Пояснительная записка

Односвязные списки для меня непростая тема, но они будут нужны мне в будущем, скорее всего, поэтому я решил подойти к ним чуть иначе и написать реализацию используя структуру. Структуры мы пока не проходили, однако я знаком с ними по работе с C#, и в целом понимаю, как они работают и на C++. Поэтому односвязный список реализован как универсальная структура (она не привязана к типу данных) и имеет методы структуры, такие как:

```
void pop_front(); // удаление первого элемента
void pop_back();  // удаление послднего элемента
void push_back(T data);  // добавление новой ячейки к концу
void push_front(T data); // добавление новой ячейки к началу
void insert(T value, int index); // вставка в указанную ячейку без потери
void replacement(T value, int index);
void removeAt(int index); // удаление по указанному индексу
void print_list(); // печать списка
void clear(); // очистка списка

int GetSize() { return Size; }; // показать кол-во элементов в списке
```

Также для удобства обращения по индексу я написал перегрузку оператора:

```
/* Перегрузка [] */
T& operator[](const int index);
```

Еще важно сказать что я нумерую ячейки (при выводе в консоль) с 1ой чтобы пользователю было удобно их смотреть, и функции, связанные с четность/нечетность ячейки также ориентируются на их порядковый номер, а не системный индекс.

Иными словами ячейку номер 2 программа считает четной, хотя на самом деле эта ячейка имеет индекс 1.

Сам код (Visual Studio 2022)

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <string>
using namespace std;

template <typename T>
class List
{
    public:
        List();  // конструктор
        ~List(); // диструктор

        void pop_front(); // удаление первого элемента
        void pop_back();  // удаление послднего элемента
        void push_back(T data);  // добавление новой ячейки к концу
        void push_front(T data); // добавление новой ячейки к началу
        void insert(T value, int index); // вставка в указанную ячейку без потери
        void replacement(T value, int index);
        void removeAt(int index); // удаление по указанному индексу
        void print_list(); // печать списка
        void clear(); // очистка списка

        int GetSize() { return Size; }; // показать кол-во элементов в списке

        /* Перегрузка [] */
        T& operator[](const int index);

    private:
        template <typename T>
```

```
        class Node
        {
            public:
                Node* pNext;
                T data;

                Node(T data = T(), Node* pNext = nullptr)
                {
                    this->data = data;
                    this->pNext = pNext;
                }
        };

        int Size;
        Node<T> *head;
};

template <typename T>
List<T>::List()
{
    Size = 0;
    head = nullptr;
}

template <typename T>
List<T>::~List()
{
    /* В момент, когда компилятор видит, что
    переменная нигде больше не используется,
    автоматически запускается деструктор
    очищающий динамическую память и
    стирающий список*/

    clear();
}

template<typename T>
void List<T>::pop_front()
{
    Node<T> *temp = head;
    head = head->pNext;
    delete temp;
    Size--;
}

template<typename T>
void List<T>::pop_back()
{
    removeAt(Size - 1);
}

template<typename T>
void List<T>::push_back(T data)
{
    if (head == nullptr)
    {
        head = new Node<T>(data);
    }
    else
    {
        Node<T>* current = this->head;
        while (current->pNext != nullptr)
        {
            current = current->pNext;
        }
```

```
            current->pNext = new Node<T>(data);
        }

        Size++;
    }

    template<typename T>
    void List<T>::push_front(T data)
    {
        head = new Node<T>(data, head);
        Size++;
    }

    template<typename T>
    void List<T>::insert(T value, int index)
    {
        if (index == 0)
        {
            push_front(value);
        }
        else
        {
            Node<T>* previous = this->head;

            for (int i = 0; i < index - 1; i++)
            {
                previous = previous->pNext;
            }

            Node<T>* newNode = new Node<T>(value, previous->pNext);
            previous->pNext = newNode;

            Size++;
        }
    }

    template<typename T>
    void List<T>::replacement(T value, int index)
    {
        Node<T>* previous = this->head;
        if (index == 0)
        {
            previous->data = value;
        }
        else
        {
            for (int i = 0; i < index - 1; i++)
            {
                previous = previous->pNext;
            }

            previous->data = value;
        }
    }

    template<typename T>
    void List<T>::removeAt(int index)
    {
        if (index == 0)
        {
            pop_front();
        }
        else
        {
            Node<T>* previous = this->head;
```

```
        for (int i = 0; i < index - 1; i++)
        {
            previous = previous->pNext;
        }

        Node<T>* toDelete = previous->pNext;
        previous->pNext = toDelete->pNext;

        delete toDelete;
        Size--;
    }
}

template<typename T>
void List<T>::print_list()
{
    Node<T>* current = this->head;
    cout << " N   Data" << endl;
    for (int i = 0; i < GetSize(); i++)
    {
        if (i < 9)
        {
            cout << "0" + to_string(i + 1) << "    " << current->data << endl;
        }
        else
        {
            cout << i + 1 << "    " << current->data << endl;
        }
        current = current->pNext;
    }
    cout << endl;
}

template<typename T>
void List<T>::clear()
{
    while (Size)
    {
        pop_front();
    }
}

template<typename T>
T& List<T>::operator[](const int index)
{
    int counter = 0;
    Node<T>* current = this->head;

    while (current != nullptr)
    {
        if (counter == index)
        {
            return current->data;
        }
        current = current->pNext;
        counter++;
    }
}

/* Создает "пустой" (заполненный нулями) список */
void CreatingEmptyList(List<int> &lst, int N)
{
    for (int i = 0; i < N; i++)
    {
```

```
        lst.push_back(0);
    }
}

/* JoinList() копировать все элемент ы из первого списка
во второй список. первый список не изменился, второй добавился.
Второй список располагается в другом массиве.*/
void JoinList(List<int> &Long, List<int> &Short)
{
    for (int i = 0; i < Long.GetSize(); i++)
    {
        if (Long[i] != 0)
        {
            Short.push_back(Long[i]);
        }
    }
}

/* SplitList() – копировать первый список так, что элементы
с нечетными номерами позиций записать во второй список, а
с четными в третий. Второй и третий списки пустые. */
void SplitList(List<int>& Long, List<int>& ShortEvenNumbers, List<int>&
ShortOddNumbers)
{
    for (int i = 0; i < Long.GetSize(); i++)
    {
        if (Long[i] != 0)
        {
            if ((i + 1) % 2 == 0)
            {
                ShortEvenNumbers.push_back(Long[i]);
            }
            else
            {
                ShortOddNumbers.push_back(Long[i]);
            }
        }
    }
}

int main()
{
    setlocale(LC_ALL, "ru");
    srand(time(NULL));

    List<int> LongList;
    CreatingEmptyList(LongList, 50);
    cout << " Пустой список,\n заполненный нулями" << endl;
    LongList.print_list();


    /* Зададим случайные элементы */
    int ListNumberCell[15];

    for (int i = 0; i < 15; i++)
    {
        bool SearchON = true;
        bool Error = false;
        int numberCell;

        while (SearchON)
        {
            numberCell = rand() % 49 + 1;

            for (int i = 0; i < 14; i++)
```

```
            {
                if (ListNumberCell[i] == numberCell)
                {
                    Error = true;
                    break;
                }
            }
            if (Error)
            {
                SearchON = true;
                Error = false;
            }
            else
            {
                SearchON = false;
            }
        }
        ListNumberCell[i] = numberCell;
        int value = rand() % 99 + 100;
        LongList.replacement(value, numberCell);
    }

    /* Выведем заполненный список */
    cout << " Список, 15-ать ячеек которого\n заполнены случайным образом" << endl;
    LongList.print_list();

    List<int> ShortList;
    JoinList(LongList, ShortList);
    cout << " Результат работы JoinList()" << endl;
    ShortList.print_list();

    List<int> ShortEvenNumbers;
    List<int> ShortOddNumbers;
    SplitList(LongList, ShortEvenNumbers, ShortOddNumbers);
    cout << " Результат работы SplitList()" << endl;
    cout << " Числа стоящие на четных позициях" << endl;
    ShortEvenNumbers.print_list();
    cout << " Числа стоящие на нечетных позициях" << endl;
    ShortOddNumbers.print_list();


    /*
        InsertFirst() - включить элемент в голову списка.
        DeleteFirst() - удалить первый элемент из списка
        InsertLast() - включить элемент как последний
        DeleteLast() - удалить последний элемент

        Данные элементы я уже реализовал как метод струкуры,
        но назвал их по другому. Продемонстрирую их работу
        на списке ShortEvenNumbers т.к. он короткий и будет
        наглядно
    */

    cout << " Продемонстрирую работу:\n InsertFirst()\n DeleteFirst()\n
InsertLast()\n DeleteLast()\n на списке ShortEvenNumbers\n" << endl;

    cout << " Оригинальный ShortEvenNumbers: " << endl;
    ShortEvenNumbers.print_list();

    cout << " Результат работы InsertFirst()" << endl;
    ShortEvenNumbers.push_front(666);
    ShortEvenNumbers.print_list();

    cout << " Результат работы DeleteFirst()" << endl;
    ShortEvenNumbers.pop_front();
```

```
    ShortEvenNumbers.print_list();

    cout << " Результат работы InsertLast()" << endl;
    ShortEvenNumbers.push_back(999);
    ShortEvenNumbers.print_list();

    cout << " Результат работы DeleteLast()" << endl;
    ShortEvenNumbers.pop_back();
    ShortEvenNumbers.print_list();

    return 0;
}
```

Обзорно, что она выводит:

1. Выводим на экран пустой (заполненный нулями) список, чтобы убедиться, что он создался корректно:

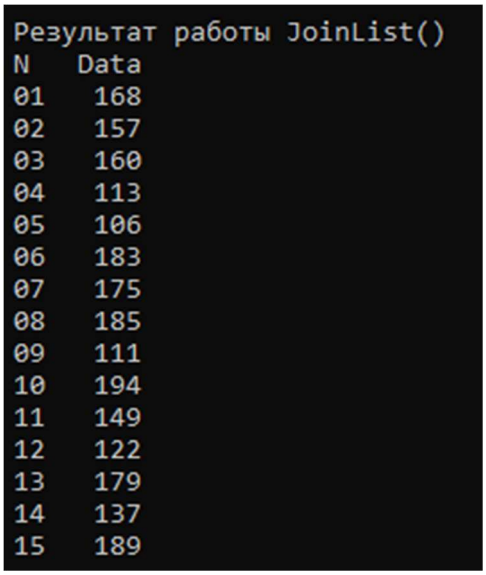
```
Пустой список,
заполненный нулями
N   Data
01   0
02   0
03   0
04   0
05   0
06   0
07   0
08   0
09   0
10   0
11   0
12   0
13   0
14   0
15   0
16   0
17   0
18   0
19   0
20   0
21   0
22   0
23   0
24   0
25   0
26   0
27   0
28   0
29   0
30   0
31   0
32   0
33   0
34   0
35   0
36   0
37   0
38   0
39   0
40   0
41   0
42   0
43   0
44   0
45   0
46   0
47   0
48   0
49   0
50   0
```

2. Вставляем в случайные ячейки (их всегда 15) случайные числа от 100 до 200

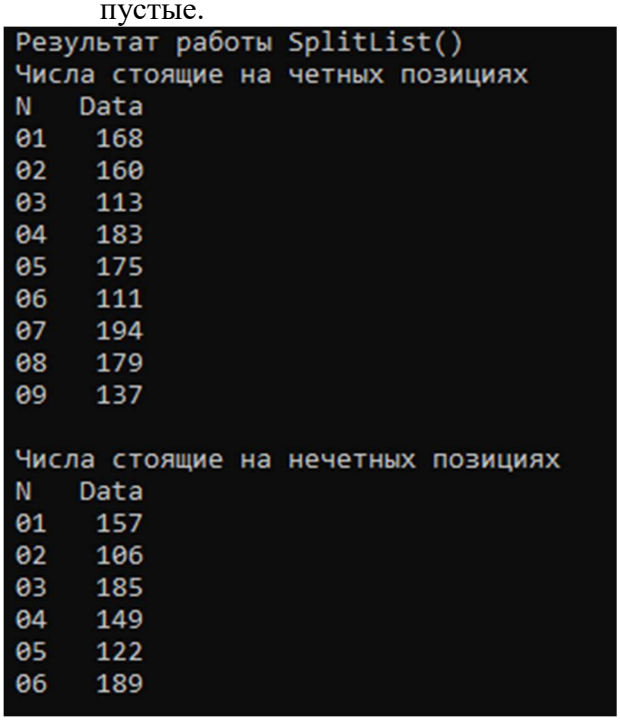
```
Список, 15-ать ячеек которого
заполнены случайным образом
N   Data
01   0
02   0
03   0
04   168
05   0
06   0
07   157
08   160
09   0
10   113
11   0
12   0
13   106
14   183
15   0
16   0
17   0
18   0
19   0
20   0
21   0
22   0
23   0
24   175
25   0
26   0
27   185
28   0
29   0
30   111
31   0
32   194
33   149
34   0
35   0
36   0
37   122
38   0
39   0
40   0
41   0
42   179
43   0
44   137
45   0
46   0
47   189
48   0
49   0
50   0
```

Далее демонстрируем результат работы наших функций

1. JoinList() - копировать все элементы из первого списка по второй список. первый список не изменился, второй добавился. Второй список располагается в другом массиве.



2. SplitList() - копировать первый список так, что элементы с нечетными номерами позиций записать во второй список, а с четными в третий. Второй и третий списки пустые.



3. Продемонстрирую работу: InsertFirst(), DeleteFirst(), InsertLast(), DeleteLast()

```
Продемонстрирую работу:
InsertFirst()
DeleteFirst()
InsertLast()
DeleteLast()
на списке ShortEvenNumbers

Оригинальный ShortEvenNumbers:
N   Data
01   168
02   160
03   113
04   183
05   175
06   111
07   194
08   179
09   137

Результат работы InsertFirst()
N   Data
01   666
02   168
03   160
04   113
05   183
06   175
07   111
08   194
09   179
10   137

Результат работы DeleteFirst()
N   Data
01   168
02   160
03   113
04   183
05   175
06   111
07   194
08   179
09   137
```

```
Результат работы InsertLast()
N   Data
01   168
02   160
03   113
04   183
05   175
06   111
07   194
08   179
09   137
10   999

Результат работы DeleteLast()
N   Data
01   168
02   160
03   113
04   183
05   175
06   111
07   194
08   179
09   137
```

Программа «перетирает» каждую анкету, т.е. вопросы второй анкеты появляются на месте первых, а не снизу. Программа умеет делать список ответивших и выводит результаты анализа.

```
 Введите ваш возраст (цифра от 18 до 99)
 45
 Далее для выбора ответа вводите цифры
 Ваш пол?
 1. Я мужчина
 2. Я женщина
 >> 1
Ваше образование?
 1. Начальное
 2. Среднее
 3. Высшее
 >> 3
Ваш выбор?!
 1. ДА!
 2. НЕТ!
 >> 1
Продолжить анкетирование?
 1. Да
 2. Нет
 >>
```

```
 N   Age  M/W     Education  Question
01   45   Мужчина Высшее     Да
02   29   Женщина Среднее    Нет
03   18   Мужчина Начальное  Да
04   78   Женщина Высшее     Да


мужчин старше 40 лет, имеющих высшее образование, ответили ДА на вопрос анкеты -> 1
женщин моложе 30 лет, имеющих среднее образование, ответили НЕТ на вопрос анкеты -> 1
мужчин моложе 25 лет, имеющих начальное образование, ответили ДА на вопрос анкеты -> 1
```

```cpp
#include <iostream>
#include <stdlib.h>
#include <fstream>
#include <windows.h>
#include <time.h>
#include <string>
using namespace std;

template <typename T>
class List
{
public:
    List();  // конструктор
    ~List(); // диструктор

    void pop_front(); // удаление первого элемента
    void pop_back();  // удаление послднего элемента
    void push_back(T age = T(), T manwoman = T(), T education = T(), T question =
T());  // добавление новой ячейки к концу
    void push_front(T age = T(), T manwoman = T(), T education = T(), T question =
T()); // добавление новой ячейки к началу
    void removeAt(int index); // удаление по указанному индексу
    void print_list(); // печать списка
    void clear(); // очистка списка

    int GetSize() { return Size; }; // показать кол-во элементов в списке

private:
    template <typename T>
    class Node
    {
    public:
        Node* pNext;
        T age;
        T manwoman;
        T education;
        T question;


        Node(T age = T(), T manwoman = T(), T education = T(), T question = T(),
Node* pNext = nullptr)
        {
            this->age = age;
            this->manwoman = manwoman;
            this->education = education;
            this->question = question;
            this->pNext = pNext;
        }
    };

    int Size;
    Node<T>* head;
};

template <typename T>
List<T>::List()
{
    Size = 0;
    head = nullptr;
}
```

```cpp
template <typename T>
List<T>::~List()
{
    /* В момент когда компилятор видит что
    переменная нигде больше не используется,
    автоматически запускается диструктор
    очищающий динамическую память и
    стирающий список*/

    clear();
}

template<typename T>
void List<T>::pop_front()
{
    Node<T>* temp = head;
    head = head->pNext;
    delete temp;
    Size--;
}

template<typename T>
void List<T>::pop_back()
{
    removeAt(Size - 1);
}

template<typename T>
void List<T>::push_back(T age, T manwoman, T education, T question)
{
    if (head == nullptr)
    {
        head = new Node<T>(age, manwoman, education, question);
    }
    else
    {
        Node<T>* current = this->head;
        while (current->pNext != nullptr)
        {
            current = current->pNext;
        }
        current->pNext = new Node<T>(age, manwoman, education, question);
    }

    Size++;
}

template<typename T>
void List<T>::push_front(T age, T manwoman, T education, T question)
{
    head = new Node<T>(age, manwoman, education, question, head);
    Size++;
}

template<typename T>
void List<T>::removeAt(int index)
{
    if (index == 0)
    {
        pop_front();
    }
    else
    {
        Node<T>* previous = this->head;
```

```
        for (int i = 0; i < index - 1; i++)
        {
            previous = previous->pNext;
        }

        Node<T>* toDelete = previous->pNext;
        previous->pNext = toDelete->pNext;

        delete toDelete;
        Size--;
    }
}

template<typename T>
void List<T>::print_list()
{
    Node<T>* current = this->head;
    cout << " N   Age  M/W     Education  Question" << endl;
    for (int i = 0; i < GetSize(); i++)
    {
        if (i < 9)
        {
            cout << "0" + to_string(i + 1) << "    " << current->age << "    " <<
current->manwoman << " " << current->education << "  " << current->question << endl;
        }
        else
        {
            cout << i + 1 << "    " << current->age << " " << current->manwoman << "
" << current->education << "  " << current->question << endl;
        }
        current = current->pNext;
    }
    cout << endl;
}

template<typename T>
void List<T>::clear()
{
    while (Size)
    {
        pop_front();
    }
}

void Clear_screen()
{
    HANDLE Console = GetStdHandle(STD_OUTPUT_HANDLE);

    SetConsoleCursorPosition(Console, { 0, 0 });
    printf("% 80s\n", " ");    printf("% 80s\n", " ");
    printf("% 80s\n", " ");    printf("% 80s\n", " ");
    printf("% 80s\n", " ");    printf("% 80s\n", " ");
    printf("% 80s\n", " ");    printf("% 80s\n", " ");
    printf("% 80s\n", " ");    printf("% 80s\n", " ");
    printf("% 80s\n", " ");    printf("% 80s\n", " ");
    printf("% 80s\n", " ");    printf("% 80s\n", " ");
    printf("% 80s\n", " ");    printf("% 80s\n", " ");
    printf("% 80s\n", " ");    printf("% 80s\n", " ");
    printf("% 80s\n", " ");    printf("% 80s\n", " ");
}



int main()
```

```
{
    setlocale(LC_ALL, "ru");
    srand(time(NULL));
    HANDLE Console = GetStdHandle(STD_OUTPUT_HANDLE);

    List<string> LongList;

    bool Go = true;

    int ConditionCounter1 = 0;
    int ConditionCounter2 = 0;
    int ConditionCounter3 = 0;

    while (Go)
    {
        SetConsoleTextAttribute(Console, 12);
        SetConsoleCursorPosition(Console, { 0, 0 });

        cout << " Введите ваш возраст (цифра от 18 до 99)" << endl << " ";
        int AgeInt;
        cin >> AgeInt;
        string age = to_string(AgeInt);

        cout << " Далее для выбора ответа вводите цифры " << endl;

        cout << " Ваш пол?" << endl << " 1. Я мужчина" << endl << " 2. Я женщина" <<
endl << " >> ";
        string manwoman;
        int answer; cin >> answer;
        if (answer == 1)
        {
            manwoman = "Мужчина";
        }
        else
        {
            manwoman = "Женщина";
        }

        cout << "Ваше образование?" << endl << " 1. Начальное" << endl << " 2.
Среднее" << endl << " 3. Высшее" << endl << " >> ";
        cin >> answer;
        string education;
        if (answer == 1)
        {
            education = "Начальное";
        }
        else if (answer == 2)
        {
            education = "Среднее  ";
        }
        else
        {
            education = "Высшее   ";
        }

        cout << "Ваш выбор?!" << endl << " 1. ДА!" << endl << " 2. НЕТ!" << endl <<
" >> ";
        cin >> answer;
        string question;
        if (answer == 1)
        {
            question = "Да";
        }
        else
        {
```

```
            question = "Нет";
        }

        if (AgeInt > 40 and manwoman == "Мужчина" and education == "Высшее   " and
question == "Да")
        {
            ConditionCounter1++;
        }
        else if (AgeInt < 30 and manwoman == "Женщина" and education == "Среднее  "
and question == "Нет")
        {
            ConditionCounter2++;
        }
        else if (AgeInt < 25 and manwoman == "Мужчина" and education == "Начальное"
and question == "Да")
        {
            ConditionCounter3++;
        }

        LongList.push_back(age, manwoman, education, question);

        cout << "Продолжить анкетирование? " << endl << " 1. Да" << endl << " 2.
Нет" << endl << " >> ";
        cin >> answer;
        if (answer == 2)
        {
            Go = false;
        }

        Clear_screen();
    }

    // age manwoman education question
    SetConsoleTextAttribute(Console, 12);
    SetConsoleCursorPosition(Console, { 0, 0 });
    LongList.print_list();

    cout << endl << ConditionCounter1 << endl << ConditionCounter2 << endl <<
ConditionCounter3;
}
```