Лабораторная работа №3

По дисциплине «программирование и основы алгоритмизации»

Списки. Наследование

Цель лабораторной работы

Ознакомиться с реализацией базовых структур данных – односвязный и двусвязный список, на языке C++ с применением механизма наследования и абстрактных классов.

Задание

1. Разработать абстрактный класс Container
2. Унаследовать от него классы «Односвязный список» и «Двусвязный список».

Реализация списков должна позволять добавлять и удалять элементы из любого места в списке, получить его размер и содержимое по указанному индексу.

Краткие методические указания, примеры и теория

1. Односвязный список

1.1. Основные определения и структура

Односвязный список – динамическая структура данных, элементы которой содержат 2 поля: ***данные*** (что хранится в каждом «звене») и ***указатель на следующий элемент списка***. Нулевой элемент односвязного списка принято называть ***«Голова» (Head)***.

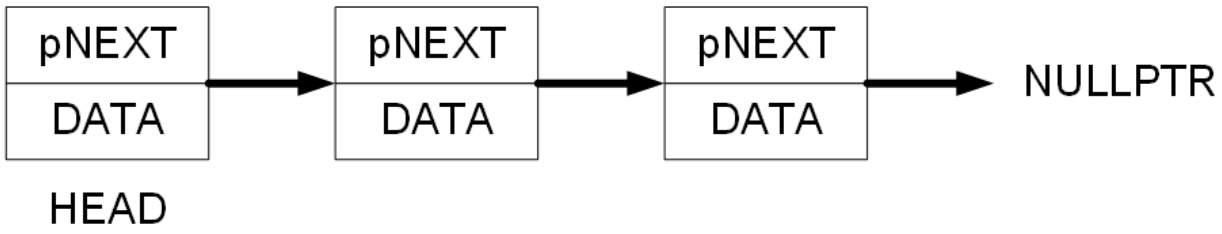

Рисунок 1 – Структура односвязного списка

Каждый отдельный элемент односвязного списка принято называть «Узел» или «Node». Последний элемент списка не указывает ни на что или, иными словами, является так называемым «пустым указателем». В языке C++ имеется зарезервированное ключевое слово **nullptr** для определения подобного указателя. Поле DATA непосредственно содержит хранимую в списке информацию и в данной лабораторной работе допускается использование

любого произвольного типа данных: как встроенных (int, double, std::string и т.д.), так и собственных (создание собственного класса).

1.2. Объявление класса односвязный список

Ниже представлено объявление класса односвязный список (List)

```
class List
{
public:
    List();
    ~List();

private:

    class Node
    {
    public:
        Node *pNext;
        int data;
        Node(int data, Node *pNext = nullptr)
        {
            this->data = data;
            this->pNext = pNext;
        }
    };
    Node *head;
    int Size;
};
```

Рисунок 2 – Листинг объявления класса List

На рисунке 2 в публичной секции объявлены конструктор и деструктор класса. В приватной секции объявлен класс Node, который описывает узел односвязного списка.

Примечание: подумать, какой модификатор доступа должен быть у класса Node, чтобы он был доступен в классе-наследнике.

В данном примере в качестве хранимого типа выбран простейший тип **int**. Также, в качестве полей указана «голова» списка (указатель на объект класса Node) и целочисленная переменная Size, которая будет хранить количество элементов в нашем списке (по сути, размер списка).

Обратите внимание, в объявлении класса на рисунке 2 отсутствуют прототипы методов добавления и удаления элементов, вывода отдельного элемента и размера списка.

1.3. Аспекты итерирования односвязного списка

Все действия с односвязным списком начинается с головы списка. Таким образом, чтобы добраться до элемента под номером N необходимо совершить N шагов по указателям pNext. Ниже представлена реализация метода получения значения в узле односвязного списка под индексом index

```
Тип_данных ПОЛУЧЕНИЕ_ДАННЫХ (const int index)
{
        Int counter=0;
        Создание указателя на объект узел с записью в него адреса головы списка; // например с именем «текущий»
        Пока (текущий != пустому указателю)
        {
            Если (counter == index)
            {
                Вернуть текущий->data;
            }
            Текущий = Текущий->pNext;
            Инкремент counter;
        }
}
```

В отличие от обычного массива (статического или динамического), элементы в списках расположены в памяти не последовательно друг за другом, а в разных участках памяти, но каждый узел в списке точно знает где расположен следующий. Данная структура хранения позволяет более быстро добавлять и удалять узлы из середины списка.

Дополнительное мини-задание: реализовать получение данных по индексу не методом, а с помощью перегрузки оператора [] (квадратные скобки).

1.4. Добавление элемента в односвязный список

Ниже представлена графическая интерпретация добавления элемента в произвольное место (по указанному индексу) в односвязный список

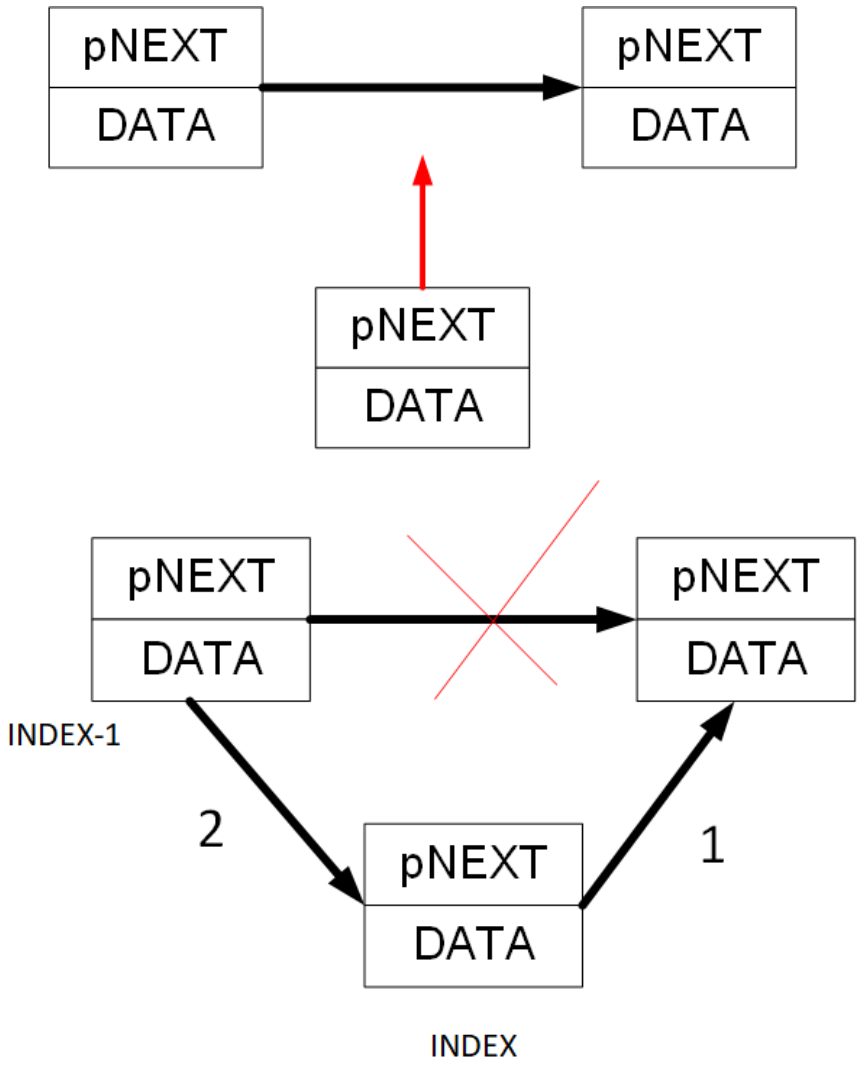

Рисунок 3 – Добавление элемента в односвязный список

Алгоритмически, для добавления узла в односвязный список по указанному адресу, необходимо «добраться» до элемента **INDEX-1**, затем создать новый объект типа «узел», у которого поле pNext будет равно полю pNext, но в **INDEX-1** узле, что обеспечит формирование связи под номером 1 на рисунке 3. Далее, у узла с индексом **INDEX-1** полю pNext присвоить адрес нового созданного узла. Последним шагом является инкремент поля Size, т.к. количество элементов (узлов) в списке было увеличено на 1.

1.5. Удаление элемента из односвязного списка

Ниже представлена графическая интерпретация удаления узла из односвязного списка по индексу INDEX.

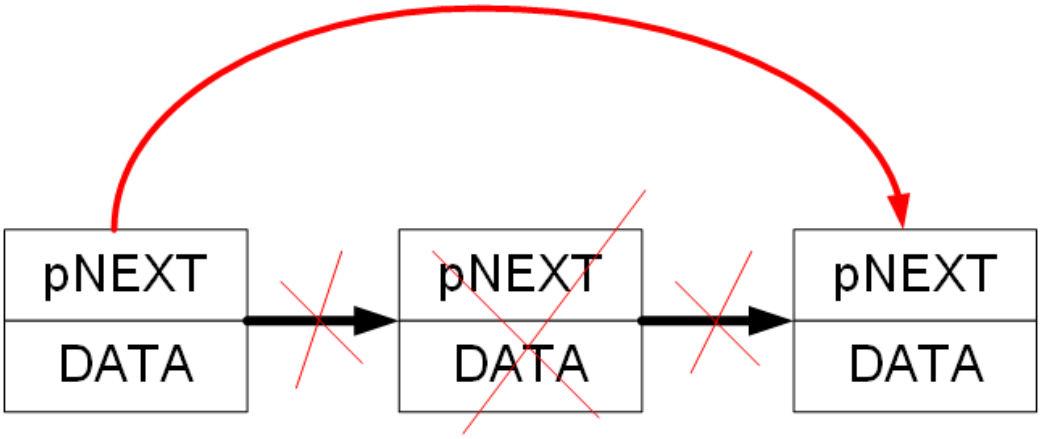

Рисунок 4 – Удаление элемента из односвязного списка

Алгоритмически, начало процедуры в точности совпадает с процедурой добавления – необходимо добраться до элемента INDEX-1. Затем, происходит создание буферного (временного) указателя на узел, который подлежит удалению. Далее, в поле pNext узла INDEX-1 записываем значение поля pNext узла, который подлежит удалению, что обеспечит переопределение связи узлов, как показано на рисунке 4. Наконец, буферный указатель удаляется с помощью delete, а поле Size декрементируется.

1.6. Получение размера списка

Обращаясь к рисунку 2 можно заметить, что поле Size, хранящее размер списка, т.е. количество его узлов находится под модификатором доступа private. Сделано это с той целью, чтобы не допустить произвольного изменения данной переменной, ее перезаписи, обнуления и т.д. Однако, размер списка, с другой стороны, должны быть всегда известны, и программист должен иметь возможность получения данного значения по запросу. Для этого, на языке C++, в частности, предусмотрены так называемые «Геттеры» (от англ. Get – получить) и «Сеттеры» (от англ. Set – установить). Геттеры и сеттеры – это специальные публичные методы класса, обеспечивающие доступ к приватным ключевым свойствам класса по определенному правилу. В данному случае, у поля Size будет только геттер, т.к. модификация данного поля осуществляется только при добавлении и удалении узлов, но не может быть задана программистом в явном виде. Псевдокод реализации данного метода представлен ниже

```
Тип_данных ПОЛУЧИТЬ_РАЗМЕР_СПИСКА ()
{
        ВЕРНУТЬ Size;
}
```

2. Двусвязный список

2.1. Общие сведения

Двусвязный список – структура данных, в которой каждый узел, помимо непосредственно самих данных, содержит указатель на следующий узел и на предыдущий узел. Для данной структуры, элемент типа «Узел» представлен на рисунке 5.

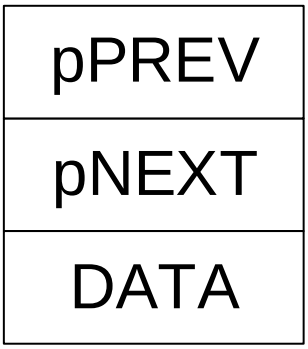

Рисунок 5 – Узел двусвязного списка

Общий вид двусвязного списка представлен на рисунке 6

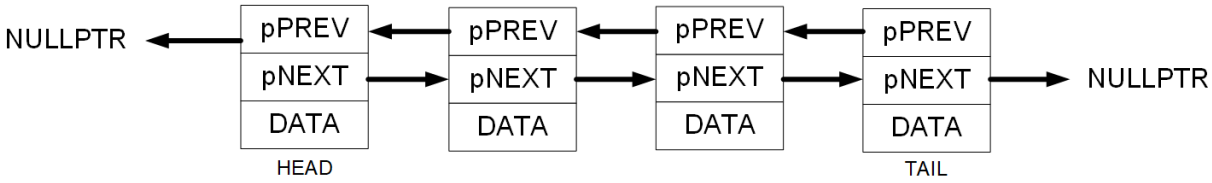

Рисунок 6 – Общий вид двусвязного списка

В данной структуре, по аналогии с односвязным списком, есть нулевой элемент под название «Голова» (Head), в котором указатель на предыдущий элемент pPrev равен nullptr, и элемент «Хвост», т.е. самый последний элемент в списке, у которого, в свою очередь, поле pNext равно nullptr.

2.2. Добавление элемента в двусвязный список

Алгоритм добавления узла в двусвязный список идентичен добавлению узла в односвязный список и представлен ниже.

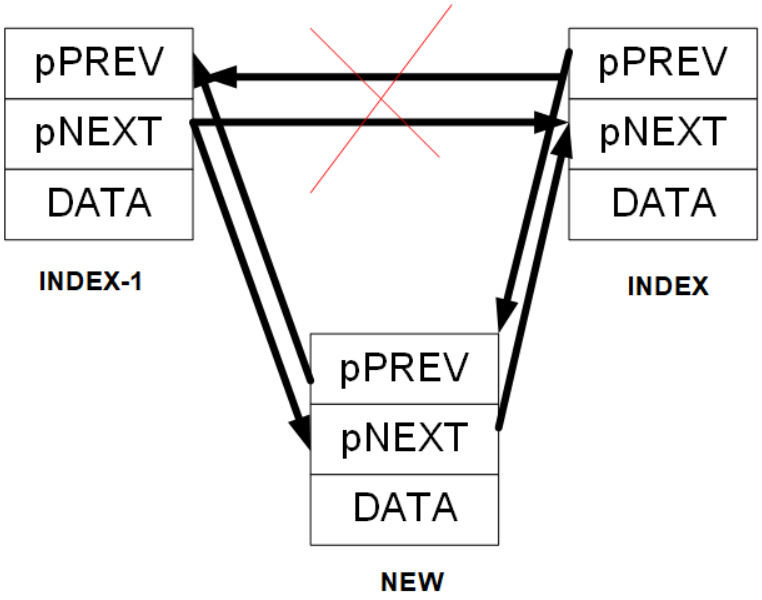

Рисунок 7 – Схема добавления узла в двусвязный список

Как и в односвязном списке, необходимо итеративно добраться от головы до элемента, предшествующего месту вставки (INDEX-1), создать указатель на новый узел (NEW) такой, чтобы его поле pPrev указывало на узел (INDEX-1). Полю pNext узла NEW присваиваем значение pNext узла (INDEX-1), у узла, INDEX полю pPREV присваиваем значение адреса узла NEW. Наконец, у узла INDEX-1 поле pNEXT должно указывать на узел NEW.

Описанная выше схема соответствует движению к узлу от головы двусвязного списка. Однако, ключевым преимуществом двусвязного списка по отношению к односвязному является возможность движения в обе стороны. Поэтому, возможна иная схема, а именно движение к нужному месту с хвоста списка. Двигаться к нужному индексу от головы или от конца мы выбираем исходя из его расположения, т.е. находится искомый индекс в первой половине списка или во второй.

Удаление элемента в двусвязном списке и получение элемента по нужному индексу подчиняется тем же правилам.