Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Восточно-Сибирский государственный университет технологий и управления

(ФГБОУ ВСГУТУ)

Электротехнический факультет

Кафедра «Системы Информатики»

Дисциплина

«Программирования»

Отчет по лабораторной работе №2

На тему:

Создание сложных структур данных

Вариант3

|  |  |
| --- | --- |
| Выполнил: | студент гр. Б661-2 |
|  | Коковихин А.В. |
| Проверил: | преподаватель  Жамбалов Э.Б. |

Улан-Удэ

2022

1. **Словесная постановка задачи**

Измените программу индивидуального задания прошлой лабораторной работы. Теперь данные должны быть представлены не в массиве, а в односвязном списке. Соответствующим образом переделайте все функции.

1. **Решение задачи**
   1. **Формальная постановка задачи**

Для выполнения данной работы изначальная структура была слегка переработана, В неё были добавлены указатели предыдущего и следующего элемента списка Next и Previos соответственно. А поля представлены указателем на структуру

Структура «Person» состоит из следующих полей.

* Указатель на следующий элемент;
* Указатель на предыдущий элемент;
* Структура с данными
  1. **Проектирование**

Теперь обращение к структурам будет происходить посредством организованного списка. Через указатели, предыдущего и следующего элемента. А для того чтобы обратиться к полям с данными, нужно будет сначала обратиться к вложенной структуре или её заранее заданному экземпляру.

* 1. **Реализация**

Структура теперь будет иметь следующий вид

struct Linked\_PersonPointers

{

Linked\_PersonPointers\* Next = NULL;

Linked\_PersonPointers\* Previos = NULL;

Person\* Person;

};

Где «Person\* Person» указатель на структуру, именно он и будет представлять поля структуры.

Для пунктов меню также будет использоваться перечисление

enum class PersonMenuCommand

{

Exit,

Create,

Sort,

Edit,

Delete,

Show\_All,

Search

};

Для функционирования списка и его основных функций были созданы следующие вспомогательные функции:

void Print(Person\* p)

{

cout << "Person: " << LR\_1::Get\_FIO(\*p) << endl;

cout << "- Adress:\t" << p->Adress << endl;

cout << "- Phone:\t" << p->Phone\_Number << endl;

}

Linked\_PersonPointers\* Add\_In\_List(Person\*& p, Linked\_PersonPointers\* tal = NULL)

{

Linked\_PersonPointers\* next = new Linked\_PersonPointers;

next->Previos = tal;

next->Person = p;

if (tal != NULL)

tal->Next = next;

return next;

}

Linked\_PersonPointers\* Get\_Linc\_On\_Index(int index, Linked\_PersonPointers\*& head)

{

Linked\_PersonPointers\* curent = head;

int i = 0;

while (curent != NULL)

{

if (i == index)

break;

i++;

curent = curent->Next;

}

return curent;

}

void Remove\_From\_List(int index, Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail)

{

if (head == NULL)

{

cout << "List is empty!" << endl;

return;

}

Linked\_PersonPointers\* deleting\_person = Get\_Linc\_On\_Index(index, head);

if (deleting\_person)

{

cout << "Do you want to remove: " << endl;

Print(deleting\_person->Person);

cout << "Y/N: ";

char c;

cin >> c;

if (c != 'Y' && c != 'y')

return;

if (deleting\_person->Next && deleting\_person->Previos)

{

deleting\_person->Previos->Next = deleting\_person->Next;

deleting\_person->Next->Previos = deleting\_person->Previos;

}

else if (deleting\_person->Next)

{

deleting\_person->Next->Previos = NULL;

head = deleting\_person->Next;

}

else if (deleting\_person->Previos)

{

deleting\_person->Previos->Next = NULL;

tail = deleting\_person->Previos;

}

else

{

head = NULL;

tail = NULL;

}

delete deleting\_person;

}

else

{

cout << "Index out of range!" << endl;

}

}

Person\* Enter\_Person\_Data()

{

Person\* creating\_person = new Person();

cout << "Enter second name: ";

cin >> creating\_person->Name.SecondName;

cout << "Enter first name: ";

cin >> creating\_person->Name.FirstName;

cout << "Has middle name? (Y/N): ";

char c;

cin >> c;

if (c == 'Y' || c == 'y')

{

cout << "Enter midle name: ";

cin >> creating\_person->Name.MidleName;

creating\_person->Name.Has\_MidleName = true;

}

cout << "Enter adress: ";

cin >> creating\_person->Adress;

cout << "Enter phone number: ";

cin >> creating\_person->Phone\_Number;

return creating\_person;

}

Основными функциями будут являться следующие:

Функция для вывода:

void Print(Linked\_PersonPointers\* head)

{

if (head == NULL)

{

cout << "List is empty!" << endl;

return;

}

Print(head->Person);

if (head->Next)

{

cout << endl;

Print(head->Next);

}

}

Функция для добавления элемента в список:

void Add\_Person(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail, Person\* p)

{

if (head == NULL)

{

head = Add\_In\_List(p);

tail = head;

}

else

tail = Add\_In\_List(p, tail);

}

Функция для удаления элемента из списка:

void Delete\_Person(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail)

{

cout << "Enter deleting person number: ";

int index;

cin >> index;

Remove\_From\_List(index, head, tail);

cout << "Deleted!" << endl;

}

Функция для изменения элемента списка:

void Edit\_Person(Linked\_PersonPointers\*& head, Linked\_PersonPointers\*& tail)

{

int i;

cout << "Editing person number: ";

cin >> i;

Linked\_PersonPointers\* editing\_person = Get\_Linc\_On\_Index(i, head);

if (editing\_person)

editing\_person->Person = Enter\_Person\_Data();

else

cout << "Index out of range!" << endl;

}

Функция для сортировки элемента списка:

void Search(Linked\_PersonPointers\* curent)

{

cout << "Enter FIO patern: ";

char find[255];

cin.ignore();

cin.getline(find, 254);

bool is\_finded = false;

while (curent)

{

string fio = LR\_1::Get\_FIO(\*curent->Person);

if (fio.find(find) != -1)

{

Print(curent->Person);

is\_finded = true;

}

curent = curent->Next;

}

if (is\_finded == false)

cout << "Nothing!" << endl;

}

Функция для поиска элемента списка:

void Search(Linked\_PersonPointers\* curent)

{

cout << "Enter FIO patern: ";

char find[255];

cin.ignore();

cin.getline(find, 254);

bool is\_finded = false;

while (curent)

{

string fio = LR\_1::Get\_FIO(\*curent->Person);

if (fio.find(find) != -1)

{

Print(curent->Person);

is\_finded = true;

}

curent = curent->Next;

}

if (is\_finded == false)

cout << "Nothing!" << endl;

}

Главная функция, соединяющая всё в единое целое и представляющая работу программы, будет иметь следующий вид:

void Update\_PersonMenu(Linked\_PersonPointers\* head = NULL, Linked\_PersonPointers\* tail = NULL)

{

switch (LR\_1::Choose\_Command())

{

case MenuCommand::Create:

Create\_Person(head, tail);

break;

case MenuCommand::Sort:

Sort(head, tail);

Print(head);

break;

case MenuCommand::Edit:

Edit\_Person(head, tail);

break;

case MenuCommand::Delete:

Delete\_Person(head, tail);

break;

case MenuCommand::Show\_All:

Print(head);

break;

case MenuCommand::Search:

Search(head);

break;

case MenuCommand::Exit:

return;

default:

throw "Wrong command!";

break;

}

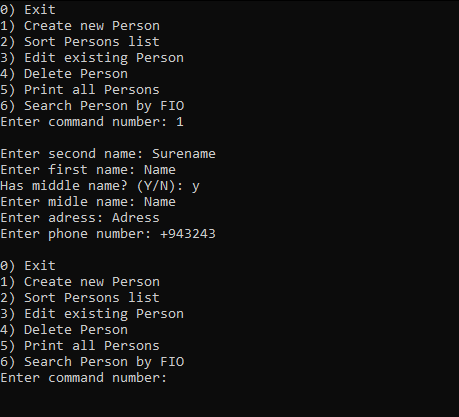
cout << endl;

Update\_PersonMenu(head, tail);

}

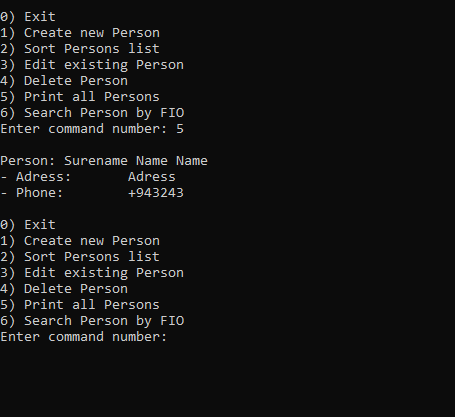
* 1. **Тестирование**

Первым на очереди «Добавление записи».



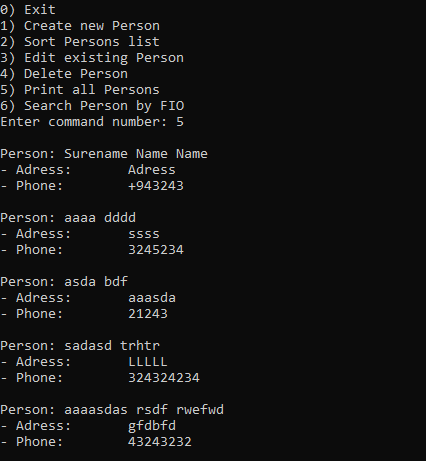
Всё работает корректно.

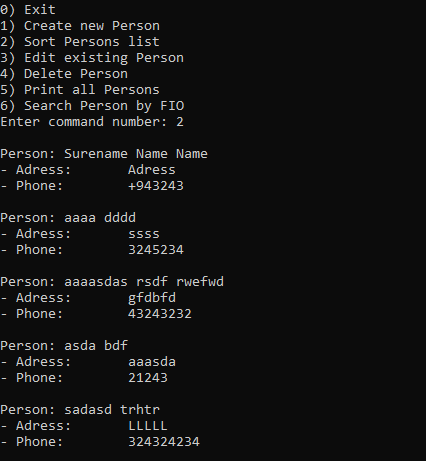
Теперь, выведем все записи.



Записи выводятся корректно.

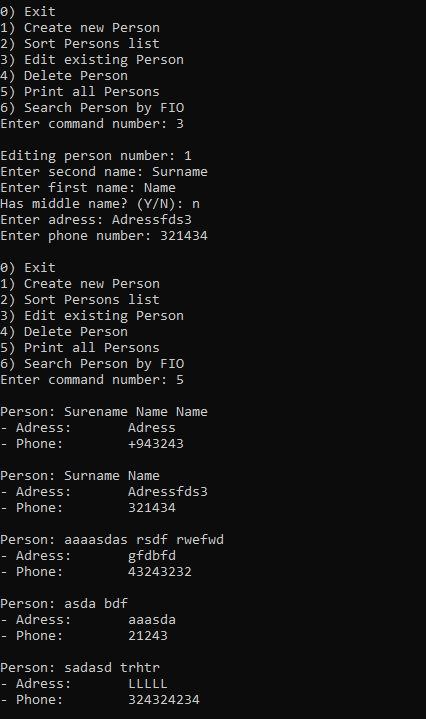
Теперь попробуем отсортировать записи.





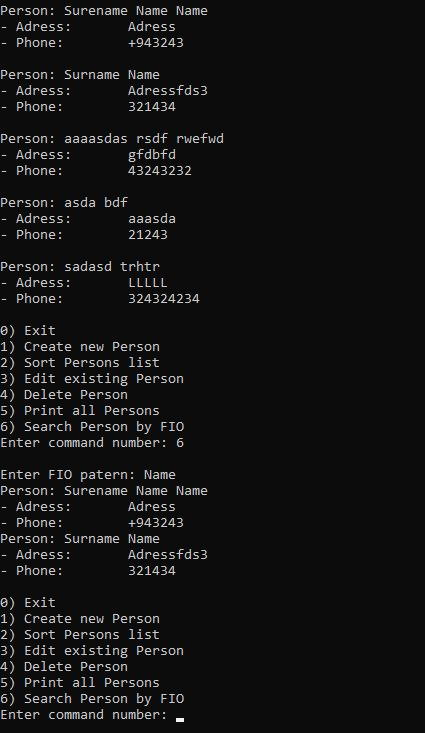
После повторного вывода записей видно, что записи были отсортированы корректно.

Теперь заменим запись под номером «2».



Запись успешно заменилась и теперь там находятся другие данные.

Ну и наконец проверим крайний пункт меню «Поиск по заданному параметру», ну или иначе говоря выборка.



Выборка была произведена по имени «Name» и были выведены записи в которых параметр совпадает с полем «Имя».

**Заключение**

В данной работе я познакомился со сложными структурами данных, понял методы и приёмы обращения со списками. Подкрепил это на практике, переработав прошлую лабораторную работу под формат списка.