Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

Колледж ВятГУ

**ОТЧЕТ**

**ПО ДОМАШНЕЙ КОНТРОЛЬНОЙ РАБОТЕ №3**

**«ОСНОВЫ СОБЫТИЙНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ»**

**ПО «МДК 05.02 Разработка кода информационных систем»**

Выполнил: Ворсин Егор Александрович

ИСПк-207-52-00

Преподаватель:

Сергеева Елизавета Григорьевна

Киров

2024

**Цель работы:** получение базовых навыков реализации приложений с графическим интерфейсом пользователя на основе событийно-ориентированной парадигмы.

**Задание:**

Вариант: 6.

1. Реализовать приложение «Секундомер».
2. Предусмотреть возможность временной остановки отсчёта и полного сброса.

**Описание алгоритма:**

1) В программе даны две структуры данных: динамическая и статическая.

Динамическая структура данных – это структура данных, которая может изменять размер во время выполнения программы. Это означает, что элементы могут быть добавлены или удалены из структуры данных в процессе выполнения программы, а размер структуры данных будет автоматически управляться.

Статическая структура данных – это структура данных, размер которой определяется заранее и остается неизменным в процессе выполнения программы. Элементы статической структуры данных могут быть изменены, но размер структуры остается постоянным.

2) Каждый элемент массива имеет свой индекс, который определяет его положение в массиве.

Для хранения информации с помощью массива, можно создать массив определенного размера и заполнить его элементами данных. Например, массив может хранить информацию о студентах в классе, где каждый элемент массива представляет собой данные об определенном студенте, такие как его имя, возраст, номер студенческого билета и т.д.

Узлы массива имеют индексы, начиная с 0. Например, в массиве размером 5 элементов, первый элемент будет иметь индекс 0, второй - индекс 1 и т.д.

Связь между узлами в массиве осуществляется через их индексы. Например, чтобы получить доступ к определенному элементу массива, нужно обратиться к его индексу. Связь между узлами в массиве является прямой и определяется порядком их расположения в массиве.

3) Для добавления элемента в структуру данных, необходимо определить место, куда нужно добавить элемент. Например, для добавления элемента в конец списка или массива, можно использовать функцию append() или методы добавления в соответствующих языках программирования. Для добавление элемента в дерево или граф, нужно выполнить соответствующие операции вставки, которые обеспечат корректное соответствие структуры данных

Для доступа к элементу в структуре данных, нужно обратиться к его индексу или ключу, в зависимости от типа структуры данных. Например, для доступа к элементу в массиве можно использовать индексацию, для взятия элемента из списка - методы доступа к элементам списка и т.д.

Статический тип данных хранится в массиве, где каждый элемент содержит информацию о значении и ссылается на следующий элемент.

Динамический тип данных хранится таким образом, что при нужде хранения информации создается новый динамический выделенный узел, т.е. память, которая содержит значения.

Для поиска элемента в структуре данных, используют различные алгоритмы в зависимости от типа структуры данных. Например, для поиска в списке или массиве можно применить линейный поиск или бинарный поиск, для поиска в дереве - методы обхода дерева (например, обход в глубину или в ширину).

Для вывода структуры данных на экран, можно использовать циклы и функции вывода в консоль или другой интерфейс. Например, для вывода элементов массива или списка на экран, можно использовать цикл for или while и выводить каждый элемент поочередно. Для вывода деревьев или графов можно использовать различные методы обхода структуры данных и печатать элементы по мере обхода.

4) Организация case-меню обычно выглядит следующим образом:

1. Задается переменная или выражение, значение которой будет проверяться.

2. В зависимости от значения переменной или выражения, программа выполняет определенный блок кода. Для каждого возможного значения переменной задается блок кода, который должен быть выполнен.

3. Case-меню заканчивается оператором break, который указывает, что выполнение блока кода должно прекратиться.

1. Определение структуры данных Node, которая содержит информацию о данных и указатель на следующий элемент.

2. Объявление глобальных переменных: массив узлов nodes, указатель на голову списка head и индекс свободного элемента freeIndex.

3. Процедура InitializeList инициализирует список: заполняет массив nodes, устанавливает указатели и индексы.

4. Функция GetNode извлекает свободный элемент из списка, обновляя указатель на следующий свободный элемент.

5. Процедура ReleaseNode освобождает элемент списка, добавляя его в список свободных элементов.

6. Процедура AddNode добавляет новый элемент в список: создает новый узел, заполняет данными и добавляет его в конец списка.

7. Процедура PrintList выводит содержимое списка, начиная с головы и переходя к следующему элементу до конца списка.

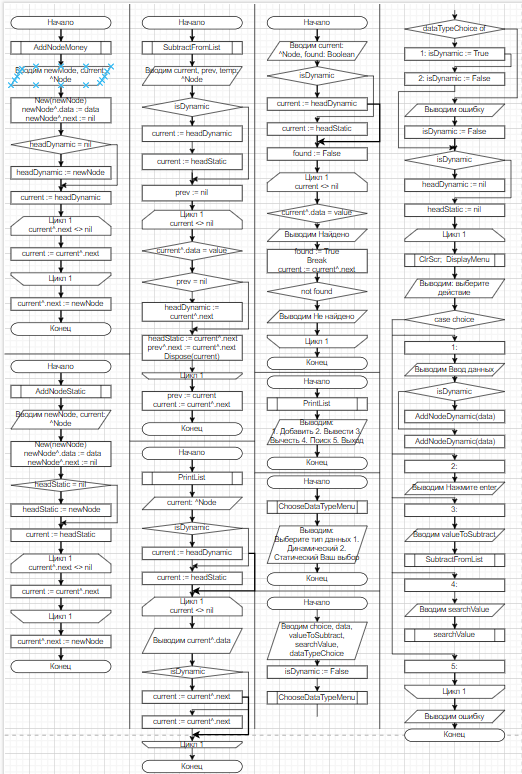
8. Процедура DisplayMenu отображает меню действий для пользователя: добавить элемент, вывести список или выйти из программы.

9. В основной части программы инициализируется список, затем в цикле пользователю предлагается выбрать действие из меню.

10. С помощью case-структуры программа выполняет нужное действие в зависимости от выбора пользователя: добавляет элемент, выводит список или завершает работу.

11. После каждой операции содержимое списка выводится на экран для наглядности.

**Схема алгоритма:**

****

*Рисунок 1 – схема-алгоритм*



**Код программы:**

**program** DynamicStaticLinkedListExample;

**uses** Crt;

**type**

Node = **record**

data: Integer;

next: ^Node;

**end**;

**var**

headDynamic, headStatic: ^Node;

isDynamic: Boolean;

**procedure** AddNodeDynamic(data: Integer);

**var**

newNode, current: ^Node;

**begin**

**New**(newNode);

newNode^.data := data;

newNode^.next := nil;

**if** headDynamic = nil **then**

headDynamic := newNode

**else**

**begin**

current := headDynamic;

**while** current^.next <> nil **do**

current := current^.next;

current^.next := newNode;

**end**;

**end**;

**procedure** AddNodeStatic(data: Integer);

**var**

newNode, current: ^Node;

**begin**

**New**(newNode);

newNode^.data := data;

newNode^.next := nil;

**if** headStatic = nil **then**

headStatic := newNode

**else**

**begin**

current := headStatic;

**while** current^.next <> nil **do**

current := current^.next;

current^.next := newNode;

**end**;

**end**;

**procedure** SubtractFromList(value: Integer);

**var**

current, prev, temp: ^Node;

**begin**

**if** isDynamic **then**

current := headDynamic

**else**

current := headStatic;

prev := nil;

**while** current <> nil **do**

**begin**

**if** current^.data = value **then**

**begin**

**if** prev = nil **then**

**begin**

**if** isDynamic **then**

headDynamic := current^.next

**else**

headStatic := current^.next;

**end**

**else**

prev^.next := current^.next;

Dispose(current);

**Break**;

**end**;

prev := current;

current := current^.next;

**end**;

**end**;

**procedure** PrintList;

**var**

current: ^Node;

**begin**

**if** isDynamic **then**

current := headDynamic

**else**

current := headStatic;

**while** current <> nil **do**

**begin**

Write(current^.data, ' ');

**if** isDynamic **then**

current := current^.next

**else**

current := current^.next;

**end**;

**end**;

**procedure** SearchInList(value: Integer);

**var**

current: ^Node;

found: Boolean;

**begin**

**if** isDynamic **then**

current := headDynamic

**else**

current := headStatic;

found := False;

**while** current <> nil **do**

**begin**

**if** current^.data = value **then**

**begin**

WriteLn('Число ', value, ' найдено в списке.');

found := True;

**Break**;

**end**;

current := current^.next;

**end**;

**if not** found **then**

WriteLn('Число ', value, ' не найдено в списке.');

**end**;

**procedure** DisplayMenu;

**begin**

WriteLn('1. Добавить элемент в список');

WriteLn('2. Вывести содержимое списка');

WriteLn('3. Вычесть число из списка');

WriteLn('4. Поиск числа в списке');

WriteLn('5. Выход');

**end**;

**procedure** ChooseDataTypeMenu;

**begin**

WriteLn('Выберите тип данных:');

WriteLn('1. Динамический');

WriteLn('2. Статический');

WriteLn;

Write('Ваш выбор: ');

**end**;

**var**

choice, data, valueToSubtract, searchValue, dataTypeChoice: Integer;

**begin**

isDynamic := False; // По умолчанию выбран статический тип данных

ChooseDataTypeMenu;

ReadLn(dataTypeChoice);

**case** dataTypeChoice **of**

1: isDynamic := True; // Выбран динамический тип данных

2: isDynamic := False; // Выбран статический тип данных

**else**

**begin**

WriteLn('Некорректный выбор типа данных. Используется статический тип данных по умолчанию.');

isDynamic := False; // Используется статический тип данных по умолчанию при некорректном вводе

**end**;

**end**;

**if** isDynamic **then** // Инициализация головы в зависимости от выбранного типа данных

headDynamic := nil

**else**

headStatic := nil;

**repeat**

ClrScr;

DisplayMenu;

WriteLn;

Write('Выберите действие: ');

ReadLn(choice);

**case** choice **of**

1:

**begin**

Write('Введите данные для нового элемента: ');

ReadLn(data);

**if** isDynamic **then**

AddNodeDynamic(data)

**else**

AddNodeStatic(data);

**end**;

2:

**begin**

Write('Linked List: ');

PrintList;

WriteLn;

WriteLn('Нажмите Enter для продолжения...');

ReadLn;

**end**;

3:

**begin**

Write('Введите число для вычитания из списка: ');

ReadLn(valueToSubtract);

SubtractFromList(valueToSubtract);

**end**;

4:

**begin**

Write('Введите число для поиска в списке: ');

ReadLn(searchValue);

SearchInList(searchValue);

WriteLn('Нажмите Enter для продолжения...');

ReadLn;

**end**;

5: ;

**else**

WriteLn('Некорректный выбор. Попробуйте снова.');

WriteLn('Нажмите Enter для продолжения...');

ReadLn;

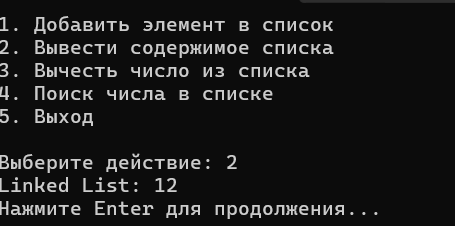
**end**;

**until** choice = 5;

**end**.

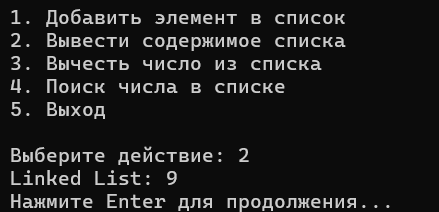
**Результат выполнения программы:**

Для проверки были введены значения 1 – выбор динамического типа данных; 1 – добавлено число 12 в список, 2 – выведено содержимое списка.



*Рисунок 2 – первый вариант проверки выполнения программы*.

Для проверки были введены значения 2 – выбор статического типа данных; 1 – добавлено число 9, 12; 3 – вычет числа 12, 2 – вывод списка.



*Рисунок 3 – второй вариант проверки выполнения программы.*

**Вывод:**

В процессе выполнения Домашней Контрольной работы №2 я изучил принципы работы с базовыми структурами данных. Получил навыки организации case-меню. Успешно получилось модифицировать программу: структура данных реализована на основе статической памяти; работа со структурой осуществляется в помощью case-меню; также предусмотрено наглядная визуализация содержимой структуры.