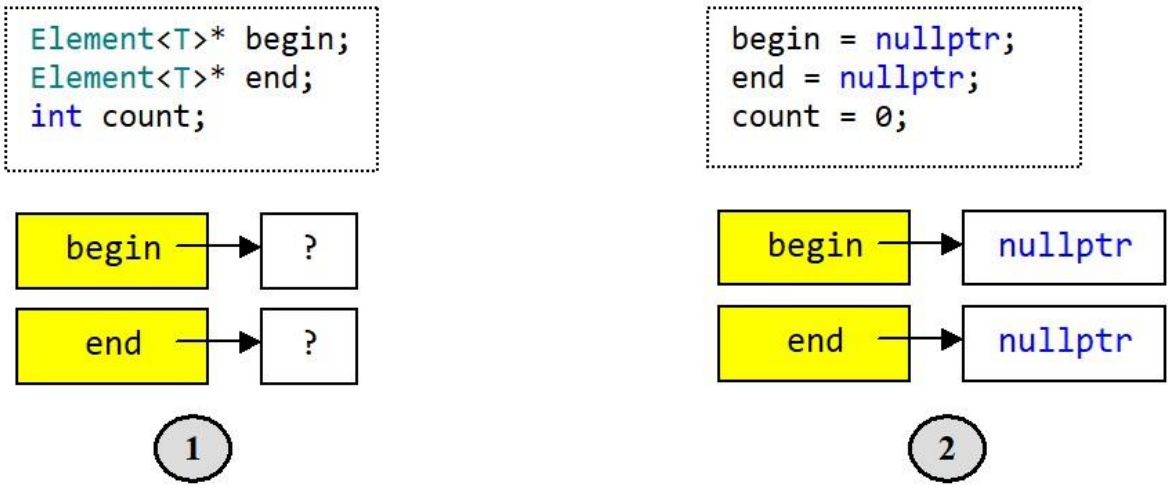


Рисунок 3. Формирование списка: 1) — объявление указателей; 2) — установка нулевых значений

**Добавление элемента в конец списка**

При добавлении элемента в конец списка рассматривают 2 возможные ситуации:

- добавление элемента в пустой список;
- добавление элемента в непустой список (список, в котором есть элементы).

**Добавление нового элемента в конец пустого списка**

При добавлении элемента в конец пустого списка выделяют следующие операции (рисунок 4):

1. Создать новый элемент. Здесь нужно выделить память для нового элемента и заполнить ее некоторыми данными (_data). Для этого объявляется указатель (например, elem):
   Element<T>* elem = new Element<T>;
   elem->data = _data;
2. Установить указатель next в нулевое значение
   elem->next = nullptr;
3. Установить указатели begin и end равными значению указателя elem.
   begin = end = elem;
4. Увеличить количество элементов в списке на 1
   count++;

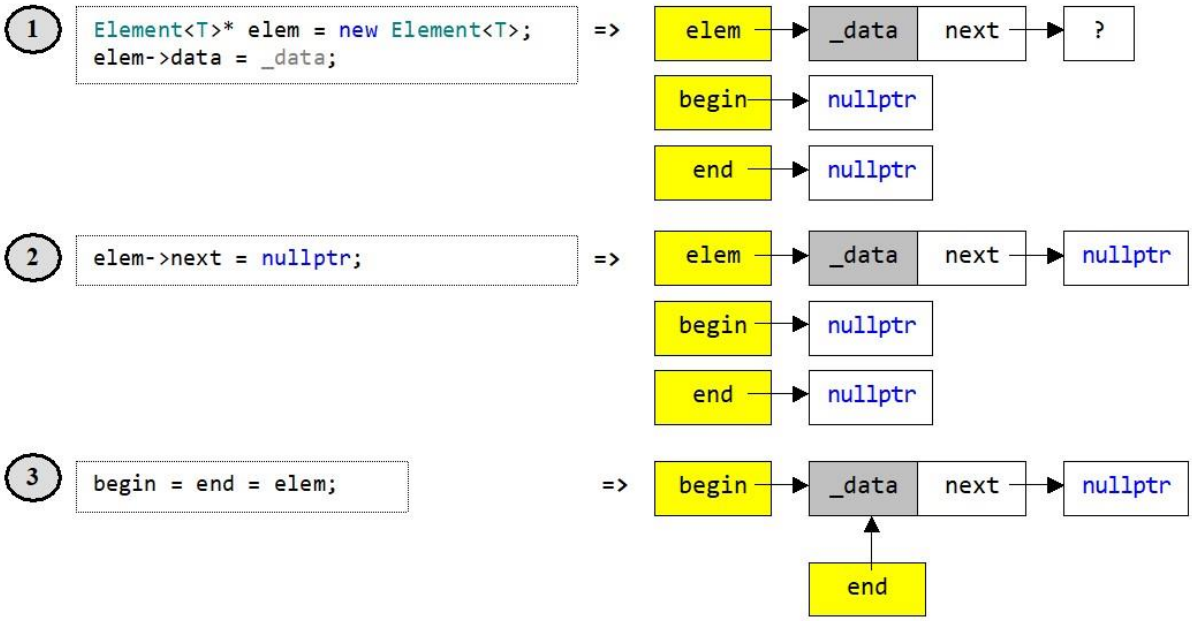


Рисунок 4. Добавление элемента в конец пустого списка: 1 – создание элемента; 2 – установка указателя next; 3 – установка указателей begin и end

**Добавление элемента в конец существующего списка**

Если список уже существует, то последовательность шагов добавляющих элемент в список следующая (рисунок 5):

1. Создать новый элемент и заполнить его данными (рисунок 5). Заполняются поля data и next. Создание элемента осуществляется таким же образом как в случае с пустым списком:

   Element<T>* elem = new Element<T>;

   elem->data = _data;

   elem->next = nullptr;
2. Установить указатель next элемента, на который указывает указатель end, в значение адреса элемента elem (рисунок 6)

   end->next = elem;
3. Установить указатель end равным значению указателя elem

   end = elem;
4. Увеличить количество элементов в списке на 1

   count++;

   Вышеприведенные шаги более красноречиво отображаются на рисунках 5, 6, 7.

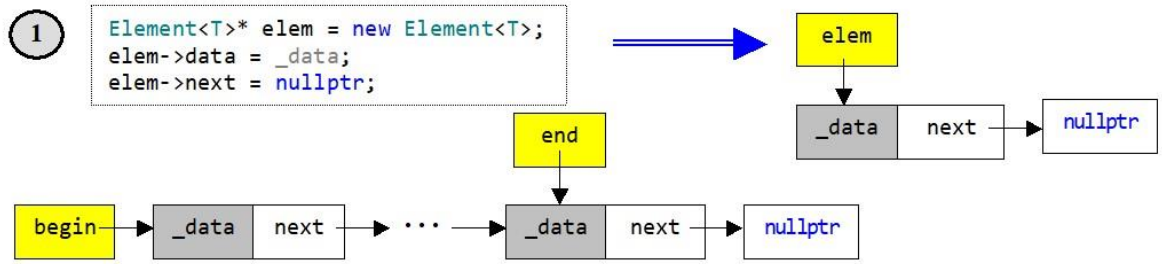


Рисунок 5. Создание элемента и заполнение его данными

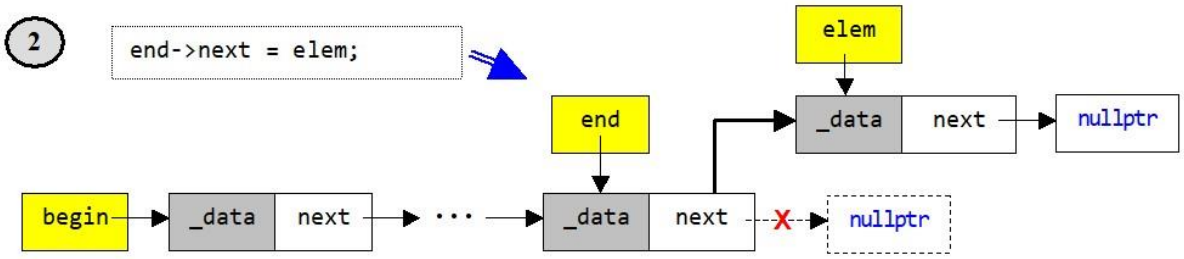


Рисунок 6. Изменение указателя next элемента end

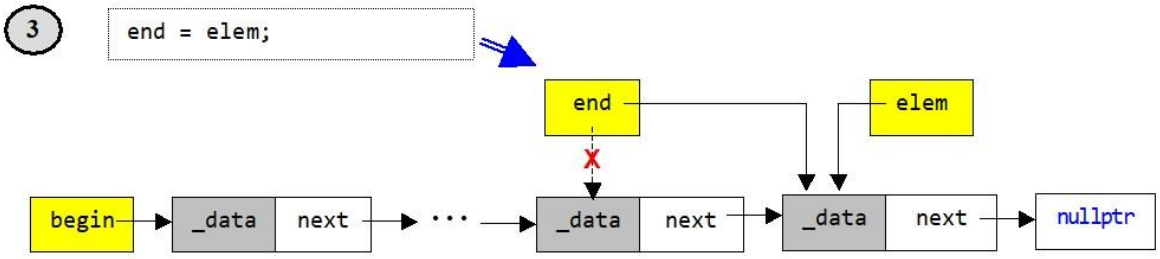


Рисунок 7. Установка указателя end на последний элемент списка

## Вставка элемента в заданную позицию списка

При вставке элемента в заданную позицию списка рассматриваются 2 варианта:

- список пуст (нет элементов);
- список имеет хотя бы один элемент.

Если список пуст, то вставка заменяется добавлением первого элемента, как описано в пункте «Добавление нового элемента в конец пустого списка».

Если в списке есть элементы, то здесь рассматриваются 3 варианта:

- вставка элемента в первую позицию списка;

- вставка элемента внутри списка;
- вставка элемента за последним элементом списка. Если происходит вставка за последним элементом списка, то это есть добавление элемента в конец списка как описано в пункте «Добавление элемента в конец существующего списка».

### Список не пуст. Вставка элемента в первую позицию списка

Если происходит вставка элемента в первую позицию списка, то последовательность операций следующая:

1. Создайть новый элемент (рисунок 8). Выделить память под новый элемент. Заполнить элемент данными.
2. // Выделение памяти под новую ячейку

   Element<T>* elem = new Element<T>;

   elem->data = _data;
3. Установить указатель next элемента на начало списка (рисунок 9).

   elem->next = begin;
4. Установить начало списка на новую ячейку (рисунок 10).

   begin = elem;

   Все вышеперечисленные шаги графически изображены на рисунках 8, 9, 10.

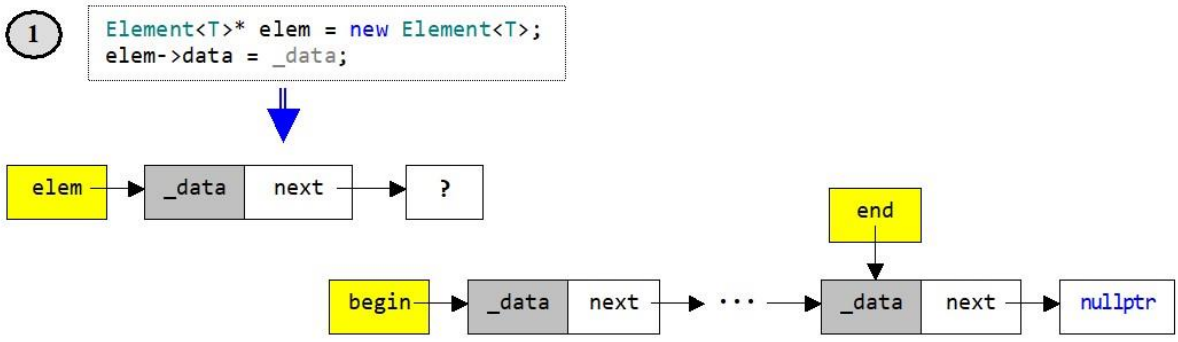


Рисунок 8. Создание нового элемента. Заполнение полей элемента значениями

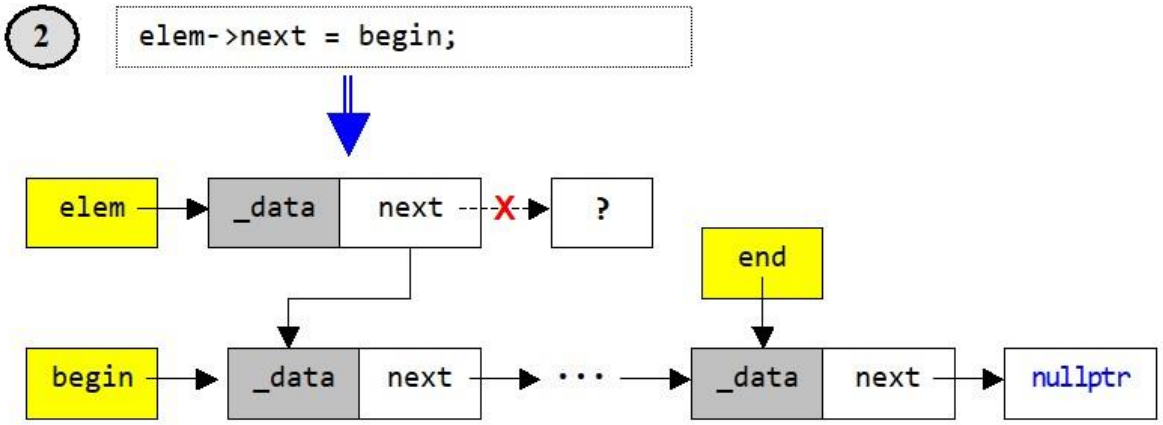


Рисунок 9. Вставка элемента в начало списка. Установка поля next равного адресу первого элемента списка

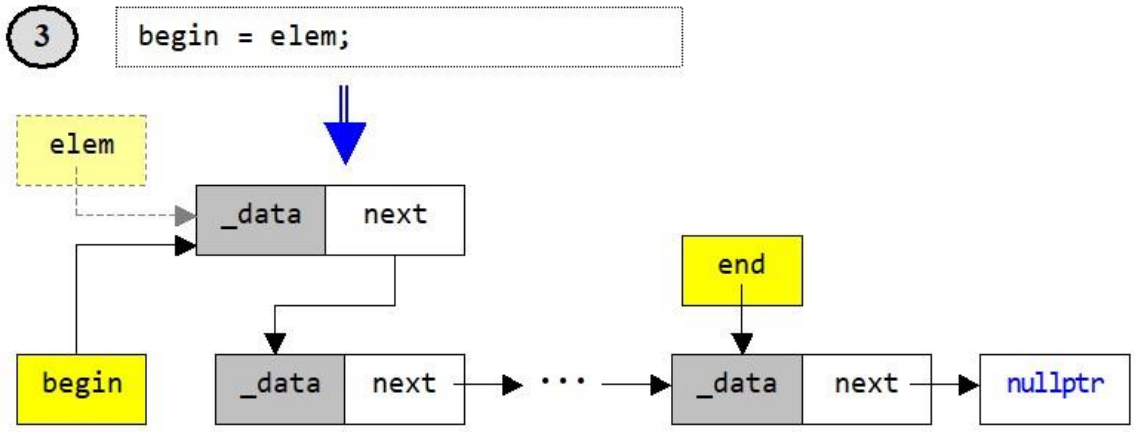


Рисунок 10. Присваивание указателю начала списка адреса новосозданного элемента

**Список не пуст. Вставка элемента в середину списка**

Если элемент вставляется перед второй, третьей и т.д. а также последней позицией списка, то выполняются нижеследующие шаги.

1. Создать элемент. Заполнить поле данных элемента

   Element<T>* elem = new Element<T>;

   elem->data = _data;
2. Определить позицию элемента, который следует перед позицией вставки элемента. Получить указатель на эту позицию.

   Element<T>* elemPrev = Move(index - 1);

здесь Move() – метод, возвращающий позицию элемента, находящегося перед позицией вставки index.

3. Установить указатель next элемента elem, который вставляется

   elem->next = elemPrev->next;

4. Установить указатель next предыдущего элемента elemPrev так, чтобы он указывал на вставляемый элемент elem

   elemPrev->next = elem;

   Предыдущие 4 шага изображены на рисунках 11, 12, 13, 14.

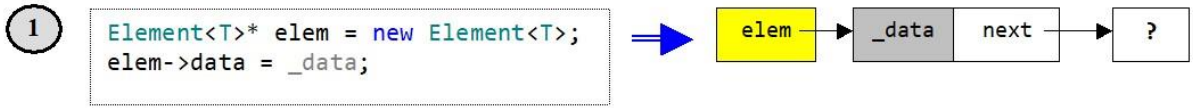


Рисунок 11. Создание элемента. Заполнение элемента данными

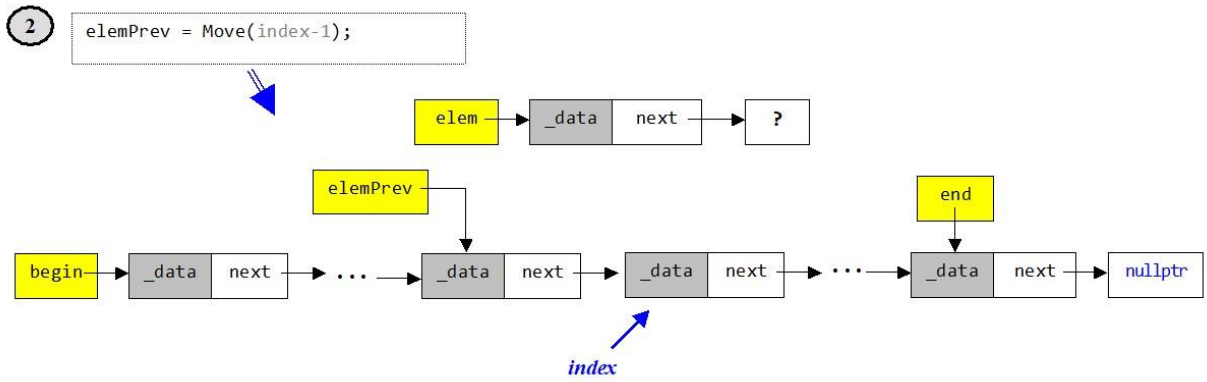


Рисунок 12. Вставка элемента в список. Поиск элемента, который следует перед позицией вставки

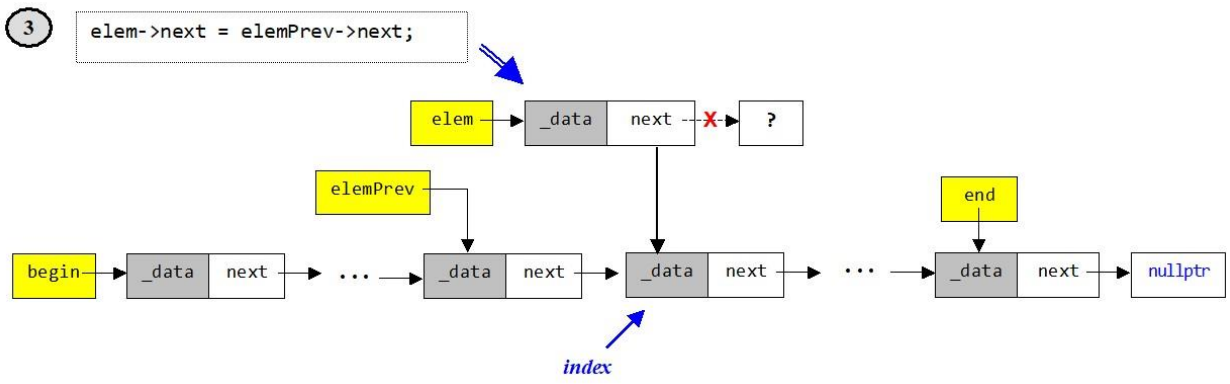


Рисунок 13. Вставка элемента в список. Установка указателя next вставляемого элемента

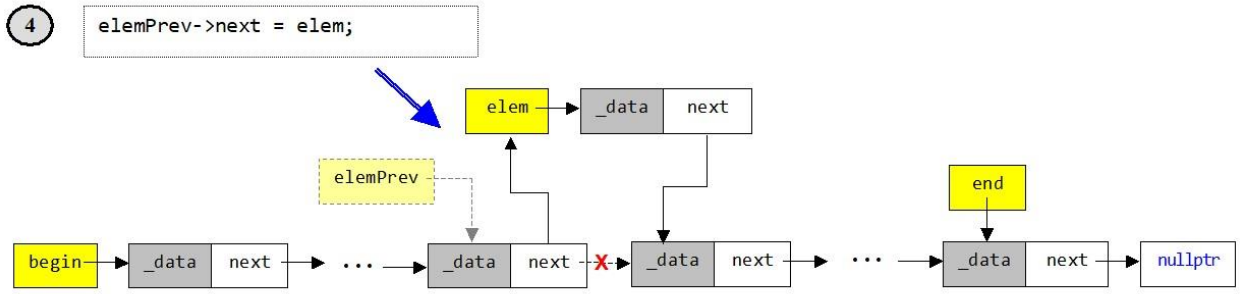


Рисунок 14. Вставка элемента в список. Установка указателя next элемента, следующего перед позицией вставки

## Удаление элемента из списка в заданной позиции

Чтобы удалить элемент из списка, сначала производится проверка, пуст ли список. Если список не пуст (список содержит хотя бы один элемент), то производится удаление элемента.

При удалении элемента из списка рассматриваются 2 возможные ситуации:

- удаляется первый элемент из списка;
- удаляется не первый элемент из списка (второй, третий и т.д.).

### Список не пуст. Удаление первого элемента из списка

Если удаляется первый элемент в списке, то последовательность шагов удаления следующая.

1. Получить адрес первого элемента (рисунок 15)
   Element<T>* delElem = begin;
2. Переместить указатель на начало списка begin на следующий элемент
   begin = begin->next;
3. Освободить память, выделенную для элемента delElem
   delete delElem;
   На рисунках 15, 16, 17 изображен весь процесс удаления первого элемента из списка.

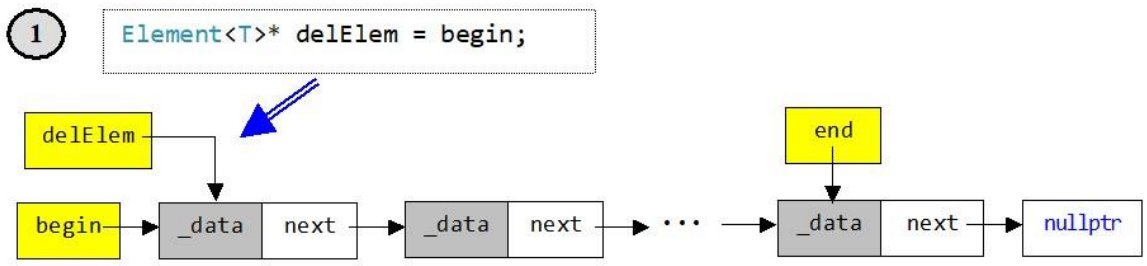


Рисунок 15. Удаление первого элемента из списка. Шаг 1. Создать указатель на первый элемент

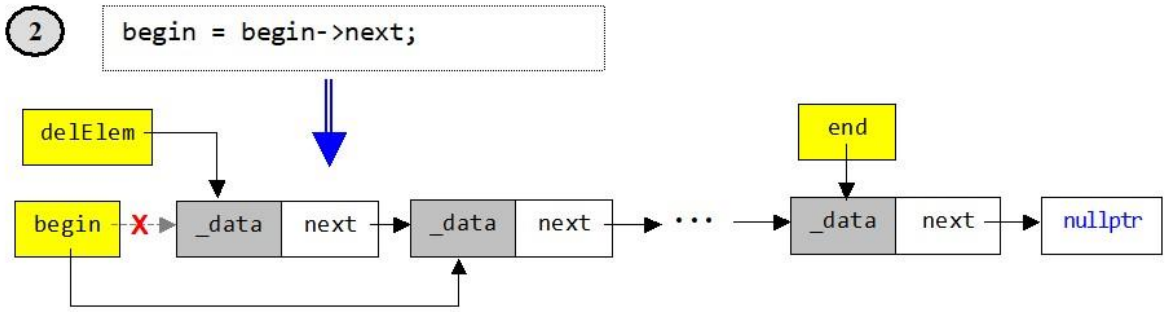


Рисунок 16. Удаление элемента. Шаг 2. Перемещение указателя начала списка на следующий элемент

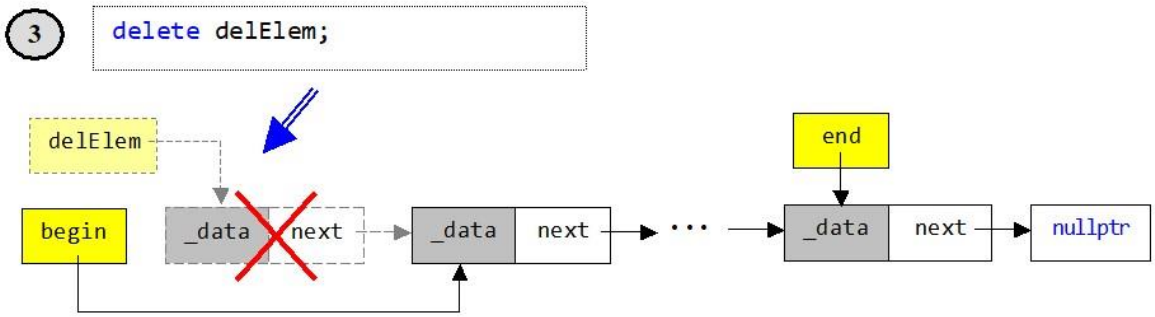


Рисунок 17. Удаление элемента. Шаг 3. Освободить память, выделенную под элемент

**Список не пуст. Удаление не первого элемента из списка**

Если в непустом списке удаляется не первый элемент (второй, третий и т.д., последний), то последовательность шагов следующая.

1. Переместить указатель на позицию, следующую перед удаляемым элементом. Получить элемент, предшествующий удаляемому элементу

Element<T>* elemPrev = Move(index - 1);

здесь Move() – метод, возвращающий элемент по указанной позиции.

2. Запомнить удаляемый элемент

   Element<T>* elemDel = elemPrev->next;

3. Сместить указатель next предыдущего элемента elemPrev в обход удаляемого элемента elemDel

   elemPrev->next = elemDel->next;

4. **У**далить элемент elemDel (освободить память, выделенную для элемента).

   delete elemDel;

   На рисунках 18, 19, 20, 21 схематически рассмотрены вышеприведенные шаги.

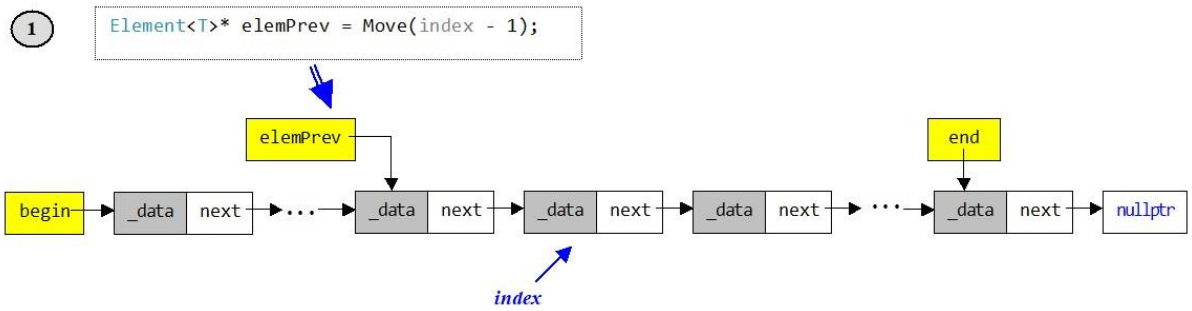


Рисунок 18. Удаление элемента всередине списка. Переход на элемент, следующий перед предыдущим

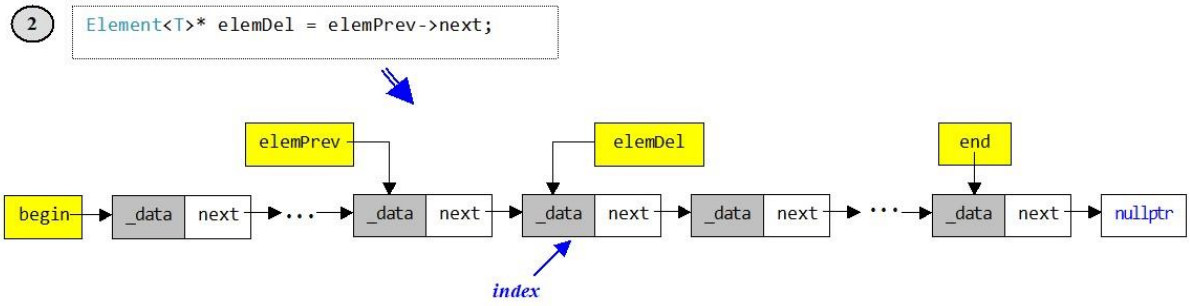


Рисунок 19. Удаление элемента. Получить удаляемый элемент

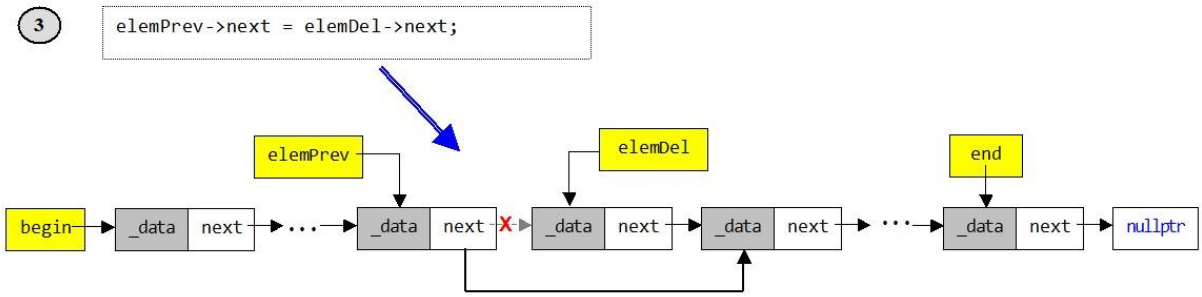


Рисунок 20. Удаление элемента из списка. Смещение указателя next предыдущего элемента в обход удаляемого элемента

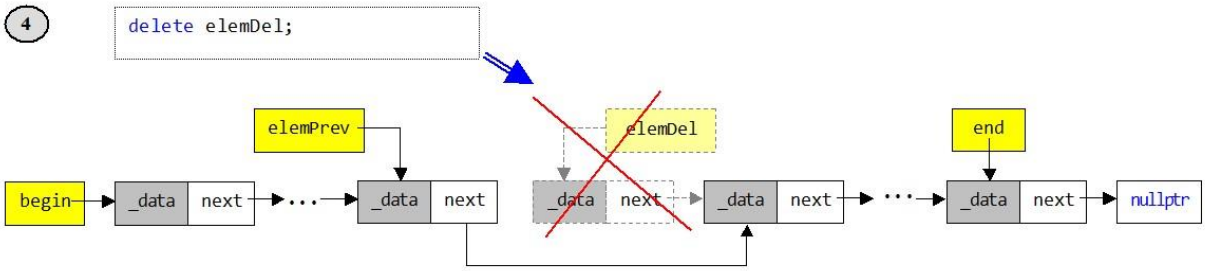


Рисунок 21. Удаление элемента из списка. Особождение памяти, занимаемой элементом

# Двухсвязный список

## Общий вид двухсвязного списка. Основной элемент списка

В двухсвязном списке базовый элемент списка (узел) имеет следующие поля:

- data – данные. Данными могут быть любое значение любого типа;
- next – указатель на следующий элемент;
- prev – указатель на предыдущий элемент.

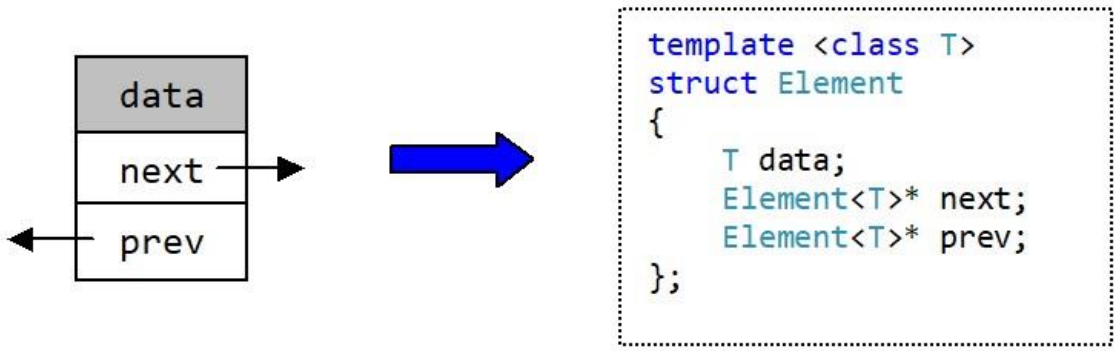


Рисунок 22. Базовый элемент списка

На языке C++ (и на других языках) двухсвязный список создается разработкой специального класса. В классе базовыми данными, представляющими список, являются:

- begin – указатель на первый элемент списка;
- end – указатель на последний элемент списка;
- count – количество элементов в списке.

Общий вид двухсвязного списка изображен на рисунке 23.

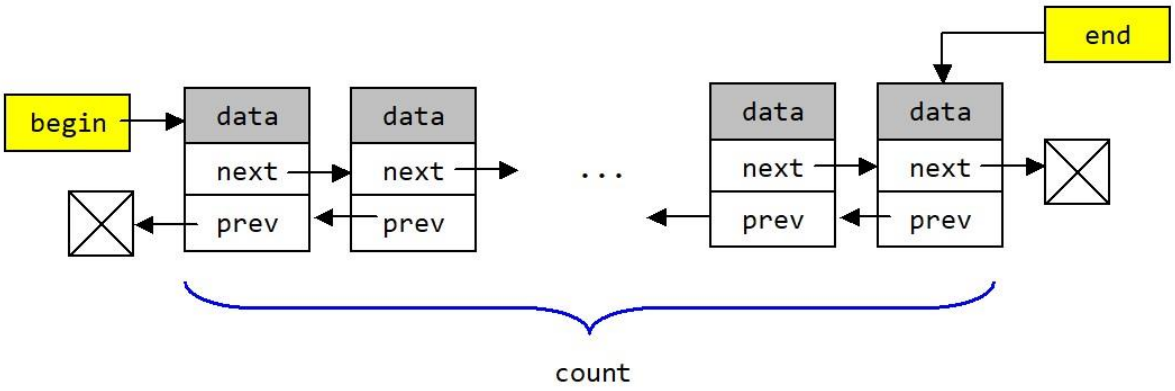


Рисунок 23. Общий вид двусвязного списка

## Создание списка

Перед добавлением новых элементов в список первоочередной задачей является создание пустого списка.

При создании списка можно выделить следующие этапы:

- объявление указателей на начало и конец списка (begin, end);
- установление нулевых значений указателям;
- установка длины списка count в нулевое значение.

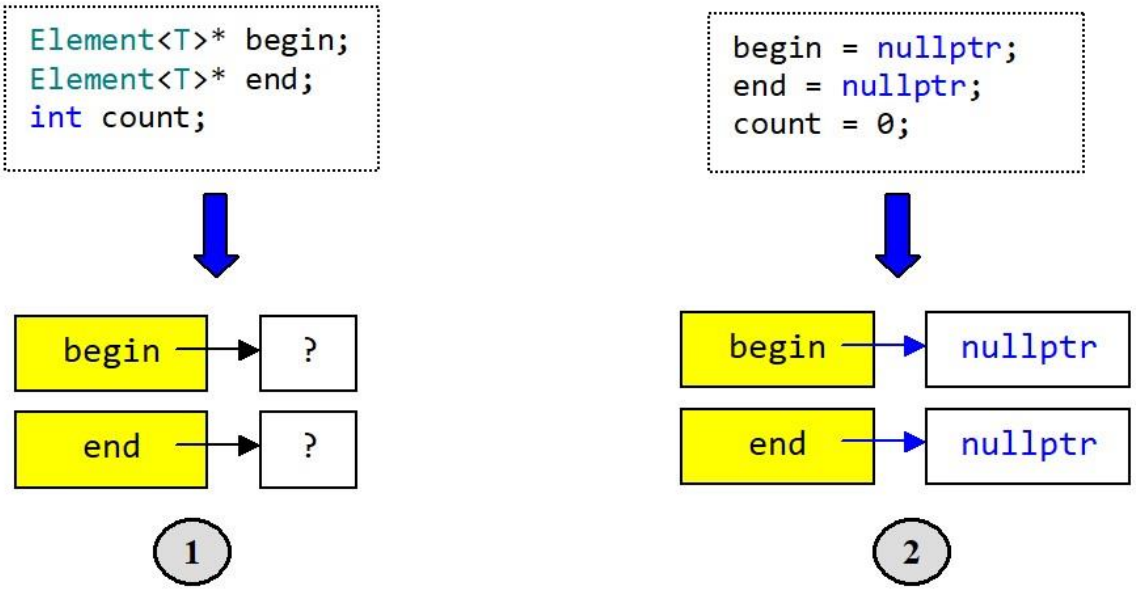


Рисунок 24. Создание списка. Этап 1 – объявление внутренних переменных. Этап 2 – заполнение переменных нулевыми значениями

## Добавление элемента в конец списка

При добавлении элемента в конец списка выделяют следующие две ситуации:

- добавление элемента в пустой список;
- добавление элемента в непустой список (список, содержащий элементы).

### Добавление нового элемента в конец пустого списка

Процесс добавления элемента в конец пустого списка состоит из следующих шагов:

- создать новый элемент elem и заполнить его данными (рисунок 25);
- установить начало и конец списка (begin, end) так, чтобы они указывали на новый элемент elem (рисунок 26);

- увеличить общее количество элементов в списке (поле count).

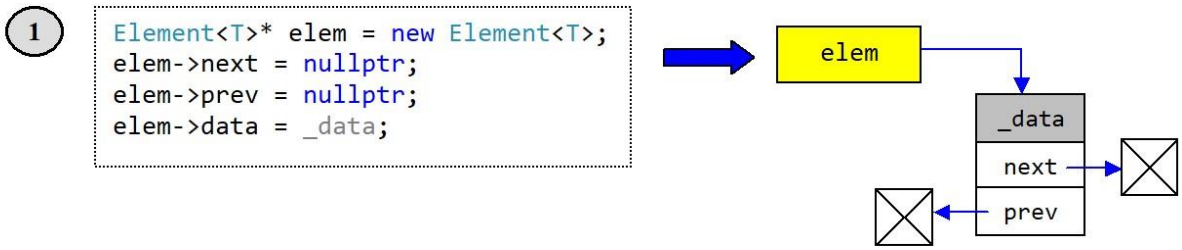


Рисунок 25. Создание элемента и заполнение данных. Заполняются поля next, prev и data элемента elem

На рисунке 26 устанавливается значение указателей begin и end так, что они указывают на созданный элемент elem.

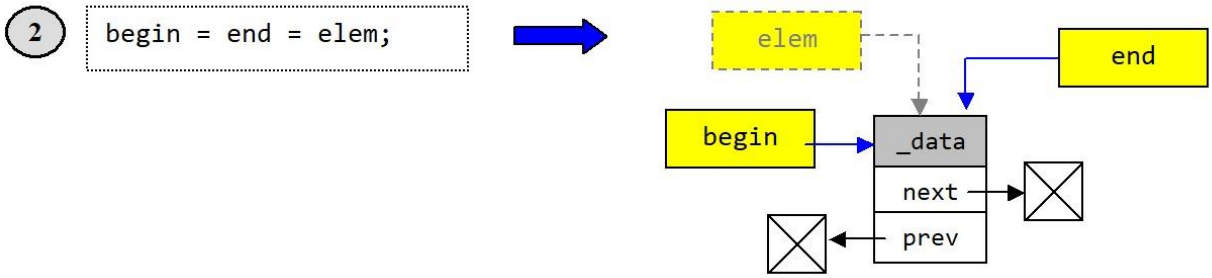


Рисунок 26. Добавление элемента в пустой список. Настройка указателей начала и конца списка

**Добавление нового элемента в список, в котором уже есть элементы**

Добавление нового элемента в пустой список состоит из следующих шагов:

- создание нового элемента elem. Заполнение полей data и next элемента (рисунок 27);
- настройка указателя prev элемента elem таким образом, что он указывает на текущую позицию указателя конца списка end (рисунок 28);
- настройка указателя next последнего элемента end таким образом, чтобы он указывал элемент elem (рисунок 29);
- смещение указателя конца списка elem на одну позицию, определенную указателем elem.

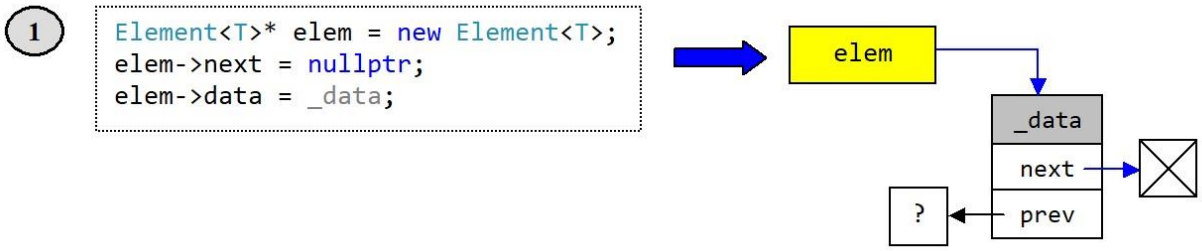


Рисунок 27. Создание элемента. Настройка указателя next и поля данных data элемента elem

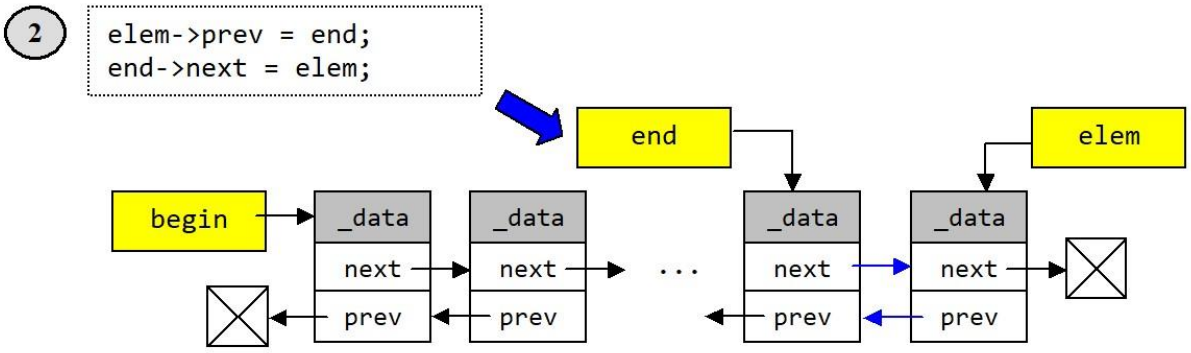


Рисунок 28. Добавление элемента в конец непустого списка. Установка указателя prev элемента elem и указателя next последнего элемента списка end

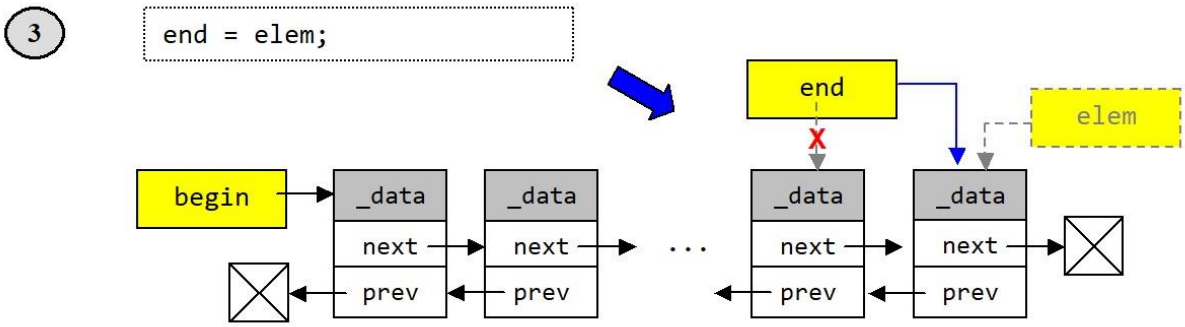


Рисунок 29. Добавление элемента в конец непустого списка. Установка указателя конца списка end на последнюю позицию

## Вставка элемента в заданную позицию списка

В отличие от операции добавления элемента в конец списка существует операция вставки элемента в список.

Если происходит вставка элемента в пустой список, то эта операция такая же, как добавление элемента в пустой список (см. пункт «Добавление нового элемента в конец пустого списка»).

При вставке элемента в список, содержащий элементы, важна позиция вставки. В зависимости от позиции рассматриваются следующие ситуации:

- элемент вставляется в первую позицию списка;
- элемент вставляется во вторую, третью и т.д. позицию списка, а также последнюю позицию списка;
- элемент вставляется за последней позицией списка. В этом случае данная вставка такая же как добавление элемента в конец непустого списка (как описывается в пункте «Добавление нового элемента в список, в котором уже есть элементы»).

### Вставка элемента в первую позицию списка, содержащего элементы

Последовательность действий при вставке элемента в первую позицию списка следующая:

- создать элемент и заполнить его данными (рис. 30). Заполняются поля data и prev;
- установить указатель next элемента elem на первый элемент в списке begin (рисунок 31);
- установить указатель prev первого элемента списка на вставляемый элемент elem (рисунок 32);
- перенаправить указатель begin так, чтобы он указывал на новый элемент elem (рисунок 33);
- увеличить общее количество элементов списка.

На рисунках 30, 31, 32, 33 изображены вышеприведенные шаги в визуализированной форме.

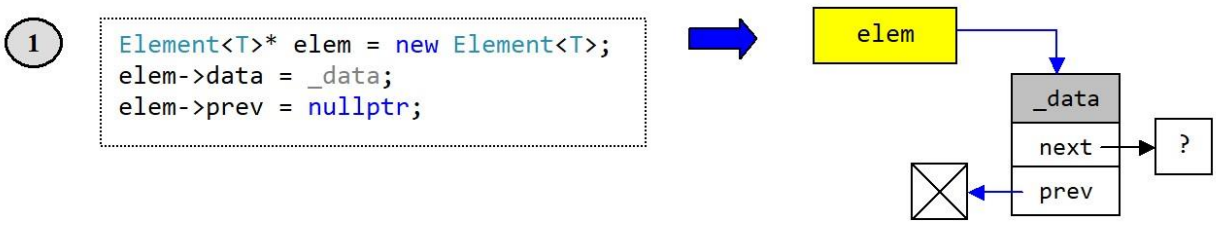


Рисунок 30. Вставка элемента в начало непустого списка. Создание нового элемента elem

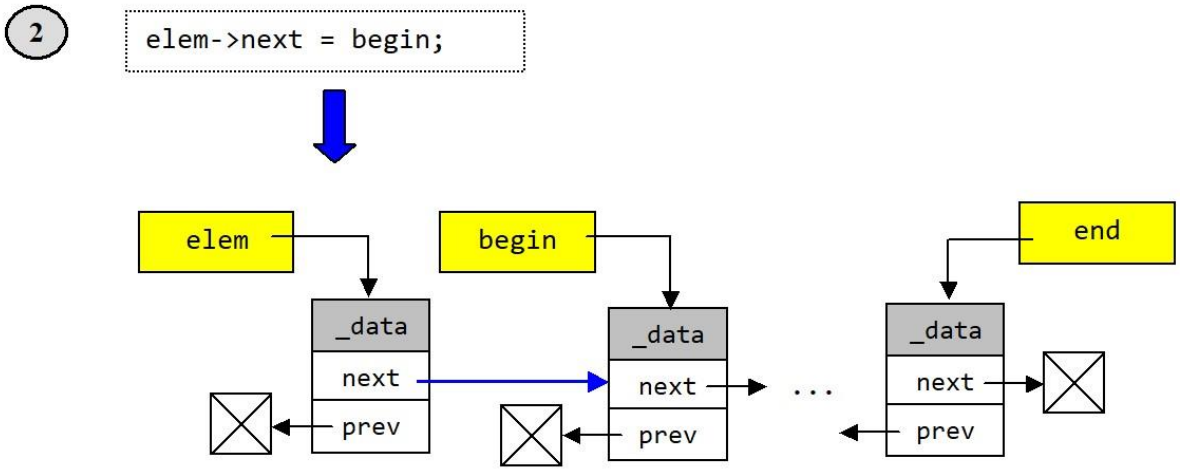


Рисунок 31. Настройка указателя next вставляемого элемента

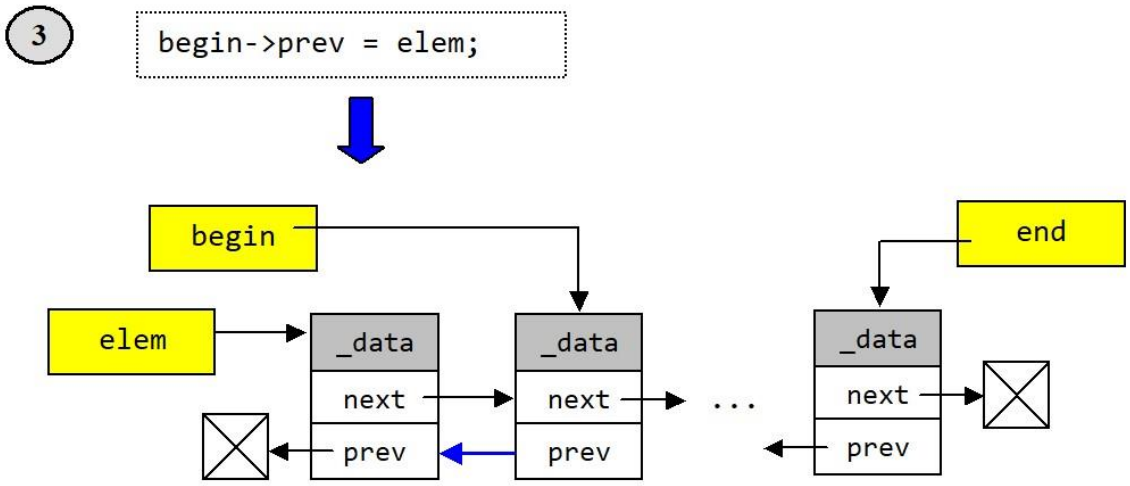


Рисунок 32. Присвоение указателю prev первого вставляемого элемента списка begin адреса элемента elem

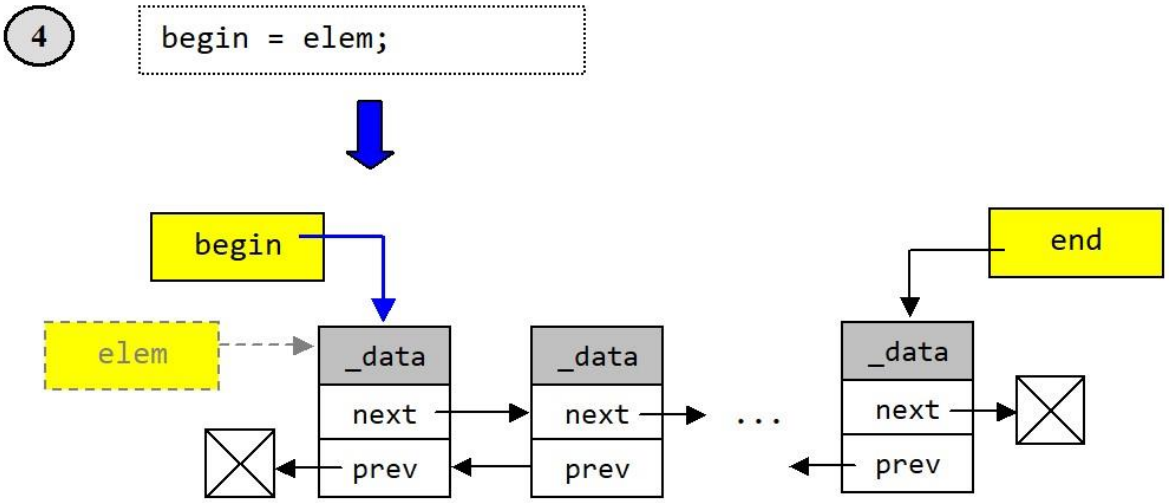


Рисунок 33. Вставка элемента в начало списка. Перенаправление указателя begin на адрес элемента elem

**Вставка элемента в середину списка, содержащего элементы**

- Для вставки элемента в середину списка в некоторую позицию index, нужно выполнить следующую последовательность шагов:
- получить элемент elemPrev, который следует перед позицией вставки. Иными словами, получить элемент в позиции index-1 (рисунок 34);
- получить элемент elemCur, размещаемый в позиции вставки index (рисунок 34);
- создать новый элемент elemIns, который будет вставлен между элементами elemPrev и elemCur. Заполнить поле data элемента (рисунок 35);
- установить указатель next элемента elemIns равным адресу элемента elemCur (рисунок 36);
- установить указатель prev элемента elemIns равным адресу элемента elemPrev (рисунок 36);
- сместить указатель next элемента elemPrev на элемент elemIns (рисунок 37);
- сместить указатель prev элемента elemCur на элемент elemIns (рисунок 37).

Вышеприведенный пошаговый процесс отражен на рисунках 34, 35, 36, 37.

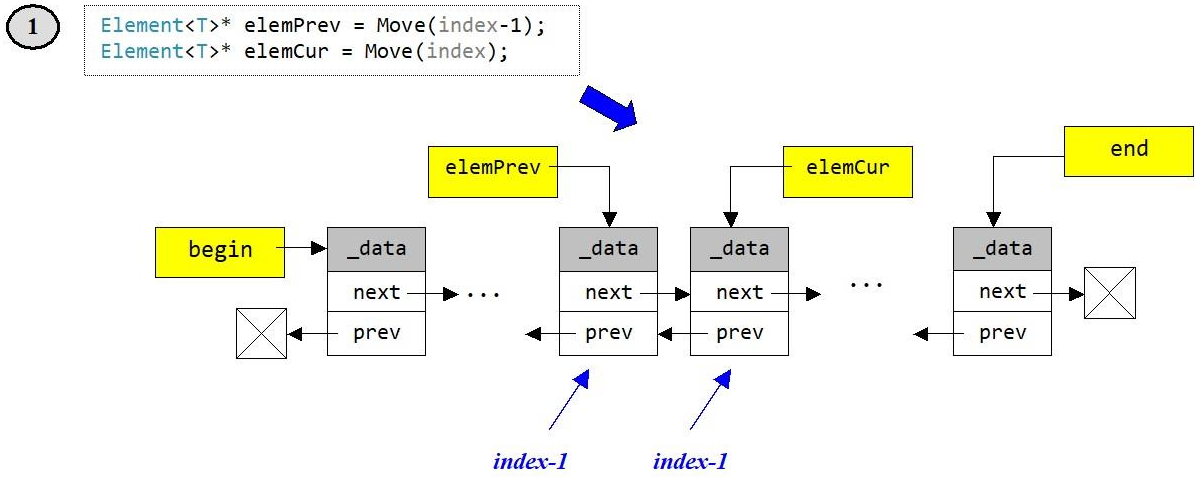


Рисунок 34. Вставка элемента в непустой список. Получить элементы в позициях index-1 и index

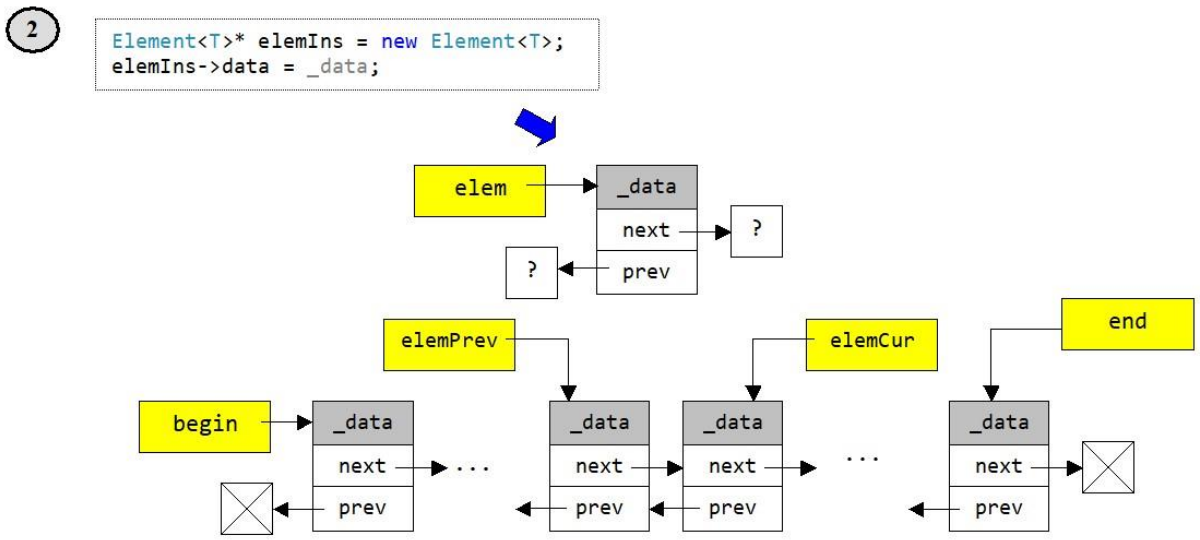


Рисунок 35. Вставка элемента в непустой список. Создание нового элемента. Заполнение поля data

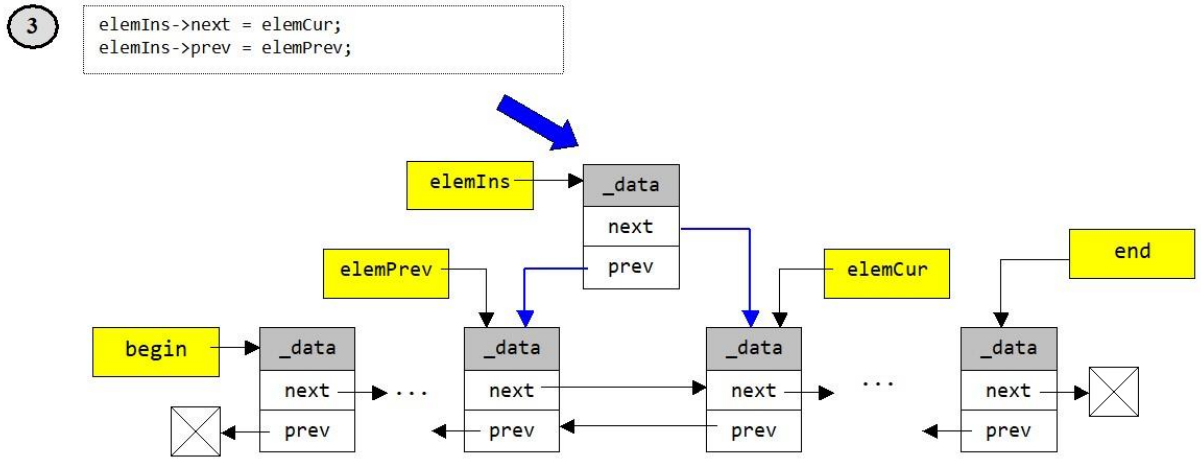


Рисунок 36. Вставка элемента в непустой список. Установка указателей pred и next вставляемого элемента

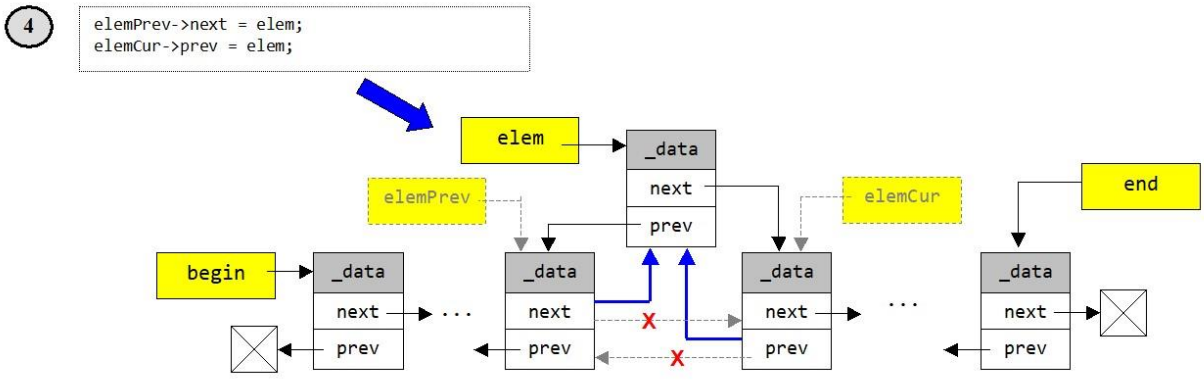


Рисунок 37. Вставка элемента в непустой список. Перенаправление указателя next предыдущего элемента elemPrev и указателя prev следующего элемента elemCur на вставляемый элемент elemIns

## Удаление элемента из списка

При удалении элемента из списка, содержащего элементы, рассматриваются три возможные ситуации:

- удаляется первый элемент списка;
- удаляется не первый и не последний элемент;
- удаляется последний элемент списка.

### Удаление первого элемента из списка

При удалении первого элемента из списка последовательность шагов следующая:

- запомнить адрес первого элемента в значение elem (рисунок 38);
- изменить значение указателя begin так, чтобы он указывал на следующий элемент (рисунок 39). Если следующего элемента нет, то указатель будет иметь значение nullptr;
- установить значение указателя prev в нулевое значение (рисунок 40);
- освободить память, выделенную для элемента elem;
- уменьшить общее количество частей на 1.

Если в списке только один элемент, то приведенные выше шаги также подойдут для этого случая. Ниже на рисунках 38, 39, 40, 41 изображены основные шаги этого процесса.

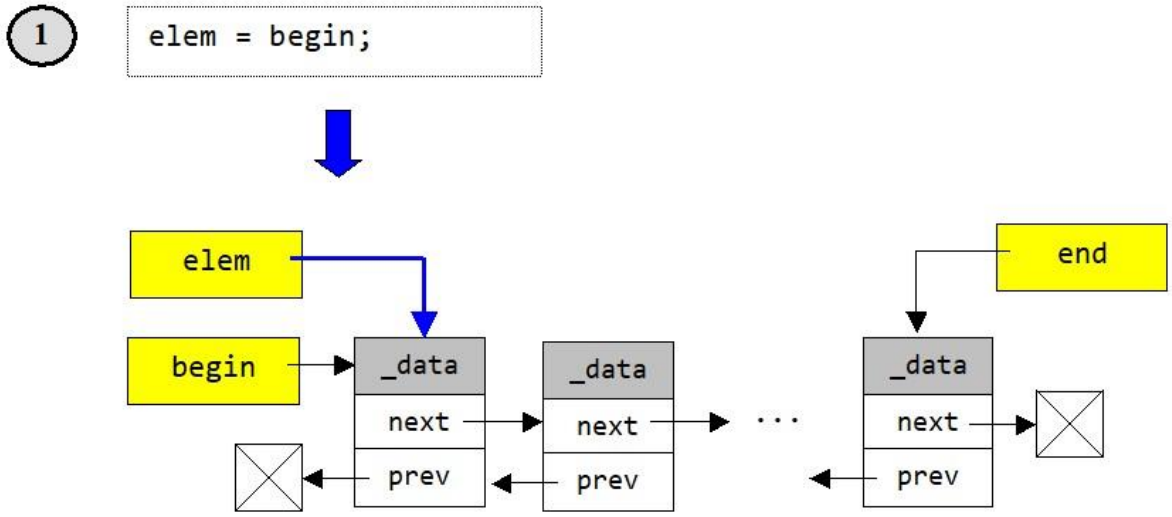


Рисунок 38. Удаление первого элемента из списка. Шаг 1. Запомнить значение начала списка begin

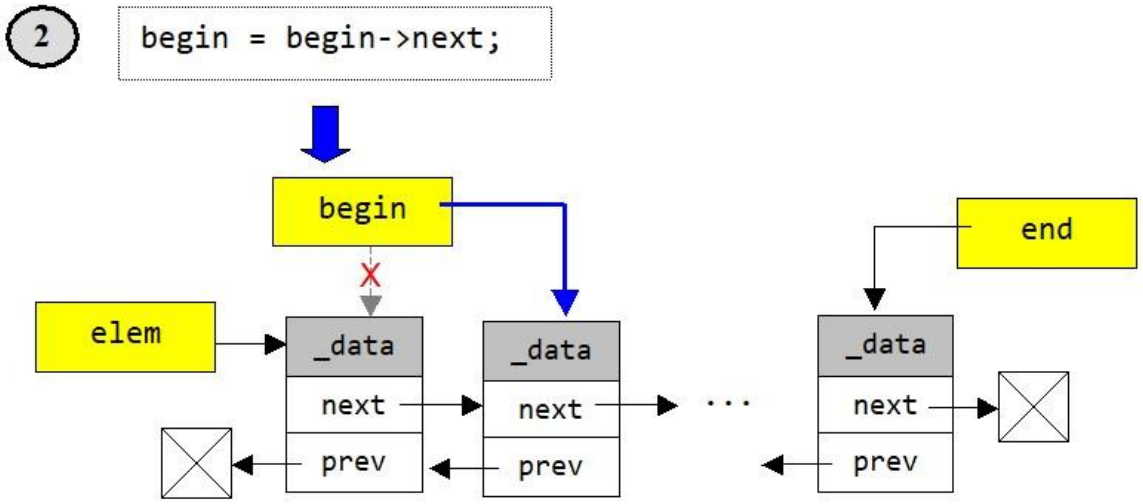


Рисунок 39. Удаление первого элемента из списка. Шаг 2. Смещение указателя begin на следующий элемент

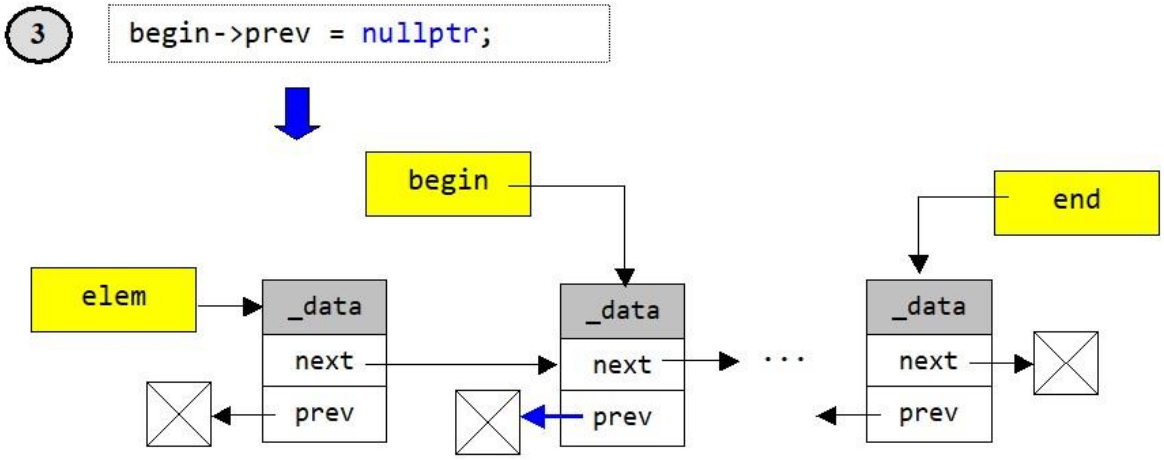


Рисунок 40. Удаление первого элемента из списка. Шаг 3. Установка указателя prev начала списка begin в нулевое значение

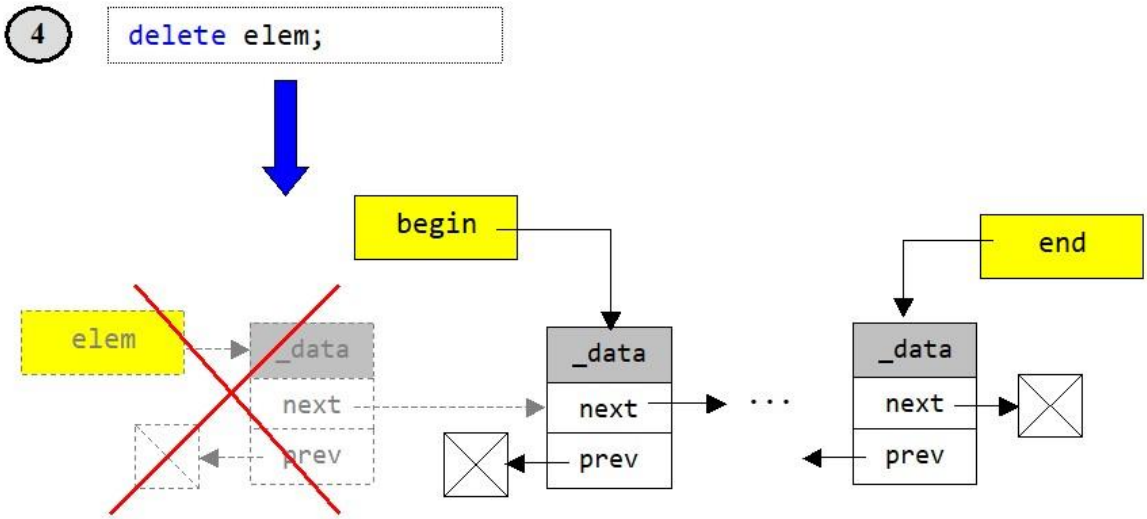


Рисунок 41. Удаление первого элемента из списка. Уничтожение элемента elem (освобождение памяти)

**Удаление из середины списка**

Если элемент удаляется из середины списка, то последовательность шагов следующая:

- на основе индекса элемента index получить указатель на удаляемый элемент elem, а также на предыдущий elemPrev и следующий elemNext элементы (рисунок 42);
- установить указатель next элемента elemPrev равным адресу элемента elemNext (рисунок 43);
- установить указатель prev элемента elemNext равным адресу элемента elemPrev (рисунок 43);
- удалить (освободить память) элемент elem (рисунок 44).

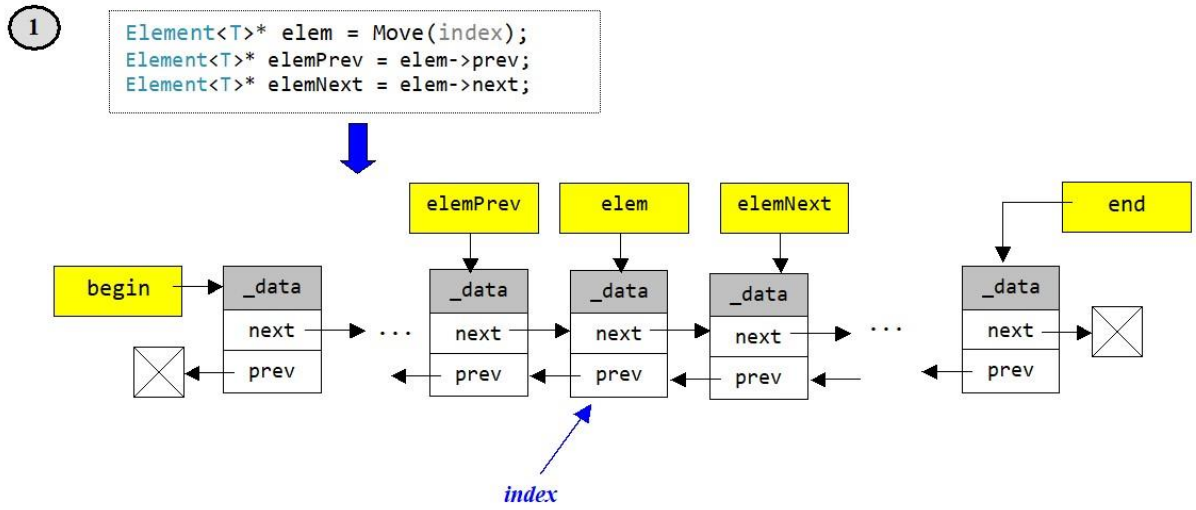


Рисунок 42. Удаление элемента из списка. Шаг 1. Получить удаляемый элемент, а также предыдущий и последующий элементы

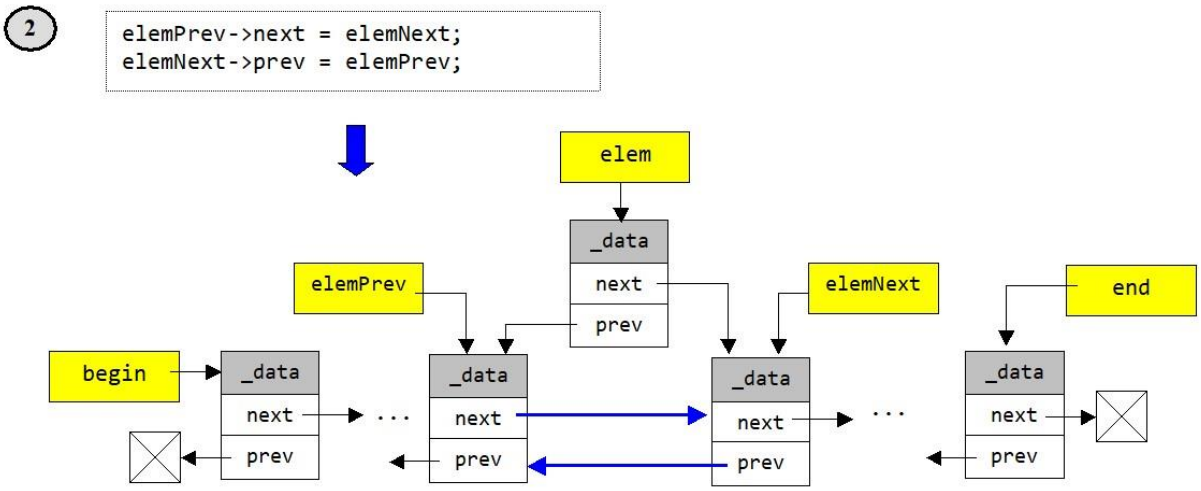


Рисунок 43. Удаление элемента из списка. Шаг 2. Установка указателей предыдущего элемента elemPrev и следующего элемента elemNext в обход элемента elem

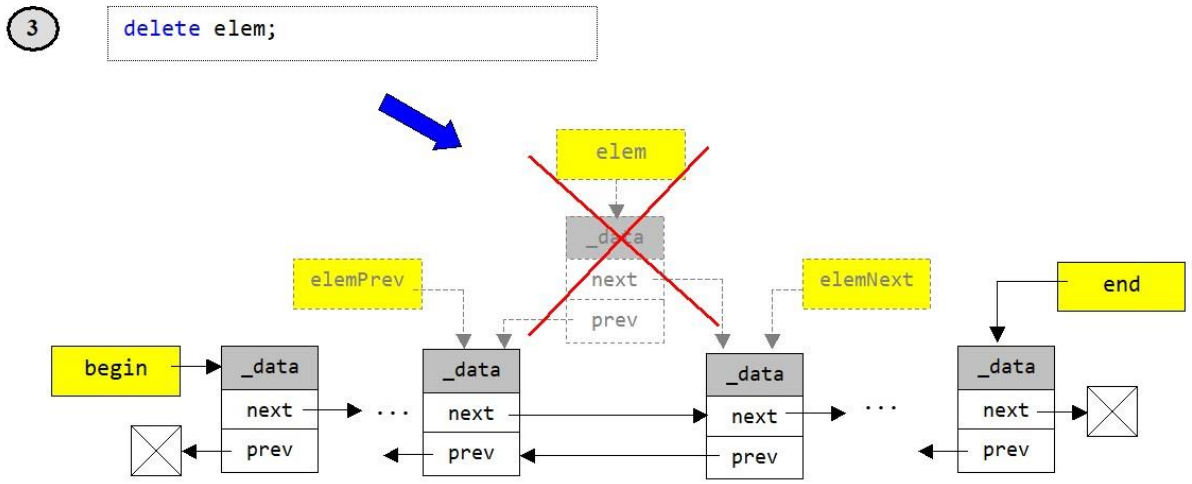


Рисунок 44. Удаление элемента из списка. Шаг 3. Удаление элемента elem

### Удаление последнего элемента из списка

Для удаления последнего элемента из списка выполняется следующая последовательность действий:

- запомнить адрес последнего элемента в списке в указателе elem;
- переместить на предыдущий элемент указатель end, указывающий на последний элемент в списке;
- освободить память, выделенную под последний элемент в списке;
- установить указатель next элемента end в нулевое значение.

Операция удаления последнего элемента из списка подходит для случая, когда в списке есть только один элемент.

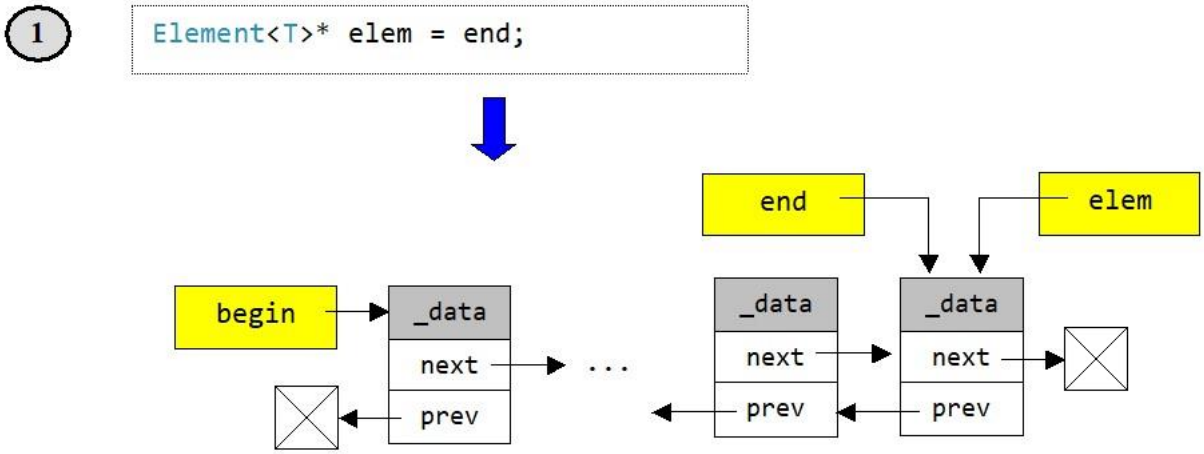


Рисунок 45. Удаление последнего элемента из списка. Запомнить адрес последнего элемента

2

```
end = end->prev;
```

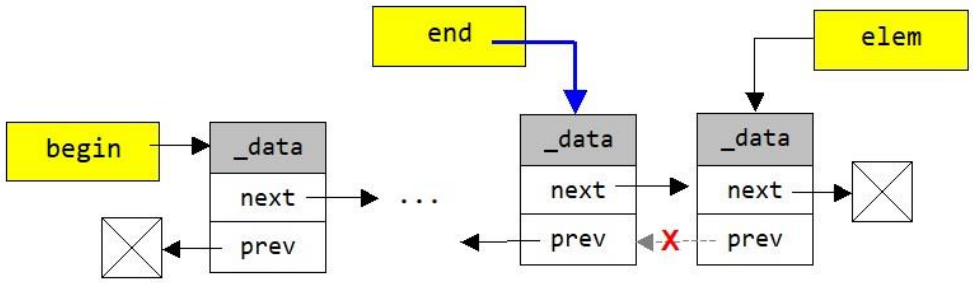


Рисунок 46. Удаление последнего элемента из списка. Перемещение указателя end на предыдущий элемент списка

3

```
delete elem;
```

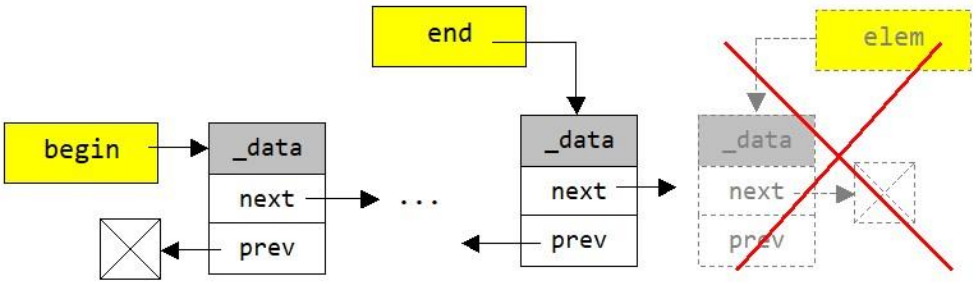


Рисунок 47. Удаление последнего элемента из списка. Освобождение памяти, выделенной под последний элемент

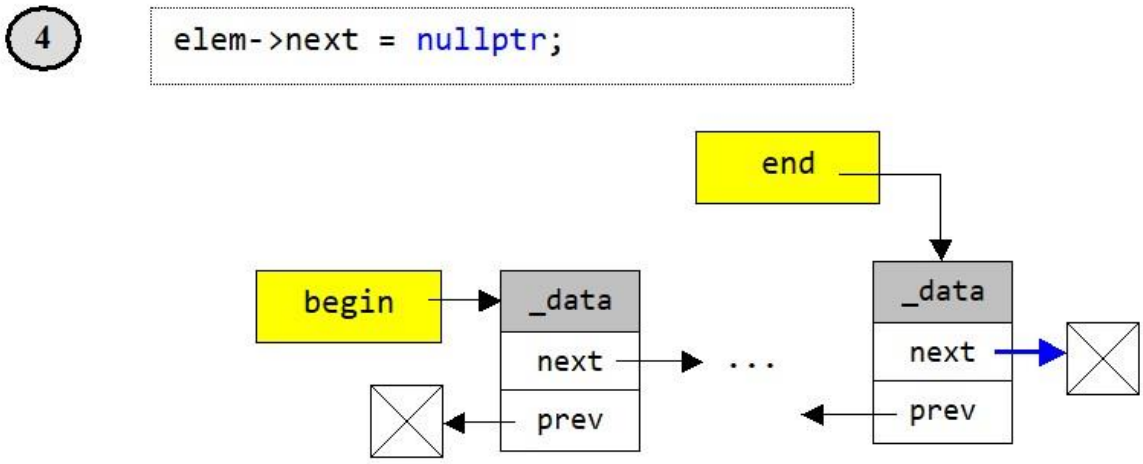


Рисунок 48. Удаление последнего элемента из списка. Установка в нулевое значение указателя next последнего элемента списка

# Шаблоны C++

Шаблоны являются основой для универсального программирования в C++. В качестве строго типизированного языка C++ требует, чтобы все переменные имели определенный тип, явно объявленный программистом или выводил компилятором. Однако многие структуры и алгоритмы данных выглядят одинаково независимо от типа, на который они работают. Шаблоны позволяют определить операции класса или функции и разрешить пользователю указать, какие типы этих операций должны работать.

**Определение и использование шаблонов**

Шаблон — это конструкция, которая создает обычный тип или функцию во время компиляции на основе аргументов, которые пользователь предоставляет для параметров шаблона. Например, можно определить шаблон функции следующим образом:

```
template <typename T>
T minimum(const T& lhs, const T& rhs)
{
    return lhs < rhs ? lhs : rhs;
}
```

Приведенный выше код описывает шаблон универсальной функции с одним параметром *типа T*, возвращаемое значение и параметры вызова (lhs и rhs) всех этих типов. Вы можете назвать параметр типа, который вы хотите, но по соглашению одноглавные буквы чаще всего используются. *T* является параметром шаблона; **typename** ключевое слово говорит, что этот параметр является заполнителем для типа. При вызове функции компилятор заменит каждый экземпляр T конкретным аргументом типа, заданным пользователем или выведенным компилятором. Процесс, в котором компилятор создает класс или функцию из шаблона, называется экземпляром шаблона. minimum<int> Это экземпляр шаблона minimum<T>.

В другом месте пользователь может объявить экземпляр шаблона, специализированного для int. Предположим, что get_a() и get_b() — это функции, возвращающие int:

```
int a = get_a();
int b = get_b();
int i = minimum<int>(a, b);
```

Тем не менее, поскольку это шаблон функции, и компилятор может выводить тип T из аргументов *a* и *b*, можно вызвать его так же, как обычная функция:

```
int i = minimum(a, b);
```

При обнаружении последней инструкции компилятор создает новую функцию, в которой каждое вхождение *T* в шаблоне заменяется следующим **int** образом:

```
int minimum(const int& lhs, const int& rhs)
{
```

```
    return lhs < rhs ? lhs : rhs;
}
```

Правила того, как компилятор выполняет вычет типов в шаблонах функций, основаны на правилах для обычных функций.

Пример: шаблон функции-член шаблона класса.

```
// member_function_templates2.cpp
template<typename T>
class X
{
public:
   template<typename U>
   void mf(const U &u)
   {
   }
};

int main()
{
}
```

# Задание на лабораторную работу

Разработать библиотеку с собственными классами, реализующие односвязный и двухсвязный списки.

- ❖ Односвязный список. Обязательно реализовать операции:
  - добавить элемент в начало списка – push_front(T).
  - добавить элемент в конец списка – push_back(T),
  - добавить элемент в указанное положение списка – insert(int, T),
  - удалить элемент в конце списка – pop_back(),
  - удалить элемент начале списка – pop_front(),
  - удалить из списка совпадающие с элементом значения – remove(T),
  - вернуть количество элементов в списке – size(),
  - вернуть ссылку на первый элемент – front(),
  - вернуть ссылку на последний элемент – back(),
  - проверить список пуст ли список – empty(),
  - реализовать итератор для хождения по списку в обе стороны (+ перегрузка операторов для класса списка: ++, --).
- ❖ Двухсвязный список. Обязательно реализовать операции:
  - добавить элемент в начало списка – push_front(T).
  - добавить элемент в конец списка – push_back(T),
  - добавить элемент в указанное положение списка – insert(int, T),
  - удалить элемент в конце списка – pop_back(),
  - удалить элемент начале списка – pop_front(),
  - удалить из списка совпадающие с элементом значения – remove(T),
  - вернуть количество элементов в списке – size(),
  - вернуть ссылку на первый элемент – front(),
  - вернуть ссылку на последний элемент – back(),
  - проверить список пуст ли список – empty(),
  - реализовать итератор для хождения по списку в обе стороны (+ перегрузка операторов для класса списка: ++, --).
- ❖ Уточнение разработки:
  - придерживаться такому же названию функций,
  - класс односвязного списка назвать OneList,
  - класс двухсвязного списка назвать DoubleList,

- класс элемента списка назвать Element,
- список должен поддерживать любой тип данных,
- при динамическом выделении памяти разрешается пользоваться возможностями C++ (new, delete).

Разработать программу обработки строки. Строка храниться в виде односвязного или двухсвязного списка. Данные вводятся с консоли, результат выводится в консоль. Вариант с логикой обработкой строки был выдан на 1 лабораторной работе.

Оценка лабораторной работы:

- до 44 – реализация хранения и обработки строки (по выданному варианту) в виде односвязного списка,
- до 54 – задание «до 44» и реализация хранения и обработки строки (по выданному варианту) в виде двухсвязного списка.

# Источники


1. Линейный односвязный список. Общие сведения. Базовые операции над списком.
2. Линейный двухсвязный (двунаправленный) список. Общие сведения.
3. Шаблоны C++