Практика 2. Консольное приложение «Список на основе адресных связей»

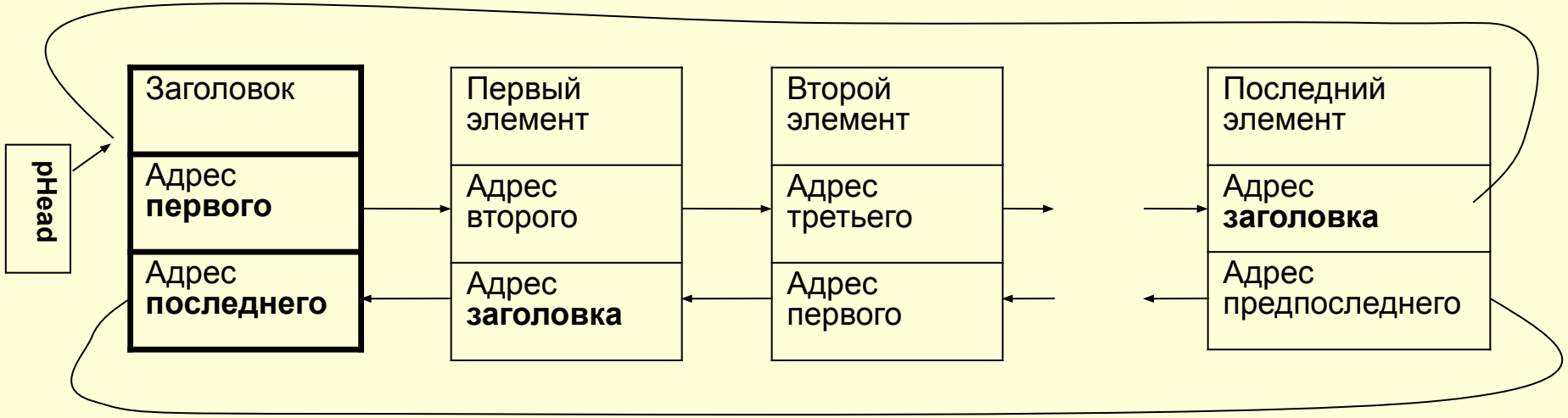
Тип списка: двунаправленный (двусвязный), замкнутый (кольцевой), с заголовочным элементом.

Структура элементов списка:

- информационная составляющая (целое число)
- поле указателя на следующий элемент **(next / sled / right / prav)**
- поле указателя на предыдущий элемент **(prev / pred / left / lev)**

Реализуемые операции:

- проход в прямом и обратном направлении
- поиск в прямом и обратном направлении
- добавление нового элемента после заданного элемента
- удаление заданного элемента



Практика 2. Структура приложения

1. **Объявление ссылочного типа для адресации элементов списка (pDLL_Item)**
2. **Объявление записи-структуры элементов списка (TDLL_Item)**
3. **Объявление указательной переменной для адресации заголовочного элемента (pHead)**
4. Объявление переменной-счетчика числа элементов (count)
5. **Объявление и реализация двух подпрограмм поиска**
6. **Объявление и реализация двух подпрограмм прохода по списку с выводом информации на экран**
7. **Объявление и реализация подпрограммы добавления элемента после заданного**
8. Объявление подпрограммы удаления заданного элемента
9. **Главная программа организации простейшего диалога с пользователем**

Практика 2. Рекомендации по реализации основной программы

1. Создание **заголовочного элемента** (используем **pHead**) с установкой адресных полей в значение **pHead**, установка счетчика в 0
2. Создание пользовательского **меню** с необходимым набором **команд** (Поиск, Добавление ПОСЛЕ, Удаление) и организация **диалогового цикла**
3. Реализация указанных команд:
   - **запрос** необходимых данных у пользователя
   - **вызов** соответствующей подпрограммы
   - **анализ** результата выполнения подпрограммы с **выводом** сообщений пользователю
   - **вывод** на экран текущего состояния списка с помощью соответствующей подпрограммы

Практика 2. Рекомендации по реализации подпрограмм вывода списка на экран

1. Тип - процедурный, формальных параметров нет
2. Объявить **локальную** указательную переменную для адресации элементов списка (**pCurrent**)
3. Установить ее начальное значение в адрес первого (последнего) **реального** элемента списка:

   pCurrent := pHead^.next; // или **prev**
4. Оформить цикл типа **WHILE** для прохода по списку, условие окончания - достижение заголовочного элемента (сравнение **pCurrent** и **pHead**)
5. В теле цикла:

   - записать оператор **вывода** информационной части очередного элемента на экран, используя локальный указатель **(pCurrent^.inf)**
   - перейти к **следующему (предыдущкму)** элементу за счет изменения значения локального указателя (**pCurrent**) на адрес следующего (предыдущего) элемента

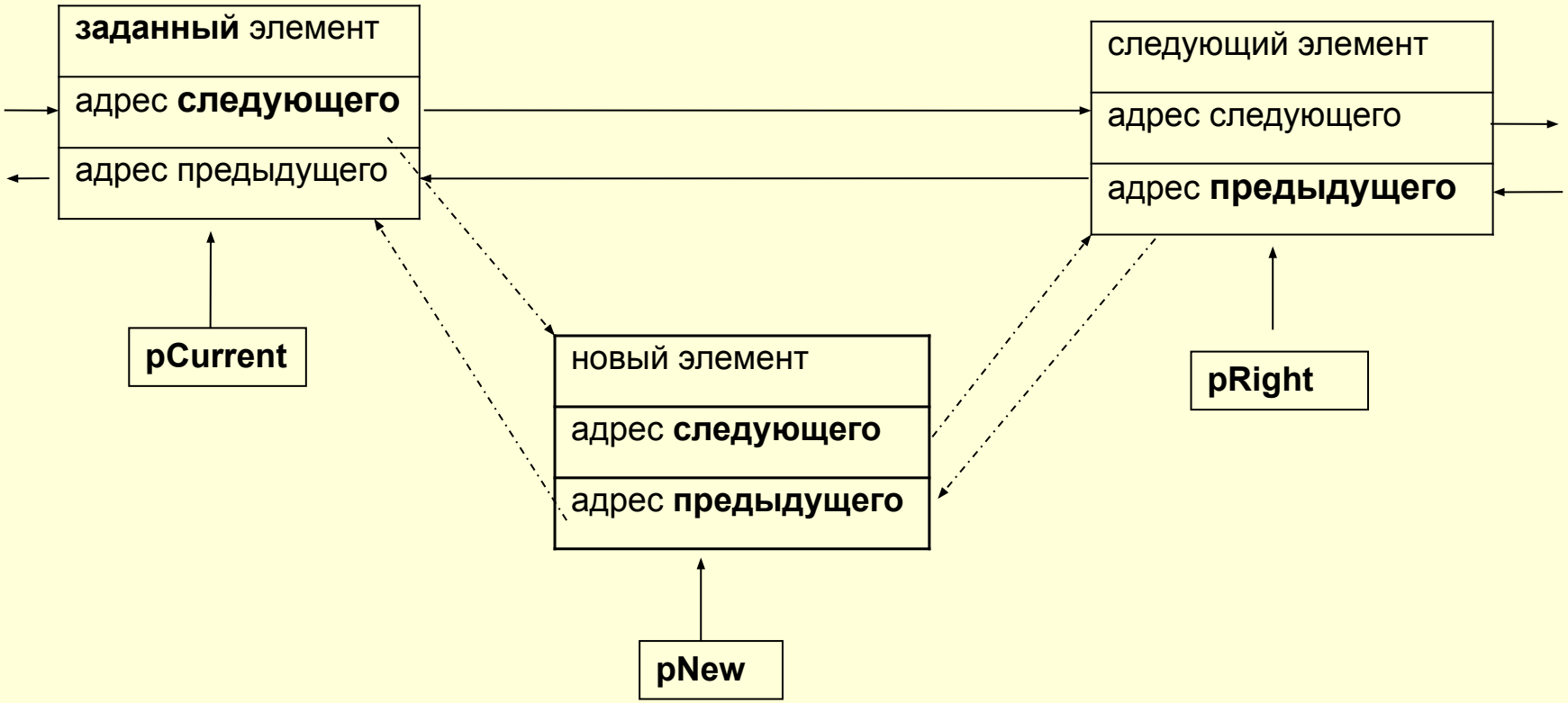
Практика 2. Рекомендации по реализации подпрограмм поиска элемента в списке

1. Тип - **функция**, **один** формальный параметр, **результат** — адрес элемента (возможно нулевой) (тип **pDLL_Item**)
2. Объявить **локальную** указательную переменную для адресации элементов списка (**pCurrent**)
3. Установить ее начальное значение в адрес первого (последнего) **реального** элемента списка
4. Оформить цикл типа **while** для прохода по списку с условием окончания по достижению заголовочного элемента
5. В теле цикла:
   - оператором **if** сравнить информационную часть очередного элемента **(pCurrent^.inf)** с заданным числом
   - если значения НЕ совпадают, то перейти к **следующему (предыдущкму)** элементу за счет изменения значения локального указателя (**pCurrent**) на адрес следующего (предыдущего) элемента
   - при совпадении установить результат работы (**result/return**) как значение указателя **pCurrent** и прервать цикл с помощью **break**
6. Установить отрицательный результат поиска (**result := nil**)

Практика 2. Рекомендации по реализации подпрограммы добавления нового элемента после заданного элемента

1. Тип - **функция**, **ДВА** формальных параметра, **результат** — логический (да/нет)
2. Объявить **ТРИ локальные** указательные переменные:
   - для адресации **заданного** элемента списка (**pCurrent**)
   - для адресации **нового** добавляемого элемента **(pNew)**
   - для адресации правого соседа заданного элемента (**pRight**)
3. Проверить **наличие** элементов в списке (счетчик **count**)
4. Если список **пустой**, то **создать** новый (первый) элемент с использованием **pNew**, **настроить** все адресные связи у **нового** элемента и у **заголовочного** элемента (**pHead**), увеличить счетчик и установить **положительный** результат операции (**result := true**)
5. Если список **НЕ** пустой, то **вызвать** функцию поиска заданного элемента **pCurrent := Poisk (число)**
6. Если поиск **удачен** (**pCurrent <> nil**), то **создать** новый элемент (указатель **pNew**), занести в него новое число (**pNew^.inf := число**), установить указатель **pRight := pCurrent^.next**, **установить** обе адресные связи у **нового** элемента, **изменить** связи у его **левого** и **правого** соседа, увеличить счетчик и установить результат в **true**
7. Если поиск **неудачен**, то установить **отрицательный** результат операции (**result := false**)

Схема добавления нового элемента ПОСЛЕ заданного



Практика 2. Рекомендации по реализации подпрограммы удаления заданного элемента

1. Тип - **функция**, **ОДИН** формальный параметр, **результат** — логический (да/нет)
2. Объявить ТРИ **локальные** указательные переменные:
   - для адресации **заданного** элемента списка (**pCurrent**)
   - для адресации **левого** соседа заданного элемента **(pLeft)**
   - для адресации **правого** соседа заданного элемента **(pRight)**
3. Проверить список на пустоту (счетчик **count**) и если он **пустой**, то оформить **отрицательный** результат (**result := false**)
4. Если список **НЕ** пустой, то **вызвать** функцию поиска заданного элемента: **pCurrent := Poisk (число)**
5. Если поиск **удачен** (**pCurren <> nil**), то установить указатель **pLeft** в адрес левого соседа (**pLeft := pCurrent^.prev**), указатель **pRight** в адрес правого соседа (**pRight := pCurrent^.next**)
6. Изменить **правую** ссылку у **левого** соседа и **левую** ссылку у **правого** соседа, **уменьшить** счетчик и установить результат в **true**
7. Если поиск **неудачен**, то установить **отрицательный** результат операции (**result := false**)

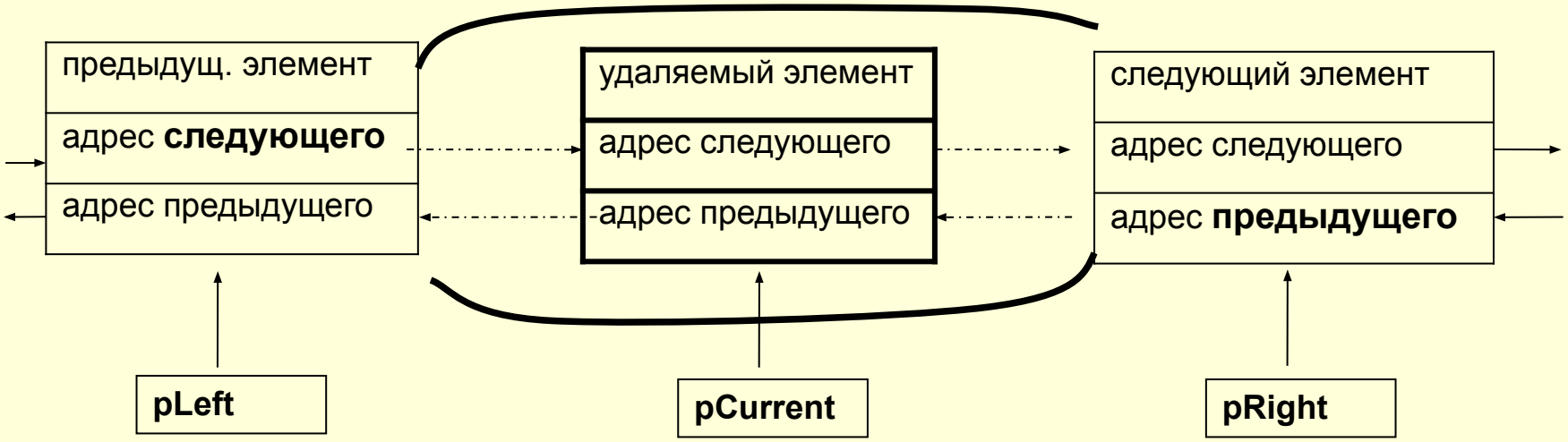



Схема удаления заданного элемента из списка