Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

Колледж ВятГУ

**ОТЧЕТ**

**ПО ДОМАШНЕЙ КОНТРОЛЬНОЙ РАБОТЕ №6**

**«РЕАЛИЗАЦИЯ ЭЛЕМЕНТАРНЫХ СТРУКТУР ДАННЫХ НА ОСНОВЕ СТАТИЧЕСКОЙ И ДИНАМИЧЕСКОЙ ПАМЯТИ»**

**МДК 05.02 «РАЗРАБОТКА КОДА ИНФОРМАЦИОННЫХ СИСТЕМ»**

Выполнил: студент учебной группы

ИСПк-203-52-00

Калинин Максим Константинович

Преподаватель:

Сергеева Елизавета Григорьевна

Киров

2024

1. **Цель работы:**

Изучение принципов работы с базовыми структурами данных, получение навыков организации case-меню.

1. **Формулировка задания:**
2. Написать программу для работы со структурой данных "Кольцевой двусвязный список".
3. Структура данных должна быть реализована на основе статической и динамической памяти, реализованной с помощью выбора в кейс-меню.

﻿﻿﻿Работа со структурой должна осуществляться с помощью case-меню. Предусмотреть наглядную визуализацию содержимого

**Описание алгоритма**

Реализованы два способа использования памяти: динамический и статический.

Схема работы:

1. инициализация списка;
2. выбор памяти;
3. использование case-меню;
4. вставка элемента в начало;
5. вставка элемента в конец;
6. вставка элемента после указанного значения;
7. вставка элемента перед указанным значением;
8. удаление элемент;
9. вывод всех элементов в списке.

**Схемы алгоритма.**

Изображение выглядит как диаграмма, Технический чертеж, рисунок, План

Автоматически созданное описание

Рисунок 1 – Схема алгоритма case-меню

**Операции по статической памяти:**

Изображение выглядит как текст, диаграмма, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2 – Схема алгоритма инициализации списка

Изображение выглядит как текст, диаграмма, линия

Автоматически созданное описание

Рисунок 3 – Схема алгоритма поиска пустого узла

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

Рисунок 4 – Схема алгоритма вставки элемента в пустой узел

Изображение выглядит как текст, чек, диаграмма, линия

Автоматически созданное описание

Рисунок 5 – Схема алгоритма вставка элемента в начало

Изображение выглядит как текст, чек, диаграмма, Шрифт

Автоматически созданное описание

Рисунок 6 – Схема алгоритма вставки элемента в конец

Изображение выглядит как текст, диаграмма, Параллельный, чек

Автоматически созданное описание

Рисунок 7 – Схема алгоритма вставка элемента перед указанным

Изображение выглядит как текст, Параллельный, документ, число

Автоматически созданное описание

Рисунок 8 **-** Схема алгоритма вставка элемента после указанного

Изображение выглядит как текст, диаграмма, Параллельный, линия

Автоматически созданное описание

Рисунок 9 - Схема алгоритма удаления элемента

Изображение выглядит как текст, диаграмма, линия, чек

Автоматически созданное описание

Рисунок 10 - Схема алгоритма вывода списка

**Операции по динамической памяти:**

Изображение выглядит как текст, диаграмма, снимок экрана, круг

Автоматически созданное описание

Рисунок 11 – Схема алгоритма создания указателя

Изображение выглядит как текст, диаграмма, линия, снимок экрана

Автоматически созданное описание

Рисунок 12 - Схема алгоритма добавления в начало

Изображение выглядит как текст, диаграмма, линия, снимок экрана

Автоматически созданное описание

Рисунок 13 - Схема алгоритма добавления в конец

Изображение выглядит как текст, диаграмма, чек, Параллельный

Автоматически созданное описание

Рисунок 14 - Схема алгоритма добавления предуказанным

Изображение выглядит как текст, диаграмма, Параллельный, чек

Автоматически созданное описание

Рисунок 15 - Схема алгоритма добавления после указанного

Изображение выглядит как текст, диаграмма, Параллельный, План

Автоматически созданное описание

Рисунок 16 - Схема алгоритма удаления элемента

Изображение выглядит как текст, диаграмма, линия, Шрифт

Автоматически созданное описание

Рисунок 17 - Схема алгоритма вывода списка

**Код программы.**

program DKR6;

uses

CRT;

const

MAX\_SIZE = 999999;

type

ElementType = Integer;

StaticNode = record

data: ElementType;

prev: Integer;

next: Integer;

end;

DynamicNodePtr = ^DynamicNode;

DynamicNode = record

data: ElementType;

prev: DynamicNodePtr;

next: DynamicNodePtr;

end;

var

UseStaticMemory: Boolean;

procedure UseStaticMemoryMenu;

var

Choice: Integer;

begin

repeat

ClrScr;

Writeln('Выберите тип памяти:');

Writeln('1. Статическая');

Writeln('2. Динамическая');

Write('Выберите тип памяти: ');

ReadLn(Choice);

case Choice of

1: UseStaticMemory := True;

2: UseStaticMemory := False;

end;

until (Choice = 1) or (Choice = 2);

end;

procedure StaticMemoryMenu; // использование статической памяти

var

list: array [1..MAX\_SIZE] of StaticNode;

head: Integer = 0;

tail: Integer = 0;

free: Integer = 1;

procedure initializeList; // инициализация списка

var

i: Integer;

begin

for i := 1 to MAX\_SIZE do

begin

list[i].prev := i - 1;

list[i].next := i + 1;

end;

list[MAX\_SIZE].next := 0;

end;

function getFreeNode: Integer; // возвращает индекс свободного узла

begin

if free = 0 then

getFreeNode := 0 // если нету

else

begin

getFreeNode := free;

free := list[free].next; // обновление индекса узла на след.

end;

end;

procedure releaseNode(index: Integer); // освобождает узал со знач.

begin

list[index].prev := 0; // обнуление ссылка на пред. узел

list[index].next := free; // установка ссылки на свободный узел

free := index; // обновление индекса совободного узла

end;

procedure addToBeginning(value: ElementType); // добавление в начало

var

index: Integer;

begin

index := getFreeNode; // получаем индекс свободного узла

if index <> 0 then

begin

list[index].data := value;

list[index].prev := 0; // ссылка на пред. узел

list[index].next := head; // ссылка на след. узел

if head <> 0 then

list[head].prev := index;

head := index;

if tail = 0 then

tail := head;

end;

end;

procedure addToEnd(value: ElementType); // добавление в конец

var

index: Integer;

begin

index := getFreeNode;

if index <> 0 then // если есть свободный узел

begin

list[index].data := value; // записываем знач. в сободный

list[index].prev := tail; // ссылка на пред. узел

list[index].next := 0; // устанавливаем 0 т.к последний

if tail <> 0 then // если не пустой

list[tail].next := index; // ссылка на след. узел

tail := index;

if head = 0 then

head := tail;

end;

end;

// вставка перед

procedure insertBefore(existingValue: ElementType; newValue: ElementType);

var

index, currentNode: Integer;

begin

currentNode := head; // начало с головного узла

// ищем нужный заданный узел

while (currentNode <> 0) and (list[currentNode].data <> existingValue) do

currentNode := list[currentNode].next;

if currentNode <> 0 then // если узел существует

begin

index := getFreeNode;

if index <> 0 then

begin

list[index].data := newValue; // записываем значение в свобод. узел

// устанавливем ссылку на пред.

list[index].prev := list[currentNode].prev;

list[index].next := currentNode; // устанавливаем ссылку на текущий

if list[currentNode].prev <> 0 then // если пред. сыществует

// обновляем ссылку на след.

list[list[currentNode].prev].next := index

else

head := index; // обновляем головной

list[currentNode].prev := index; // обновляем ссылку на пред.

end;

end;

end;

// вставка после

procedure insertAfter(existingValue: ElementType; newValue: ElementType);

var

index, currentNode: Integer;

begin

currentNode := head;

// пока не найден узал с заданный значением

while (currentNode <> 0) and (list[currentNode].data <> existingValue) do

currentNode := list[currentNode].next; // переход к след. узлу

if currentNode <> 0 then // если узел существует

begin

index := getFreeNode; // получаем индекс свободного

if index <> 0 then // если есть свободный узел в списке

begin

list[index].data := newValue; // новое знач. в свобод. узел

list[index].prev := currentNode; // ссылка на пред. узел

list[index].next := list[currentNode].next; // ссылка на след. узел

if list[currentNode].next <> 0 then // если узел существует

list[list[currentNode].next].prev := index

else

tail := index;

list[currentNode].next := index; // ссылка на след. узел текущ. узла

end;

end;

end;

procedure delete(value: ElementType); // удаление элемента

var

currentNode: Integer;

begin

currentNode := head;

while (currentNode <> 0) and (list[currentNode].data <> value) do

currentNode := list[currentNode].next;

if currentNode <> 0 then // если узел существует

begin

if list[currentNode].prev <> 0 then // если пред. узел существует

list[list[currentNode].prev].next := list[currentNode].next

else

head := list[currentNode].next;

if list[currentNode].next <> 0 then // если след. узел существует

list[list[currentNode].next].prev := list[currentNode].prev

else

tail := list[currentNode].prev; // обновляет последний узел

releaseNode(currentNode); // осводождает память удлаенного

end;

end;

procedure printList; // вывод списка

var

currentNode: Integer;

begin

currentNode := head;

while currentNode <> 0 do

begin

Write(list[currentNode].data, ' ');

currentNode := list[currentNode].next;

end;

Writeln;

end;

var

choice: Integer;

value, existingValue, newValue: ElementType;

begin

initializeList;

repeat

ClrScr;

Writeln('1. Добавить в начало');

Writeln('2. Добавить в конец');

Writeln('3. Вставить перед элементом');

Writeln('4. Вставить после элемента');

Writeln('5. Удалить элемент');

Writeln('6. Вывести список');

Writeln('7. Вернуться в предыдущее меню');

Write('Выберите действие: ');

ReadLn(choice);

case choice of

1:

begin

Write('Введите значение: ');

ReadLn(value);

addToBeginning(value);

end;

2:

begin

Write('Введите значение: ');

ReadLn(value);

addToEnd(value);

end;

3:

begin

Write('Введите существующее значение: ');

ReadLn(existingValue);

Write('Введите новое значение: ');

ReadLn(newValue);

insertBefore(existingValue, newValue);

end;

4:

begin

Write('Введите существующее значение: ');

ReadLn(existingValue);

Write('Введите новое значение: ');

ReadLn(newValue);

insertAfter(existingValue, newValue);

end;

5:

begin

Write('Введите значение для удаления: ');

ReadLn(value);

delete(value);

end;

6:

begin

Writeln('Список:');

printList;

Write('Нажмите любую клавишу для продолжения...');

ReadKey;

end;

end;

until choice = 7;

end;

procedure DynamicMemoryMenu; // использование динамичекой памяти

var

head: DynamicNodePtr = nil;

tail: DynamicNodePtr = nil;

// создание нового динамического узла

function createNode(data: ElementType): DynamicNodePtr;

var

newNode: DynamicNodePtr; // объявление указателя на новый узел

begin

New(newNode); // выделение памяти

newNode^.data := data;

newNode^.prev := nil;

newNode^.next := nil;

createNode := newNode;

end;

procedure addToBeginning(value: ElementType); // добавление в начало

var

newNode: DynamicNodePtr;

begin

newNode := createNode(value); // создание нового узла

if head = nil then // если список пуст

begin

head := newNode;

tail := newNode;

end

else

begin

newNode^.next := head;

head^.prev := newNode;

head := newNode;

end;

end;

procedure addToEnd(value: ElementType); // добавление элемента в конец

var

newNode: DynamicNodePtr;

begin

newNode := createNode(value); // создание нового узла с заданный знач.

if tail = nil then

begin

head := newNode;

tail := newNode;

end

else

begin

newNode^.prev := tail;

tail^.next := newNode;

tail := newNode;

end;

end;

// вставка перед

procedure insertBefore(existingValue: ElementType; newValue: ElementType);

var

currentNode, newNode: DynamicNodePtr;

begin

currentNode := head;

// пока не найден элемент с заданным значением

while (currentNode <> nil) and (currentNode^.data <> existingValue) do

currentNode := currentNode^.next; // переходим к след. узлу

if currentNode <> nil then // если узел существует

begin

newNode := createNode(newValue); // создаем узел с заданным знач.

newNode^.prev := currentNode^.prev; // устанавливаем ссылку

newNode^.next := currentNode;

if currentNode^.prev <> nil then // если пред. узел существует

currentNode^.prev^.next := newNode

else

head := newNode;

currentNode^.prev := newNode;

end;

end;

// вставка после

procedure insertAfter(existingValue: ElementType; newValue: ElementType);

var

currentNode, newNode: DynamicNodePtr;

begin

currentNode := head;

// пока не найден узел с заданным значением

while (currentNode <> nil) and (currentNode^.data <> existingValue) do

currentNode := currentNode^.next; // переходим к следующему узлу

if currentNode <> nil then // если узел существует

begin

newNode := createNode(newValue); // создаем узел с заданным значением

newNode^.prev := currentNode;

newNode^.next := currentNode^.next;

if currentNode^.next <> nil then

currentNode^.next^.prev := newNode

else

tail := newNode;

currentNode^.next := newNode; // обновляем след. узел текущего узла

end;

end;

procedure delete(value: ElementType); // удаление элемента

var

currentNode: DynamicNodePtr;

begin

currentNode := head;

// пока не найден узел с заданным значением

while (currentNode <> nil) and (currentNode^.data <> value) do

currentNode := currentNode^.next; // переходим к след. узлу

if currentNode <> nil then

begin

if currentNode^.prev <> nil then

currentNode^.prev^.next := currentNode^.next // обновляем ссылку

else

head := currentNode^.next;

if currentNode^.next <> nil then

currentNode^.next^.prev := currentNode^.prev

else

tail := currentNode^.prev;

Dispose(currentNode); // освобождаем память

end;

end;

procedure printList; // вывод списка

var

currentNode: DynamicNodePtr;

begin

currentNode := head;

while currentNode <> nil do

begin

Write(currentNode^.data, ' ');

currentNode := currentNode^.next;

end;

Writeln;

end;

var

choice: Integer;

value, existingValue, newValue: ElementType;

begin

repeat

ClrScr;

Writeln('1. Добавить в начало');

Writeln('2. Добавить в конец');

Writeln('3. Вставить перед элементом');

Writeln('4. Вставить после элемента');

Writeln('5. Удалить элемент');

Writeln('6. Вывести список');

Writeln('7. Вернуться в предыдущее меню');

Write('Выберите действие: ');

ReadLn(choice);

case choice of

1:

begin

Write('Введите значение: ');

ReadLn(value);

addToBeginning(value);

end;

2:

begin

Write('Введите значение: ');

ReadLn(value);

addToEnd(value);

end;

3:

begin

Write('Введите существующее значение: ');

ReadLn(existingValue);

Write('Введите новое значение: ');

ReadLn(newValue);

insertBefore(existingValue, newValue);

end;

4:

begin

Write('Введите существующее значение: ');

ReadLn(existingValue);

Write('Введите новое значение: ');

ReadLn(newValue);

insertAfter(existingValue, newValue);

end;

5:

begin

Write('Введите значение для удаления: ');

ReadLn(value);

delete(value);

end;

6:

begin

Writeln('Список:');

printList;

Write('Нажмите любую клавишу для продолжения...');

ReadKey;

end;

end;

until choice = 7;

end;

procedure MainMenu;

var

choice: Integer;

begin

repeat

ClrScr;

Writeln('1. Работа с динамическим списком');

Writeln('2. Работа со статическим списком');

Writeln('3. Выход');

Write('Выберите действие: ');

ReadLn(choice);

case choice of

1: DynamicMemoryMenu;

2: StaticMemoryMenu;

end;

until choice = 3;

end;

begin

MainMenu;

end.

**Результат выполнения программы.**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 18 – Операция по добавлению в начало и конец и вывода списка на экран

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 19 – Операция по вставке элемента перед и после указанного

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 20 – Операция по удалению элемента

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 21 – Переход между выбором памяти

Изображение выглядит как текст, снимок экрана, мультимедиа, гаджет

Автоматически созданное описание

Рисунок 22 – Выбор действий со статической и динамической памятью

Изображение выглядит как текст, снимок экрана, мультимедиа, гаджет

Автоматически созданное описание

Рисунок 23 – Вывод элементов в динамической памяти

**Изображение выглядит как текст, электроника, снимок экрана, мультимедиа

Автоматически созданное описание**

Рисунок 24 – Вывод элемента при заполненной памяти

**Вывод.**

В ходе работы над данной задачей были изучены принципы организации кольцевого двусвязного списка с использованием статической и динамической памяти. Разработана программа с case-меню, предоставляющая возможность визуализации содержимого списка и выполнения основных операций: инициализации, вставки элементов, удаления элементов и вывода списка.

Реализованы два подхода к управлению памятью. Статическая память, основанная на массиве фиксированного размера, обеспечивает быстрое выделение и освобождение памяти, но имеет ограничение на максимальное количество элементов. Динамическая память, использующая указатели, позволяет списку расти динамически, но требует дополнительных ресурсов на управление памятью.

Алгоритмы каждой операции детально описаны и проиллюстрированы схемами, демонстрирующими этапы работы с узлами списка. Результаты выполнения программы подтверждают корректность реализации алгоритмов и выбранных подходов к организации памяти.