Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное

учреждение высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Инженерно-технологическая академия ЮФУ

Институт компьютерных технологий и информационной безопасности

ЛАБОРАТОРНАЯ РАБОТА № 9

по курсу: «Алгоритмизация и программирование»

на тему:

«Динамическое распределение памяти, списки»

Выполнил

студент группы КТсо1-6 Овсянникова В. А.

Проверил

Доктор технических наук, профессор Беляков С. Л.

Таганрог 2021

# Техническое задание

## Цель работы

Целью данной лабораторной работы является освоение студентами способов представления и обработки данных в виде списков.

Вариант № 9

Написать программы для работы с односвязным и двусвязным списками в соответствии с выданным вариантом задания.

9. Адресная книга.

# Выполнение работы

Функция first(string surname, string street, string number);

Функция формирует и возвращает первый элемент списка.

1. Выделяем память под первый элемент списка.
2. Присваиваем входные значения surname, street, number первому элементу списка.
3. Нет следующего элемента.
4. Нет предыдущего элемента.
5. Возвращаем первый элемент списка.

Исходный код first(string surname, string street, string number);

Node\* first(string surname, string street, string number) {

Node\* pv = new Node;

pv->surname = surname;

pv->street = street;

pv->number = number;

pv->next = 0;

pv->prev = 0;

return pv;

}

Функция add(Node\*\* pend, string surname, string street, string number);

Функция добавляет элемент в конец списка и ничего не возвращает.

1. Выделяем память под элемент списка.
2. Присваиваем входные значения surname, street, number элементу списка.
3. Нет следующего элемента.
4. Предыдущему элементу списка присваивается указатель на конец списка.
5. Для элемента с указателем на конец списка текущий элемент становится следующим.
6. Указателю на конец списка присваивается текущий элемент.

Исходный код add(Node\*\* pend, string surname, string street, string number)

void add(Node\*\* pend, string surname, string street, string number) {

Node\* pv = new Node;

pv->surname = surname;

pv->street = street;

pv->number = number;

cout << "\n";

pv->next = 0;

pv->prev = \*pend;

(\*pend)->next = pv;

\*pend = pv;

}

Функция find(Node\* const pbeg, string surname, string street, string number);

Функция осуществляет поиск элемента по ключу и возвращает элемент.

1. Текущий элемент принимает указатель на начало списка.
2. Пока не найден текущий элемент проходим по списку:

Если найдена фамилия – выход из цикла;

Если найдена улица – выход из цикла;

Если найден номер – выход из цикла;

1. Возвращает найденный элемент.

Исходный код функции find(Node\* const pbeg, string surname, string street, string number)

Node\* find(Node\* const pbeg, string surname, string street, string number) {

Node\* pv = pbeg;

while (pv) {

if (pv->surname == surname) break;

if (pv->street == street) break;

if (pv->number == number) break;

pv = pv->next;

}

return pv;

}

Функция find1(Node\* const pend, string surname, string street, string number);

Функция осуществляет поиск элемента по ключу и возвращает элемент.

1. Текущий элемент принимает указатель на конец списка.
2. Пока не найден текущий элемент проходим по списку:

Если найдена фамилия – выход из цикла;

Если найдена улица – выход из цикла;

Если найден номер – выход из цикла;

1. Возвращает найденный элемент.

Исходный код функции find1(Node\* const pend, string surname, string street, string number)

Node\* find1(Node\* const pend, string surname, string street, string number) {

Node\* pv = pend;

while (pv) {

if (pv->surname == surname) break;

if (pv->street == street) break;

if (pv->number == number) break;

pv = pv->prev;

}

return pv;

}

Функция insert(Node\* const pbeg, Node\*\* pend, string surname, string street, string number, string surname1, string street1, string number1)

Функция осуществляет вставку элемента после заданного и возвращает 0

1. Если значение элемента найдено функцией find(Node\* const pbeg, string surname, string street, string number) и присвоено \*pkey, то выделяем память для нового элемента, присваиваем ему значения, устанавливаем связь нового узла с последующим, устанавливаем связь нового узла с предыдущим, устанавливаем связь предыдущего узла с новым, устанавливаем связь последующего узла с новым, обновляем указатель на конец списка, если узел вставляется в конец, возвращаем элемент.
2. Возвращаем 0.

Исходный код функции insert(Node\* const pbeg, Node\*\* pend, string surname, string street, string number, string surname1, string street1, string number1)

Node\* insert(Node\* const pbeg, Node\*\* pend, string surname, string street, string number, string surname1, string street1, string number1) {

if (Node\* pkey = find(pbeg, surname, street, number)) {

Node\* pv = new Node;

pv->surname = surname1;

pv->street = street1;

pv->number = number1;

pv->next = pkey->next;

pv->prev = pkey;

pkey->next = pv;

if (pkey != \*pend) (pv->next)->prev = pv;

else \*pend = pv;

return pv;

}

return 0;

}

Функция insertpered(Node\*\* pbeg, Node\* const pend, string surname, string street, string number, string surname1, string street1, string number1)

Функция осуществляет вставку элемента до заданного и возвращает 0

1. Если значение элемента найдено функцией find1(Node\* const pend, string surname, string street, string number) и присвоено \*pkey, то выделяем память для нового элемента, присваиваем ему значения, устанавливаем связь нового узла с предыдущим, устанавливаем связь нового узла с последующим, устанавливаем связь последующего узла с новым, устанавливаем связь предыдущего узла с новым, обновляем указатель на начало списка, если узел вставляется в начало, возвращаем элемент.
2. Возвращаем 0.

Исходный код функции insertpered(Node\*\* pbeg, Node\* const pend, string surname, string street, string number, string surname1, string street1, string number1)

Node\* insertpered(Node\*\* pbeg, Node\* const pend, string surname, string street, string number, string surname1, string street1, string number1) {

if (Node\* pkey = find1(pend, surname, street, number)) {

Node\* pv = new Node;

pv->surname = surname1;

pv->street = street1;

pv->number = number1;

pv->prev = pkey->prev;

pv->next = pkey;

pkey->prev = pv;

if (pkey != \*pbeg) (pv->prev)->next = pv;

else \*pbeg = pv;

return pv;

}

return 0;

}

Функция add\_sort(Node\*\* pbeg, Node\*\* pend, string surname, string street, string number)

Сортирует список по фамилии и вставляет элемент в соответствующее место, ничего не возвращает.

1. Выделяем память под элемент списка
2. Присваиваем входные значения surname, street, number элементу списка.
3. Указатель \*pt будет указывать на начало списка.
4. Просмотр списка
5. Если surname < pt->surname, тогда заносим перед текущим элементом (pt), текущий элемент становится следующим относительно внесенного.

Если surname < pt->surname и если фамилия из входных данных < первой фамилии в списке, то помещаем в начало списка. Указываем, что предыдущего элемента нет. Текущий элемент принимает указатель на начало списка.

Иначе вставляем в середину списка. Устанавливаем связь текущего узла с предыдущим и последующим.

1. После того, как список просмотрен и отсортирован, указываем, что не больше элементов в конце, последнему элементу передаем указатель на конец. Обновление указателя на конец списка, если узел вставляется в конец.

Исходный код функции add\_sort(Node\*\* pbeg, Node\*\* pend, string surname, string street, string number)

void add\_sort(Node\*\* pbeg, Node\*\* pend, string surname, string street, string number) {//surname

Node\* pv = new Node;

pv->surname = surname;

pv->street = street;

pv->number = number;

Node\* pt = \*pbeg;

while (pt) {

if (surname < pt->surname) {

pv->next = pt;

if (pt == \*pbeg) {

pv->prev = 0;

\*pbeg = pv;

}

else {

(pt->prev)->next = pv;

pv->prev = pt->prev;

}

pt->prev = pv;

return;

}

pt = pt->next;

}

pv->next = 0;

pv->prev = \*pend;

(\*pend)->next = pv;

\*pend = pv;

}

Функция remove(Node\*\* pbeg, Node\*\* pend, string surname, string street, string number)

Удаляет элемент по введенному ключу, в случае удаления возвращает true, иначе false.

1. Если значение элемента найдено функцией find(Node\* const pbeg, string surname, string street, string number) и присвоено \*pkey, то
2. Если элемент-ключ стоит в начале, то указатель на начало списка передается следующему элементу и указывается, что предыдущего элемента нет.
3. Если элемент-ключ стоит в конце, то указатель на конец списка передается предыдущему элементу и указывается, что следующего элемента нет.
4. Иначе следующий элемент связывается с предыдущим, а предыдущий со следующим.
5. Pkey удаляется.
6. Возвращаем true.
7. Если удаление по ключу не произошло, возвращаем false.

Исходный код функции remove(Node\*\* pbeg, Node\*\* pend, string surname, string street, string number):

bool remove(Node\*\* pbeg, Node\*\* pend, string surname, string street, string number) {

if (Node\* pkey = find(\*pbeg, surname, street, number)) {

if (pkey == \*pbeg) {

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

(\*pbeg)->prev = 0;

}

else if (pkey == \*pend) {

\*pend = (\*pend)->prev;

(\*pend)->next = 0;

}

else {

(pkey->prev)->next = pkey->next;

(pkey->next)->prev = pkey->prev;

}

delete pkey;

return true;

}

return false;

}

Функция remove1( Node\*\* pend)

Удаляет элемент из конца, в случае удаления возвращает true, иначе false.

1. Если элемент-ключ хранит указатель на последний элемент, то
2. указатель на конец списка передается предыдущему элементу и указывается, что следующего элемента нет.
3. Pkey удаляется.
4. Возвращаем true.
5. Если удаление по ключу не произошло, возвращаем false.

Исходный код функции remove1( Node\*\* pend)

bool remove1( Node\*\* pend) {

if (Node\* pkey = \*pend) {

if (pkey == \*pend) {

\*pend = (\*pend)->prev;

(\*pend)->next = 0;

}

delete pkey;

return true;

}

return false;

}

Функция remove2(Node\*\* pbeg)

Удаляет элемент из начала, в случае удаления возвращает true, иначе false.

1. Если элемент-ключ хранит указатель на первый элемент, то
2. указатель на конец списка передается следующему элементу и указывается, что предыдущего элемента нет.
3. Pkey удаляется.
4. Возвращаем true.
5. Если удаление по ключу не произошло, возвращаем false.

Исходный код функции remove2(Node\*\* pbeg)

bool remove2(Node\*\* pbeg)) {

if (Node\* pkey = \*pbeg) {

if (pkey == \*pbeg) {

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

(\*pbeg)->prev = 0;

}

delete pkey;

return true;

}

return false;

}

Функция Remove3(Node\*& pp, int n)

Удаляет элемент с соответствующим номером, возвращает структуру, в случае отсутствия необходимой строки возвращает Null

1. Указатель на текущий элемент
2. Отсчитываем n-ный элемент
3. Если нет элемента с таким номером, возвращает Null
4. Если необходимо удалить первый элемент, то указатель на первый элемент присваивается следующему
5. Иначе выполняется привязка предыдущего элемента к следующему
6. Если удаляем не последний элемент, то выполняется привязка предыдущего элемента к следующему
7. Возвращаем элемент

Исходный код Remove3(Node\*& pp, int n)

Node\* Remove3(Node\*& pp, int n)

{

Node\* q;

for (q = pp; q != NULL && n != 0; q = q->next, n--);

if (q == NULL) return NULL;

if (q->prev == NULL)

pp = q->next;

else q->prev->next = q->next;

if (q->next != NULL)

q->next->prev = q->prev;

return q;

}

Функция removec(struct Node\* start, Node\*& pbeg)

Удаляет элемент из середины, возвращает 0

1. Присваиваем переменной kol результат работы функции, считающей количество элементов
2. Присваиваем переменной n количество элементов, которое нужно пройти до достижения середины.
3. Функция удаления элемента с заданным номером
4. Возвращает 0.

Исходный код removec(struct Node\* start, Node\*& pbeg)

Node\* removec(struct Node\* start, Node\*& pbeg) {

int kol = kolichestvo(start);

int n = (kol / 2);

Remove3(pbeg, n);

return 0;

}

Функция display(struct Node\* start)

Выводит на экран результат работы

Пока есть элементы, выводит их на экран.

Исходный код display(struct Node\* start):

void display(struct Node\* start) {

while (start) {

cout << start->surname << " " << start->street << " " << start->number<< "\n";

start = start->next;

}

}

Функция kolichestvo(struct Node\* start)

Считает количество элементов

1. Присваиваем переменной kol 0;
2. При проходе по каждой строке kol++;
3. Возвращает kol.

Исходный код kolichestvo(struct Node\* start)

int kolichestvo(struct Node\* start) {

int kol = 0;

while (start) {

kol++;

start = start->next;

}

return kol;

}

Функция menu()

Выводит меню в консоль

Исходный код menu()

void menu() {

cout << "Выберите действие:" << "\n";

cout << "1. Новый элемент в конец списка" << endl << "2. Новый элемент на заданное место" << endl << "3. новый элемент после элемента с заданной нформацией" << endl

<< "4. новый элемент до элемента с заданной нформацией" << endl << "5. Добавить новый элемент в середину списка" << endl << "6. Исключить элемент из середины:" << endl

<< "7. Исключить элемент с заданной информацией" << endl << "8. Исключить элемент из конца списка" << endl

<< "9. Исключить элемент из заданного места списка" << endl << "10. Исключить элемент из начала списка" << endl;

}

Функция seredina(struct Node\* start, Node\*\* pbeg, Node\*\* pend, string surname, string street, string number, Node\* pp)

1. С помощью функции, находим глубину списка и делим на 2, чтоб узнать серединный элемент
2. Создаем указатель pv и toInsert на структуру адресной книги
3. Отсчитываем серединный элемент
4. По этому указателю toInsert передаём данные surname, street, number, которые будут новыми вставляемые элементами.
5. Если следующий элемент есть, то инициализируем указатель на предыдущий элемент текущим элементом

Исходный код seredina(struct Node\* start, Node\*\* pbeg, Node\*\* pend, string surname, string street, string number, Node\* pp)

void seredina(struct Node\* start, Node\*\* pbeg, Node\*\* pend, string surname, string street, string number, Node\* pp) {

int kol = kolichestvo(start);

int n = (kol / 2);

Node\* pv = new Node;

Node\* toInsert = new Node();

// Указатель на текущий элемент

for (pv = pp; pv != NULL && n != 0; pv = pv->next, n--); // Отсчитать n -ый

toInsert->surname = surname;

toInsert->street = street;

toInsert->number = number;

toInsert->next = pv->next;

toInsert->prev = pv;

pv->next->prev = toInsert;

pv->next = toInsert;

}

Инициализация структуры

struct Node {

string surname, street, number;

Node\* next;

Node\* prev;

};

Функция main()

1. Вызов функции меню
2. Выделяем память под структуру
3. Указатель начала списка присваиваем результату работы функции first("Petrov", "Chehova", "23");
4. Указатель конца присваиваем первому элементу.
5. Инициализируем переменную i и вводим через консоль ее значение
6. Оператор switch принимает значение i
7. Если i=1, объявляем переменную kol и вводим ее значение. Затем вводим значения для каждой структуры, исходя из указанного количества. Функция добавления в конец.

Результат выводится в консоль. Появляется предложение продолжить или завершить работу программы.

1. Если i=2, вводим значение структуры. Функция добавления по заданному принципу.

Результат выводится в консоль. Появляется предложение продолжить или завершить работу программы.

1. Если i=3, вводится элемент, после которого надо выполнить вставку, а затем элемент, который надо вставить. Функция вставки результата после указанного элемента. Вывод результата на экран. Появляется предложение продолжить или завершить работу программы.
2. Если i=4, вводится элемент, перед которым надо выполнить вставку, а затем элемент, который надо вставить. Функция вставки результата перед указанным элементом. Вывод результата на экран. Появляется предложение продолжить или завершить работу программы.

Если i=5, вводим значение структуры. Функция добавления структуры в середину. Результат выводится в консоль. Появляется предложение продолжить или завершить работу программы.

1. Если i=6, то функция удаления элемента из середины. Результат выводится в консоль. Появляется предложение продолжить или завершить работу программы.
2. Если i=7, то вводим элемент, который надо удалить. Функция удаления по заданному элементу. Результат выводится в консоль. Появляется предложение продолжить или завершить работу программы.
3. Если i=8, функция удаления из конца списка.

Результат выводится в консоль. Появляется предложение продолжить или завершить работу программы.

1. Если i=9, то вводим номер строки, которую хотим удалить. Функция удаления строки по заданному номеру.

Результат выводится в консоль. Появляется предложение продолжить или завершить работу программы.

1. Если i=10, функция удаления из начала списка.

Результат выводится в консоль. Появляется предложение продолжить или завершить работу программы.

1. Если i=0, функция меню и ввод соответствующего действию значения.
2. Вывод результата работы программы в консоль при завершении программы.
3. Возвращает 0.

Исходный код main():

int main()

{

setlocale(LC\_ALL, "Russian");

menu();

Node\* u = new Node;

Node\* pbeg = first("Petrov", "Chehova", "23");

Node\* pend = pbeg;

int i;

cin >> i;

while (i != 12) {

switch (i) {

case 1:

int kol;

cout << "Введите количество элементов, которое хотите добавить:";

cin >> kol;

cout << endl;

for (int j = 0; j < kol; j++) {

string n1, s1, nu1;

cin >> n1;

cin >> s1;

cin >> nu1;

add(&pend, n1, s1, nu1);

}

display(pbeg);

cout << endl << "(0) Назад в меню (12) Закрыть программу\n";

cin >> i;

break;

case 2:

cout << "Введите элемент: ";

cout << endl;

for (int j = 0; j < 1; j++) {

string n1, s1, nu1;

cin >> n1;

cin >> s1;

cin >> nu1;

add\_sort(&pbeg, &pend, n1, s1, nu1);

}

display(pbeg);

cout << endl << "(0) Назад в меню (12) Закрыть программу\n";

cin >> i;

break;

case 3:

cout << "Введите элементы, после которых надо выполнить вставку, а затем вставляемые элементы: ";

cout << endl;

for (int j = 0; j < 1; j++) {

string n1, s1, nu1, n2, s2, nu2;

cin >> n1;

cin >> s1;

cin >> nu1;

cin >> n2;

cin >> s2;

cin >> nu2;

insert(pbeg, &pend, n1, s1, nu1, n2, s2, nu2);

}

display(pbeg);

cout << endl << "(0) Назад в меню (12) Закрыть программу\n";

cin >> i;

break;

case 4:

cout << "Введите элементы, до которых надо выполнить вставку, а затем вставляемые элементы: ";

cout << endl;

for (int j = 0; j < 1; j++) {

string n1, s1, nu1, n2, s2, nu2;

cin >> n1;

cin >> s1;

cin >> nu1;

cin >> n2;

cin >> s2;

cin >> nu2;

insertpered(&pbeg, pend, n1, s1, nu1, n2, s2, nu2);

}

display(pbeg);

cout << endl << "(0) Назад в меню (12) Закрыть программу\n";

cin >> i;

break;

case 5:

cout << "Введите элемент: ";

cout << endl;

for (int j = 0; j < 1; j++) {

string n1, s1, nu1;

cin >> n1;

cin >> s1;

cin >> nu1;

seredina(pbeg, &pbeg, &pend,n1, s1, nu1, pbeg);

}

display(pbeg);

cout << endl << "(0) Назад в меню (12) Закрыть программу\n";

cin >> i;

break;

case 6:

removec(pbeg, pbeg);

display(pbeg);

cout << endl << "(0) Назад в меню (12) Закрыть программу\n";

cin >> i;

break;

case 7:

cout << "Введите элемент: ";

cout << endl;

for (int j = 0; j < 1; j++) {

string n1, s1, nu1;

cin >> n1;

cin >> s1;

cin >> nu1;

remove(&pbeg, &pend, n1, s1, nu1);

}

display(pbeg);

cout << endl << "(0) Назад в меню (12) Закрыть программу\n";

cin >> i;

break;

case 8:

remove1(&pend);

display(pbeg);

cout << endl << "(0) Назад в меню (12) Закрыть программу\n";

cin >> i;

break;

case 9:

int n;

cout << "укажите номер строки, которую хотите удалить:" << endl;

cin >> n;

Remove3(pbeg, n);

display(pbeg);

cout << endl << "(0) Назад в меню (12) Закрыть программу\n";

cin >> i;

break;

case 10:

remove2(&pbeg);

display(pbeg);

cout << endl << "(0) Назад в меню (12) Закрыть программу\n";

cin >> i;

break;

cout << endl << endl;

case 0:

menu();

cin >> i;

break;

}

Node\* pv = pbeg;

while (pv) {

cout << pv->surname << " " << pv->street << " " << pv->number << "\n";

pv = pv->next;

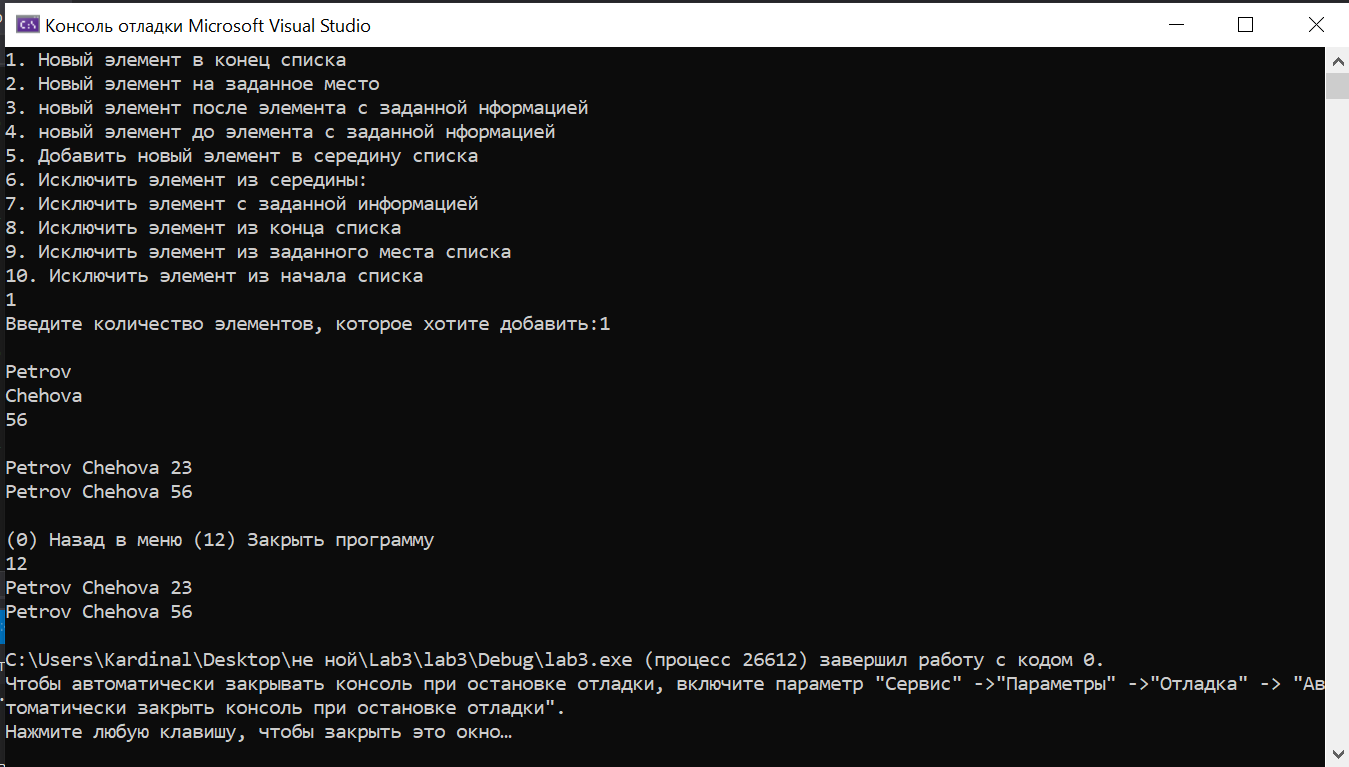
}

}

return 0;

}

Пример работы программы:



# Вывод

В процессе подготовки лабораторной работы были изучены основы языка С. Были освоены способы сортировки.