**РАЗДЕЛ 2. РЕАЛИЗАЦИЯ ОДНОСВЯЗНОГО СПИСКА И ФУНКЦИЙ ДЛЯ РАБОТЫ С НИМ**

**2.1. Листинг List.cpp**

#include <iostream>

#include <iomanip>

#include <fstream>

#include "List\_Header.h"

int menu\_selection; // Переменная для выбора функции

int counter = 0; // Счётчик узлов списка

int main()

{

List \*u = NULL; // Определяем указатель на начало //будущего списка

List \*x = u; // Вспомогательная переменная- //указатель, которая будет хранить //адрес последнего узла списка

// Сейчас последний узел списка //совпадает с его началом

do

{

Menu(); // Меню выбора действия над списком

switch (menu\_selection)

{

case 1:

Add\_top(&u, &x);

break;

case 2:

Add\_end(&u, &x);

break;

case 3:

Print(u);

break;

case 4:

Delete\_top(&u, &x);

break;

case 5:

Delete\_end(&u, &x);

break;

case 6:

Delete\_index(&u, &x);

break;

case 7:

Clear(&u, &x);

break;

case 8:

Sort\_rad\_asc(&u, &x);

break;

case 9:

Move\_right(&u, &x);

break;

case 10:

Move\_left(&u, &x);

break;

case 11:

File\_in(u);

break;

case 12:

File\_in\_rad(u);

break;

case 13:

File\_out(u);

break;

default:

break;

}

} while (menu\_selection != 0);

return 0;

}

**2.2. Листинг List\_Header.h**

#pragma once

extern int menu\_selection;

extern int counter;

struct Data // Структура, содержащая характеристики данных

{

double coord\_x;

double coord\_y;

double rad;

};

struct List // Структура, содержащая поле типа Data и поле - //адрес последующего элемента next

{

Data d;

List \*next;

};

int Menu();

void Add\_top(List \*\*u, List \*\*x);

void Add\_end(List \*\*u, List \*\*x);

void Print(List \*u);

void Delete\_top(List \*\*u, List \*\*x);

void Delete\_end(List \*\*u, List \*\*x);

void Delete\_index(List \*\*u, List \*\*x);

void Clear(List \*\*u, List \*\*x);

void Sort\_rad\_asc(List \*\*u, List \*\*x);

void Move\_right(List \*\*u, List \*\*x);

void Move\_left(List \*\*u, List \*\*x);

void File\_in(List \*u);

void File\_in\_rad(List \*u);

void File\_out(List \*u);

**2.3. Листинг List\_Func.cpp**

#include <iostream>

#include <iomanip>

#include <fstream>

#include "List\_Header.h"

**2.3.1. Меню выбора действия**

int Menu()

{

std::cout << "1 : Add an element to the top" << std::endl;

std::cout << "2 : Add an element to the end" << std::endl;

std::cout << "3 : Show List" << std::endl;

std::cout << "4 : Remove an element at the beginning of the list" << std::endl;

std::cout << "5 : Remove an element at the end of the list"

<< std::endl;

std::cout << "6 : Delete an item with serial number N"

<< std::endl;

std::cout << "7 : Remove the list" << std::endl;

std::cout << "8 : Sort ascending radius" << std::endl;

std::cout << "9 : Move the list to the right" << std::endl;

std::cout << "10 : Move the list to the left" << std::endl;

std::cout << "11 : Save list to a file" << std::endl;

std::cout << "12 : Save an element of radius R in a file"

<< std::endl;

std::cout << "13 : Show list from file" << std::endl;

std::cout << "0 : EXIT" << std::endl;

std::cout << "Select an action: ";

std::cin >> menu\_selection;

std::cout << std::endl;

return menu\_selection;

}

**2.3.2. Добавление нового узла в начало списка**

void Add\_top(List \*\*u, List \*\*x)

{

double temp\_x, temp\_y, temp\_rad;

std::cout << std::setw(15) << "Enter values:" << std::endl;

std::cout << std::setw(15) << "x = ";

std::cin >> temp\_x;

std::cout << std::setw(15) << "y = ";

std::cin >> temp\_y;

std::cout << std::setw(15) << "radius = ";

std::cin >> temp\_rad;

std::cout << std::endl;

if (\*u == NULL) // Выполняется, если в списке ещё //нет узлов

{

\*u = new List; //Выделяем память под узел списка

(\*u)->d.coord\_x = temp\_x; // Заполняем поля с данными

(\*u)->d.coord\_y = temp\_y; //

(\*u)->d.rad = temp\_rad; //

(\*u)->next = NULL; // Указатель на следующий узел //списка пуст

(\*x) = (\*u);

}

else

{

List \*t = new List;

t->d.coord\_x = temp\_x;

t->d.coord\_y = temp\_y;

t->d.rad = temp\_rad;

t->next = \*u;

\*u = t;

}

counter++; //Инкремент счётчика узлов списка

}

**2.3.3. Добавление нового узла в конец списка**

void Add\_end(List \*\*u, List \*\*x)

{

double temp\_x, temp\_y, temp\_rad;

std::cout << std::setw(15) << "Enter values:" << std::endl;

std::cout << std::setw(15) << "x = ";

std::cin >> temp\_x;

std::cout << std::setw(15) << "y = ";

std::cin >> temp\_y;

std::cout << std::setw(15) << "radius = ";

std::cin >> temp\_rad;

std::cout << std::endl;

if (\*u == NULL)

{

\*u = new List;

(\*u)->d.coord\_x = temp\_x;

(\*u)->d.coord\_y = temp\_y;

(\*u)->d.rad = temp\_rad;

(\*u)->next = NULL;

(\*x) = (\*u);

}

else

{

(\*x)->next = new List;

(\*x) = (\*x)->next;

(\*x)->d.coord\_x = temp\_x;

(\*x)->d.coord\_y = temp\_y;

(\*x)->d.rad = temp\_rad;

(\*x)->next = NULL;

}

counter++;

}

**2.3.4. Просмотр списка**

void Print(List \*u)

{

if (u)

{

List \*p = u;

std::cout << std::setw(15) << "List:" << std::endl;

while (p)

{

std::cout << std::setw(15) << "x = " << p->d.coord\_x

<< "\ty = " << p->d.coord\_y << "\tradius = " << p->d.rad << std::endl;

p = p->next;

}

std::cout << std::endl;

}

else

{

std::cout << "\t nothing to print - the list is empty"

<< std::endl;

std::cout << std::endl;

}

}

**2.3.5. Удаление первого узла списка**

void Delete\_top(List \*\*u, List \*\*x)

{

if (\*u == NULL) {

std::cout << "\t the list is empty - nothing to remove"

<< std::endl;

std::cout << std::endl;

}

else if (\*u == \*x) {

delete (\*u);

\*u = NULL;

\*x = NULL;

counter = 0;

std::cout << "\t list was deleted" << std::endl;

std::cout << std::endl;

}

else

{

List \*t = \*u;

\*u = t->next;

delete t;

std::cout << "\t the first element of the list is removed"

<< std::endl;

std::cout << std::endl;

counter--;

}

}

**2.3.6. Удаление последнего узла списка**

void Delete\_end(List \*\*u, List \*\*x)

{

if (\*u == NULL)

{

std::cout << "\t the list is empty - nothing to remove"

<< std::endl;

std::cout << std::endl;

}

else if (\*u == \*x)

{

delete (\*u);

\*u = NULL;

\*x = NULL;

counter = 0;

std::cout << "\t list was deleted" << std::endl;

std::cout << std::endl;

}

else

{

List \*p = (\*u);

List \*t = (\*u);

while (p != (\*x))

{

t = p;

p = p->next;

}

delete p;

t->next = NULL;

(\*x) = t;

counter--;

std::cout << "\t the last item removed" << std::endl;

std::cout << std::endl;

}

}

**2.3.7. Удаление n-го узла списка**

void Delete\_index(List \*\*u, List \*\*x)

{

if (\*u == NULL) // Если список пуст

{

std::cout << "\t the list is empty - nothing to remove"

<< std::endl;

std::cout << std::endl;

return;

}

int index;

std::cout << std::setw(15) << "\t Enter the serial number of the element to remove (from 1): ";

std::cin >> index;

if ((index < counter) && (index != 1)) // Если n-ый узел //существует и не первый

{

List \*left = (\*u);

List \*del = left->next;

List \*right = del->next;

int local\_counter = 2; // Потому что в троице // left|del|right

//удаляемый узел //изначально второй по //счёту

while (local\_counter != index)

{

left = left->next;

del = del->next;

right = right->next;

local\_counter++;

}

delete del;

left->next = right;

printf("\t Item number %i has been removed", index);

std::cout << std::endl;

counter--;

}

else if (counter < index) // Если n-ый узел не //существует

{

printf("\t The element %i does not exist. The list contains %i elements", index, counter);

std::cout << std::endl;

std::cout << std::endl;

}

else if (index == 1) // Если необходимо удалить

//первый узел

{

List \*t = \*u;

\*u = t->next;

delete t;

printf("\t Item number %i has been removed", index);

std::cout << std::endl;

counter--;

}

else if (index == counter) // Если необходимо удалить

//последний узел

{

List \*p = (\*u);

List \*t = (\*u);

while (p != (\*x))

{

t = p;

p = p->next;

}

delete p;

t->next = NULL;

(\*x) = t;

counter--;

printf("\t Item number %i has been removed", index);

std::cout << std::endl;

}

}

**2.3.8. Удаление (очистка) всего списка**

void Clear(List \*\*u, List \*\*x)

{

if (\*u == NULL)

{

std::cout << "\t the list is empty - nothing to remove"

<< std::endl;

std::cout << std::endl;

}

else

{

List \*p = \*u;

List \*t;

while (p)

{

t = p;

p = p->next;

delete t;

}

\*u = NULL;

\*x = NULL;

counter = 0;

std::cout << "\t list was deleted" << std::endl;

std::cout << std::endl;

}

}

**2.3.9. Сортировка узлов списка по возрастанию радиуса**

void Sort\_rad\_asc(List \*\*u, List \*\*x)

{

if (\*u != NULL)

{

List \*left;

List \*right;

List \*tmp;

List \*p = (\*u);

List \*end = (\*x);

while (p != end)

{

left = p;

right = left->next;

do

{

if (right->d.rad < left->d.rad) {

double tmp\_rad = right->d.rad;

right->d.rad = left->d.rad;

left->d.rad = tmp\_rad;

double tmp\_x = right->d.coord\_x;

right->d.coord\_x = left->d.coord\_x;

left->d.coord\_x = tmp\_x;

double tmp\_y = right->d.coord\_y;

right->d.coord\_y = left->d.coord\_y;

left->d.coord\_y = tmp\_y;

}

tmp = left;

left = left->next;

right = right->next;

} while (right != end->next);

end = tmp;

}

std::cout << "\t the list has been sorted" << std::endl;

std::cout << std::endl;

}

else

{

std::cout << "\t the list is empty - nothing to sort"

<< std::endl;

std::cout << std::endl;

}

}

**2.3.10. Сдвиг списка на N узлов вправо**

void Move\_right(List \*\*u, List \*\*x)

{

if ((\*u) == NULL)

{

std::cout << "\t the list is empty - there is nothing to move" << std::endl;

std::cout << std::endl;

}

else if (\*u == \*x)

{

std::cout << "\t list consists of a single element - move does not make sense" << std::endl;

std::cout << std::endl;

}

else

{

int shift;

int local\_counter = 0;

(\*x)->next = (\*u); // Замыкаем связный список

std::cout << "\t the value to which to move to the right= ";

std::cin >> shift;

std::cout << std::endl;

do

{

(\*u) = (\*u)->next;

(\*x) = (\*x)->next;

local\_counter++;

} while (local\_counter != shift);

(\*x)->next = NULL; // Преобразуем кольцевой список //назад в односвязный

std::cout << "\t list has been moved" << std::endl;

std::cout << std::endl;

}

}

**2.3.11. Сдвиг списка на N узлов влево**

void Move\_left(List \*\*u, List \*\*x)

{

if ((\*u) == NULL)

{

std::cout << "\t the list is empty - there is nothing to move" << std::endl;

std::cout << std::endl;

}

else if (\*u == \*x)

{

std::cout << "\t list consists of a single element - move does not make sense" << std::endl;

std::cout << std::endl;

}

else

{

int shift;

int local\_counter = 0;

(\*x)->next = (\*u); // Замыкаем связный список

std::cout << "\t the value to which to move to the left = ";

std::cin >> shift;

std::cout << std::endl;

if (counter > shift)

{

do

{

(\*u) = (\*u)->next;

(\*x) = (\*x)->next;

local\_counter++;

} while (local\_counter != (counter - shift));

}

else if (counter < shift)

{

do

{

shift = shift % counter;

(\*u) = (\*u)->next;

(\*x) = (\*x)->next;

local\_counter++;

} while (local\_counter != (counter - shift));

}

(\*x)->next = NULL;

std::cout << "\t list has been moved" << std::endl;

std::cout << std::endl;

}

}

**2.3.12. Сохранение в файл**

void File\_in(List \*u)

{

if (u)

{

List \*p = u;

std::ofstream fout("linked\_list.txt");

fout << std::setw(15) << "List:" << std::endl;

while (p)

{

fout << std::setw(15) << "x = " << p->d.coord\_x

<< "\ty = " << p->d.coord\_y << "\tradius = " << p->d.rad << std::endl;

p = p->next;

}

fout.close();

std::cout << "\t the list has been saved to a file \"linked\_list.txt\" " << std::endl;

std::cout << std::endl;

}

else

{

std::cout << "\t list is empty - nothing to save in a file"

<< std::endl;

std::cout << std::endl;

}

}

**2.3.13. Сохранение в файл узлов с заданным радиусом**

void File\_in\_rad(List \*u)

{

if (u)

{

int radius\_search;

int local\_count = 1;

int search\_quantity = 0;

std::cout << std::setw(15) << "\t Enter the radius for the search in the list: ";

std::cin >> radius\_search;

List \*p = u;

std::ofstream fout("linked\_list\_rad.txt");

fout << std::setw(15) << "List of elements with a radius "

<< radius\_search << std::endl;

while (p)

{

if (p->d.rad == radius\_search)

{

fout << std::setw(15) << local\_count << "\tx = "

<< p->d.coord\_x << "\ty = " << p->d.coord\_y

<< "\tradius = " << p->d.rad << std::endl;

search\_quantity++;

}

p = p->next;

local\_count++;

}

fout.close();

printf("\t List of elements with a radius %i has been saved in a file linked\_list\_rad.txt", radius\_search);

std::cout << std::endl;

printf("\t %i elements found", search\_quantity);

std::cout << std::endl;

}

else

{

std::cout << "\t list is empty - nothing to save in a file"

<< std::endl;

std::cout << std::endl;

}

}

**2.3.14. Отображение из файла**

void File\_out(List \*u)

{

if (u)

{

setlocale(LC\_ALL, "rus"); // корректное отображение //Кириллицы

char buff[50]; // буфер промежуточного //хранения считываемого из //файла текста

std::ifstream fin("linked\_list.txt");

while (!fin.eof()) // eof() возвращает 1, если //был достигнут конец //файла;

{

fin.getline(buff, 50);

std::cout << buff << std::endl;

}

fin.close();

}

else

{

std::cout << "\t list is empty - nothing to print"

<< std::endl;

std::cout << std::endl;

}

}