Лабораторная работа н.1

Односвязный список

Петров Артем Евгеньевич

Содержание

Список иллюстраций

1 Цель работы

Целью данной работы служит идея научится работать с классами и структурами в банальнейшем их представлении-односвязном списке, дополненном различными функциями: добавление элемента в начало списка, в конец, перед выбранным элементов, после выбранного элемента, удаление узла по его имени и печать всего списка.

2 Задание

Написать компьютерную программу, содержащую

- Описание структуры, содержащей поля типа string, int, double;
- Набор функций для работы со списком на базе этой структуры:
  - Добавление элемента в начало списка;
  - Добавление элемента в конец списка;
  - Добавление элемента в список после заданного элемента;
  - Добавление элемента в список перед заданным элементом;
  - Удаление из списка элемента с заданным именем;
  - Вывод содержания списка на экран;
- Функцию main, содержащую сценарий работы со списком, использующий разработанный инструментарий.

3 Выполнение лабораторной работы

ОТЧЕТ ПОЛНОСТЬЮ НАПИСАН НА ЯЗЫКЕ РАЗМЕТКИ MARKDOWN!!!

3.1 Задание 1. Создание структуры списка.

1. При создании односвязного списка я руководствовался следующей идей:

Создать две структуры-структуру Звена(Node, далее Thing) и структуру списка, чтобы в дальнейшем структура Списка, со всеми добавлениями, оставалсь полной до завершения программы. Кроме того, структура самого бы списка содержала все методы класса(или структуры) для достижения полной автономности внутри себя, что позволило бы использовать обе структуры в любой другой функции(помимо main), любом другом классе и т.д. Таким образом, программа становилась бы невероятно гибкой для дополнения, обновления, внедрени и т.д. В итоге, я так же смог добиться небольшого пользовательского интерфейса, который при пользовании обывателем не требует знания языка программирования.

2. Приступим же к описанию структур:

    1. Структура звена, которое в нашем случае является единицей товара в магазине или буфете. Следовательно, звено должно содержать сведения о своем названии, цене, количестве в магазине, на складе и т.п. Однако, т.к. все реализуется на примере списка, надо так же добавить указатель на адрес в памяти на следующий элемент. Выглядеть это будет так:

```
struct Thing
{
    string name; //имя
    float price; //цена(значение с плавающей запятой при надобности)
    int amount; //количество
    Thing* next; // указатель на адрес след. элемента
};
```

2. Структура списка элементов, которая будет содержать методы, указанные в задании, и которая обязана содержать указатель на первый(начальный) элемент списка(англ. head). В начале она будет выглядеть так:

```
struct List
{
    Thing* head=NULL;// указатель на первый элемент(по счету)
/* тут будут различные методы структуры */
};
```

3.2 Задание 2. Создание набора функций на базе структуры Thing в List

1. СОЗДАНИЕ НЕЗАВИСИМОГО ЗВЕНА.Для начала, стоит сказать, что создание самого элемента будет нужно нам минимум для 4 функций из заданных 6, поэтому если добавлять пользовательский интерфейс в терминал(или имитировать его), то огромного количества команд ввода и считываний значений не избежать, если не сделать создание звена без заведомого включения в список. Поэтому(внути структуры List):

```
Thing* createNode()//обязательно делаем функцию того же типа, что и звено,
```

```
//чтобы в будущем возвращать адрес новосозданной ячейки. Указатель нужен для того
// плодить сущности, ведь по сути ниже tovar является указателем типа Thing, кото
//получает адрес памяти, зарезервированной командой new под тип Звена(Thing).
{
    //ввод названия от пользователя
    cout<< "---------------------------------------" << endl;
    cout << "\nVvedite nazvanie novogo producta: ";
    string nazvanie;
    cin >> nazvanie;

    //Ввод цены для нового продукта
    cout << "\nVvedite tsenu novogo producta: ";
    float tsena;
    cin >> tsena;

    //Ввод количества для нового пользователя
    cout << "\nVvedite kolichestvo novogo tovara: ";
    int kolichestvo;
    cin >> kolichestvo;

    Thing *tovar=new Thing; //Выделение памяти под тип данных Thing(т.е.
    // под все его параметры)
    // Теперь адресам в памяти нового звена присвоим введенные значения
    // при помощи указателя нового звена на его параметры
    tovar->amount=kolichestvo;
    tovar->name=nazvanie;
    tovar->price=tsena;
    tovar->next=NULL; //зануляем указатель на след. элемент, чтобы
    //избежать непредвиденных значений
```

```
    return tovar; //возвращаем эту ячейку, чтобы в будущем эта функция
    // передавала значение в другие функции
}
```

Теперь приступим к написанию заданных функций.

2. ДОБАВЛЕНИЕ ЭЛЕМЕНТА В НАЧАЛО СПИСКА. Будем рассматривать два случая: Головного элемента нет и он уже есть. В случае, если его нет, мы присваиваем значению head из поля списка значение нового звена, созданного функцией createNode. В ином же случае, мы сначала создаем временную переменную(TMP), которой присваиваем значение прошлого головного элемента, потом присваем переменной головного элемента новое звено, а адресом следующего элемента послужит уже переменная TMP. Т.е. head->next=TMP. Выглядеть будет так:

```
void PushToHead(Thing* node) // Передаем в функцию при вызове значение нового зве
{
    Thing* TMP=new Thing;
    TMP=head;
    if(head==NULL)
    {
        head=node;
        head->next=NULL;
    }
    else
    {
        head=node;
        head->next=TMP;
    }
}
```

3. ДОБАВЛЕНИЕ ЭЛЕМЕНТА В КОНЕЦ СПИСКА. Рассмотрим три случая: Если головной элемент отсутствует, если отсутствует элемент следующий за головным элементом и если ни одно из этих условий не выполняет. В первом случае мы присваем головному элементу наш новый элемент. Во втором мы указателю головного элемента на следующий элемент присваем адрес нового элемент. В третьем же мы опять создаем переменную tmp3, которой присваем головной элемент и циклом прокручиваем нашу переменную, пошагово присваивая ей ее же указатель на следующий элемент, пока указатель переменной не будет указывать на NULL, потом же присваиваем указателю переменной на след. элемент наше новое звено.

```
void PushToTail(Thing* node)
{
    if(head==NULL)//Первый случай
    {
        head=node;
        node->next=NULL;
    }
    else if (head->next == NULL)//Второй случай
    {
        head->next=node;
        node->next=NULL;
    }
    else // Третий случай
    {
        Thing* tmp3=head;
        while (tmp3->next != NULL)
        {
            tmp3=tmp3->next;
        }
```

```
        tmp3->next=node;
    }
}
```

4. ДОБАВЛЕНИЕ ЭЛЕМЕНТ ПЕРЕД ЗАДАННЫМ. В этом случае мы создадим две переменные структуры звена-предыдущий элемент и ведущий, чтобы вклинить между ними элемент.

```
int AddBefore(Thing* node, unsigned int BeforeWhatElem) /*интовое значение, котор
{
    Thing* tmp4=new Thing;
    Thing* prev=new Thing;
    prev = NULL;
    tmp4=head;
    unsigned int counter1=0; /*счетчик, который в цикле будет инкрементироваться,
    сравняется с номером того элемента, перед которым надо добавить.

    if (BeforeWhatElem == 0) //случай, если надо добавить перед головным, сделав
    //головным
    {
        head=node;
        head->next=tmp4;
        return 0;
    }

    while (counter1 != BeforeWhatElem) //цикл, итерирующийся, пока текущим элемен
    //станет тот, перед которым надо вставить звено
    {
        if (tmp4 == NULL) //если цикл прокрутился до значения текущего эл NULL, т
        {
            cout << "no such elem";
```

```
            return 0;
        }
        prev=tmp4; //сохраняем предыдущий текущий эл от прошлой итерации, чтобы в
        //между ними без потери предыдущего
        tmp4=tmp4->next;
        counter1++;
    }
    prev->next=node; //делаем указателем на следующий эл адрес нового эл и перебр
    //указатель нового эл на след. на текущий(тот перед которым вставляем)
    node->next=tmp4;
    return 0;
    }
```

5. ДОБАВЛЕНИЕ ЭЛЕМЕНТА ПОСЛЕ ЗАДАННОГО. На самом деле, мы повторяем предыдущую функцию, но цикл проделывает на шаг больше, делай уже тот элемент, после которого надо добавить, предыдущим(для цикла)

```
int AddAfter(Thing* node, int AddAfterWhat)
    {
    Thing* tmp5=new Thing;
    Thing* prev1=new Thing;
    prev1 = NULL;
    tmp5=head;
    unsigned int counter1=0;

    if (AddAfterWhat == 0)
    {
        head=node;
        head->next=tmp5;
        return 0;
    }
```

```
    while (counter1 <= AddAfterWhat)
    {
        if (tmp5 == NULL)
        {
            cout << "no such elem";
            return 0;
        }
        prev1=tmp5;
        tmp5=tmp5->next;
        counter1++;
    }
    prev1->next=node;
    node->next=tmp5;
    return 0;
    }
```

6. УДАЛЕНИЕ ЭЛЕМЕНТА ИЗ СПИСКА С ЗАДАННЫМ ИМЕНЕМ. В данном случае мы повторим немного две предыдущие функции, т.к. нам придется связать два элемента между собой после удаления одного.

```
int deleteNode()
{
    cout << "\nVvedite nazvanie tovara, kotoriy nado udalit': ";
    string imya;
    cin >> imya;

    Thing* prev=new Thing;
    prev=NULL;
    Thing* tmp3=new Thing;
    tmp3=head;
```

```
    if (tmp3->name == imya) //если головной элемент является нужным
    {
        delete tmp3;
        head=head->next; //следующий за головным становится головным
        return 0;
    }

    while (tmp3->name != imya)
    {
        prev=tmp3;
        if (tmp3 == NULL) //если прошли весь список, не найдя элемента
        {
            cout << "\nNet takogo elementa";
            return 0;
        }
        tmp3=tmp3->next;
    }
    tmp3=tmp3->next; //делаем еще один шаг по списку, а пред. остает на два сзади
    prev->next=tmp3; //привязываем предыдущий к тому, что на два шага впереди.
    //возможно, я захламляю память, но я иначе создавал бы очередную бы переменну
    return 0;
    }
```

7. ПЕЧАТЬ ВСЕГО СПИСКА

```
void printAll()
    {
    Thing* tmp=new Thing; //создаем переменную для прошагивания списка циклом
    tmp=head; //копируем в нее голову.
    unsigned int counter=0; //очевидно, счетчик
```

```
    cout << setiosflags(ios::left) << setw(5) << "Num" << setw(10) << "Name" << s
    while(tmp != NULL) //печать циклом, пока текущий элемент не выйдет за предель
    {
        cout << setiosflags(ios::left) << setw(5) << counter << setw(10) << tmp->
        tmp=tmp->next;
        counter++;
    }
    delete tmp; //удаляем элемент
    }
```

8. ИМИТИРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА. Выполним ее оператором ветвления switch. Всего у нас 6 функций-методов списка, добавим еще выход, полное завершение программы. На самом деле, можно спокойно обойтись и без нее, но создавать новые списки придется с помощью программного кода. Кроме того, используется функция прочтения символа getch() (conio.h) и реурсия для повторения выбора функции. Позже я приложу два варианта, когда будет рассматриваться функция main.

```
int ViborFunktsii()
{
    cout << "Spisok funktsiy:\n" << "1. Create & Push to head\n" << "2. Create &
    int choice;
    cin >> choice;
    switch (choice)
    {
    case 1:
        PushToHead(createNode());
        cout << "\nContinue?(y/n):";
        if (getch() == 'y')
        {
            ViborFunktsii();
```

```
            break;
        }
        else
        {
            break;
        }
    case 2:
        PushToTail(createNode());
        cout << "Continue?(y/n):";
        if (getch() == 'y')
        {
            ViborFunktsii();
            break;
        }
        else
        {
            break;
        }
    case 3:
        printAll();
        cout << "Continue?(y/n):";
        if (getch() == 'y')
        {
            ViborFunktsii();
            break;
        }
        else
        {
            break;
```

```
    }
case 4:
    deleteNode();
    cout << "Continue?(y/n):";
    if (getch() == 'y')
    {
        ViborFunktsii();
        break;
    }
    else
    {
        break;
    }
case 5:
    cout << "Vvedite poryadkoviy nomer:";
    int a;
    cin >> a;
    AddAfter(createNode(), a);
    cout << "Continue?(y/n):";
    if (getch() == 'y')
    {
        ViborFunktsii();
        break;
    }
    else
    {
        break;
    }
case 6:
```

```
            cout << "Vvedite poryadkoviy nomer:";
            int b;
            cin >> b;
            AddBefore(createNode(), b);
            cout << "Continue?(y/n):";
            if (getch() == 'y')
            {
                ViborFunktsii();
                break;
            }
            else
            {
                break;
            }
        case 7:
            break;
        default:
            cout << "Net takogo varianta!!!";
            ViborFunktsii();
            break;
        }
}
```

- Полный код будет в приложенном файле!!! Не забудьте, что все вышеописанные функции являются методом структуры list!!!

3.3 Задание 3. Функция main, содержащая сценарий работы со списком, использующая разработанный инструментарий.

1. Вариант

```
int main()
{
    List* lst=new List;
    lst->PushToHead(lst->createNode());
    lst->PushToHead(lst->createNode());
    lst->PushToTail(lst->createNode()); //В аргументах передаем метод создания зв
    //В данном случае мы вызываем функцию через указатель на метод экземпляра
    //класса(ПО АДРЕСУ)
    lst->printAll();
    lst->deleteNode();
    lst->AddBefore(lst->createNode(), 0); //добавим перед первым(счет с 0 начинае
    lst->printAll();
    lst->AddAfter(lst->createNode(), 1);//добавим после второго элемента
    lst->printAll();
    delete lst;
    return 0;
}
```

2. Вариант(с использованием функции н.8, иначе ее можно просто закомментировать и вернуться к первому варианту)

```
int main()
{
    cout << "\nDvusvyazniy spisok by Petrov Artem\n";
```

```
    List* lst=new List; //создаем экземпляр структуры(класса)
    lst->ViborFunktsii(); //указатель на адрес метода внутри структуры(класса)
    delete lst; //удаляем, чтобы не захламлять память, я бы мог все привести к по
    //интерфейсу с сохранением данного списка, но появилось много других дел)
    return 0;
}
```

- Пример выполнения функции(в двух вариантах):

1. См. рис. 3.1

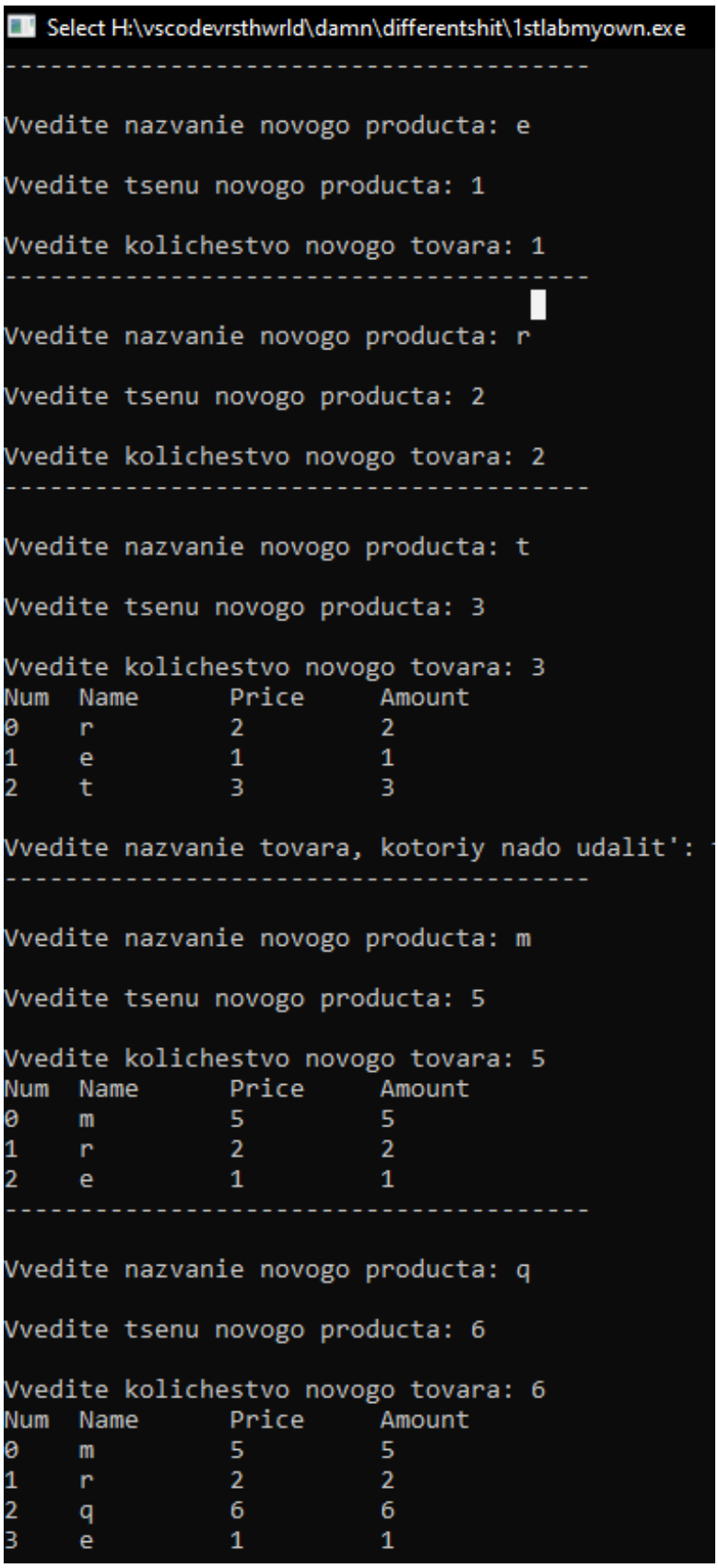

Рис. 3.1: 1.Результат выполнения первого варианта кода

2. См. рис. 3.2-3.4

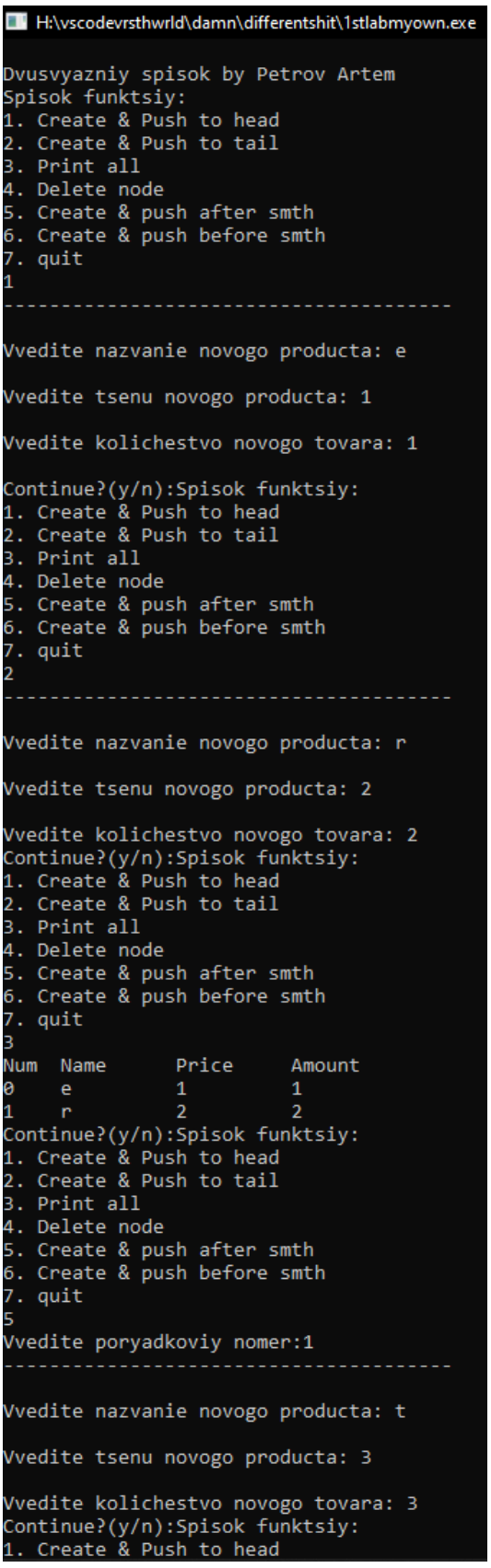

Рис. 3.2: Результат выполнения второго варианта кода

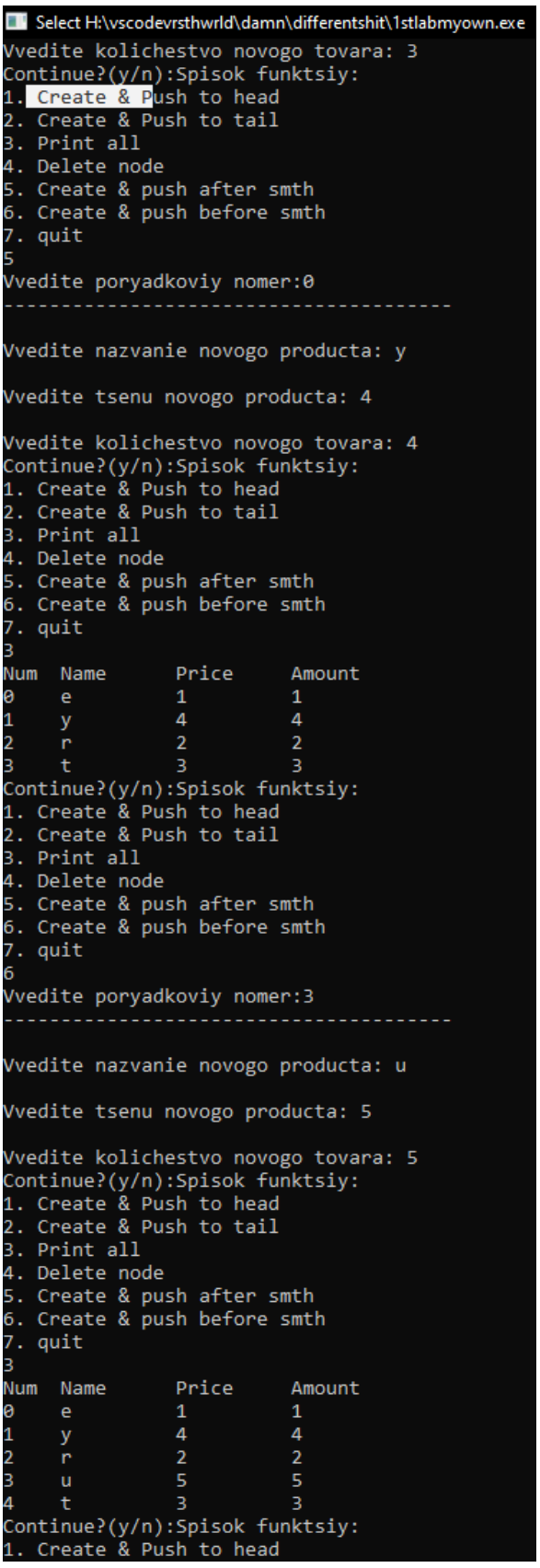

Рис. 3.3: Результат выполнения второго варианта кода

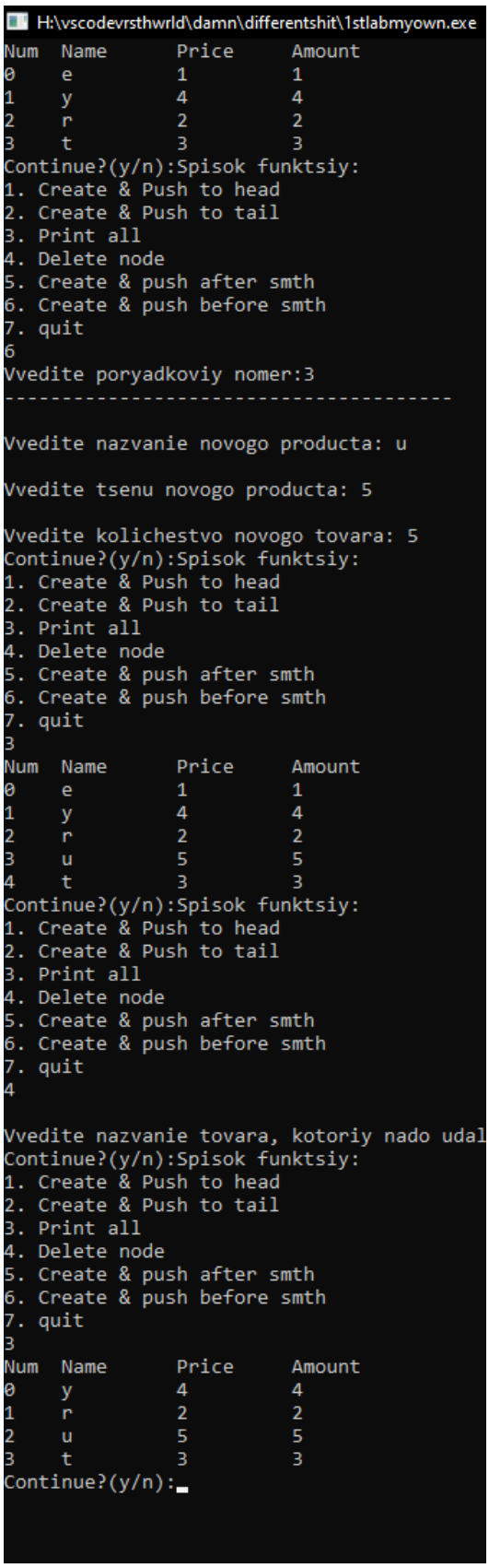

Рис. 3.4: Результат выполнения второго варианта кода

4 Выводы

В ходе данной лабораторной работы я научился работать с структурами, особенно с списками, т.к. двусвязный представляет упрощенный вариант. Кроме того, я попрактиковал понимание адресного пространства, методов структур, указателей и т.д. Так же я написал программу односвязного списка, который при небольшой корректировке мог бы быть использован с пользой.