УДК 004.658:614.253.8

РАЗРАБОТКА СИСТЕМЫ ХРАНЕНИЯ ДАННЫХ С КОНТРОЛЕМ ЦЕЛОСТНОСТИ

Янченко И.В., Кокова В.И., Дмитриченко Д.А.

Хакасский технический институт – филиал ФГАОУ ВО «Сибирский федеральный университет», Абакан, e-mail: inna-wind@mail.ru

Представлены основные результаты исследования в сфере разработки базы данных с возможностью контроля целостности информации, одного из трех защищаемых свойств информации (доступность, целостность, конфиденциальность). Реализованная идея комплекса методов для контроля целостности может быть применена для любой предметной области, в данной работе использована такая предметная область, как деятельность регистратора медицинского центра. Материалом для работы послужили нормативно-правовые документы в сфере оказания медицинских услуг, информация о бизнес-процессах медицинского центра, связанных с взаимодействием пациентов и медицинского регистратора. Модель основных функций пользовательского интерфейса представлена на языке функционального моделирования в нотации IDEF3, процессы с точки зрения хранения данных в нотации DFD. Приведены примеры необходимых функций, написанных на языке C++, использованных для обеспечения требования контроля целостности. При реализации системы использован комплекс методов, в который вошли двусвязный список, метод «канарейка в угольной шахте», контрольный список в начале и конце строки данных, хэш-таблицы. Практическим результатом исследования является созданный прототип информационной системы хранения данных с контролем целостности и возможностью поиска информации на примере данных о пациентах медицинского центра.

Ключевые слова: медицинский регистратор, база данных, моделирование, DFD, IDEF3, целостность, контроль данных, канарейка, контрольная сумма, хэш-функция, интерфейс

THE DEVELOPMENT OF SYSTEM TO STORE DATA WITH INTEGRITY CONTROL

Yanchenko I.V., Kokova V.I., Dmitrichenko D.A.

Khakas Technical Institute – the branch of Siberian Federal University, Abakan, e-mail: inna-wind@mail.ru

The author presents the main results of the research in the development of the database which allows to control the integrity of information as one of the three protected properties of information (accessibility, integrity, confidentiality). The realized idea of the range of methods to control the integrity of data can be used in any subject area. In this article the subject area used is the work of medical receptionist. The material for the work was regulatory and legal documents in the field of medical services, information about the business processes of the medical center related to the interaction of patients and the medical receptionist. The model of the main functions of the user interface is presented in the language of functional modeling in the notation of IDEF3, processes from the point of view of data storage are presented in the notation of DFD. The article gives axamples of programming the necessary functions of the C++ language, program codes used to ensure the requirement of integrity control. To realise the system we used the range of methods including doubly linked list, method of “canary in a coal mine”, control list in the beginning and end of the data line, hash tables. The practical result of the research is a prototype for an information storage system with integrity control and the ability to search for information in the patient data of a medical center.

Keywords: medical receptionist, database, modelling, DFD, IDEF3, integrity, data control, canary, checksum, hash function, interface

Современный медицинский центр у многих людей ассоциируется с точностью, аккуратностью и вниманием к пациентам. Во многом этому способствует наличие в нем электронного архива или базы данных, благодаря которым персонал может быстро найти всю необходимую информацию о пациенте. Специализированные медицинские информационные системы, хотя и предоставляют обширные возможности для учета пациентов, тем не менее зачастую по тем или иным причинам не подходят небольшим частным медицинским центрам. Таким образом, существует потребность в относительно дешевом временном решении, предшествующем переходу на специализированную медицинскую информационную систему.

Объектом исследовательской работы является деятельность регистратора медицинского центра. Предметом работы является система хранения и поиска данных о пациентах медицинского центра.

Целью исследования является создание системы хранения данных с контролем целостности, а также возможностью поиска информации. Для достижения поставленной цели необходимо выполнить следующие задачи: провести сбор и анализ информации о деятельности медицинского регистратора; определить цель и задачи системы хранения данных пациентов; выбрать программные средства разработки системы хранения данных; разработать систему хранения данных пациентов, провести ее тестирование и отладку.

Исследование выполнено при реализации педагогических условий формирования карьерных компетенций студентов, обеспечивающих мотивационно-ценностную и когнитивную основу формирования карьерной компетентности через обогащение содержания образования; актуализирующих деятельностную и рефлексивно-оценочную основу формирования карьерной компетентности посредством использования процессуальных педагогических технологий [1].

Материалы и методы исследования

Материалом работы послужили нормативно-правовые документы в сфере оказания медицинских услуг, информация о бизнес-процессах медицинского центра, связанных с взаимодействием пациентов и медицинского регистратора.

В работе использованы методы структурного и объектно-ориентированного моделирования при анализе предметной области, проектировании интерфейса пользователя в нотации IDEF3 и моделировании потоков данных DFD.

При реализации практической части исследования использованы:

– Windows Forms – интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NETFramework;

– язык программирования C# для создания графического интерфейса пользователя;

– язык программирования C++ для разработки базы данных.

Результаты исследования и их обсуждение

На аналитическом этапе исследования изучены нормативно-правовые документы и бизнес-процессы медицинского центра, связанные с взаимодействием пациентов и медицинского регистратора. Согласно приказу Министерства здравоохранения и социального развития Российской Федерации от 23 июля 2010 г. № 541н «Об утверждении Единого квалификационного справочника должностей руководителей, специалистов и служащих» должностные обязанности медицинского регистратора: ведет регистрацию больных, обратившихся в медицинскую организацию для получения медицинских услуг; обеспечивает хранение и доставку медицинских карт в кабинет врача; участвует в оформлении и регистрации листков нетрудоспособности [2].

Требования к квалификации медицинского регистратора: среднее профессиональное образование по профилю выполняемой работы без предъявления требований к стажу работы или среднее (полное) общее образование и дополнительная подготовка по направлению профессиональной деятельности не менее 6 месяцев без предъявления требований к стажу.

Таким образом, медицинский регистратор не обязан иметь профессиональных компетенций по ведению базы данных, поэтому интерфейс системы хранения данных о пациентах должен быть максимально понятным и простым.

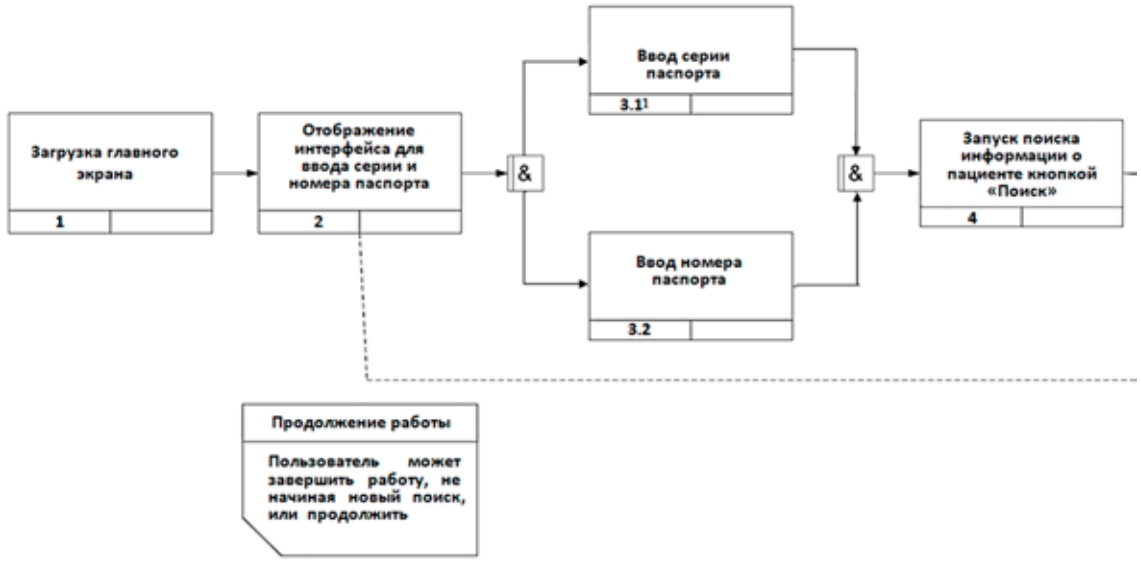
Графический интерфейс пользователя разработан с помощью Windows Forms на языке C#, он будет обращаться к базе данных, разработанной на языке C++. Механизм взаимодействия можно описать следующим образом: пользователь вводит данные или запрос в систему; система перенаправляет данные или запрос к базе данных; база данных возвращает ответ; система возвращает результат пользователю. Концепцию пользовательского интерфейса системы хранения данных опишем на языке функционального моделирования. Нотация IDEF3 позволяет описать процесс работы пользователей с информационной системой. Модель представлена на рис. 1.

После загрузки главного экрана отображается интерфейс для ввода серии и номера паспорта. Пользователь системы может сразу завершить работу по своему желанию, минуя следующие шаги работы с программой. В случае, если пользователь желает найти информацию о пользователе, он обязательно («асинхронное И») должен ввести серию и номер паспорта, после завершения ввода он может запустить процесс поиска с помощью одноименной кнопки. В этот момент система автоматически проверит целостность данных в базе данных, однако более подробно это будет рассмотрено при декомпозиции блока 6.

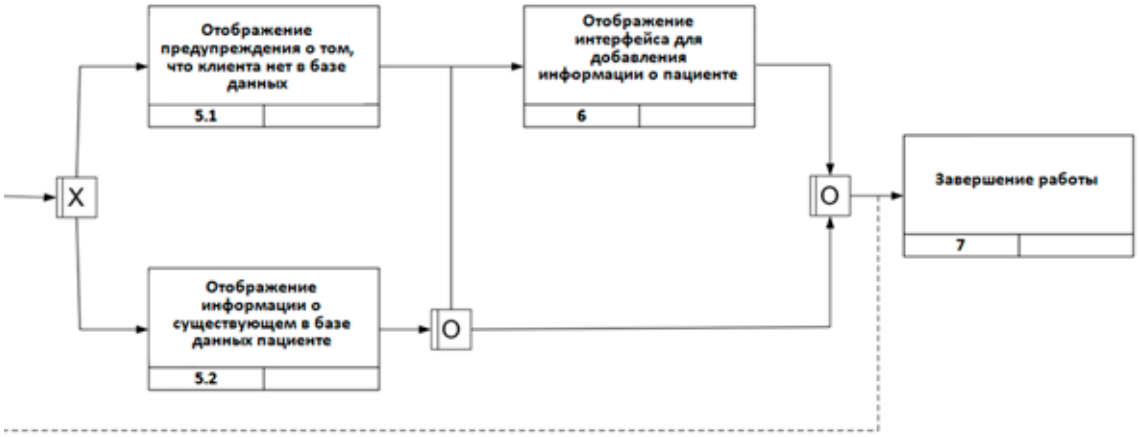
После завершения поиска может произойти одно из двух событий: отображение предупреждения о том, что пациента нет в базе, или отображение информации.

Если было отображено предупреждение, то пользователю системы будет предложено добавить информацию о новом пациенте с помощью специального интерфейса. При необходимости пользователь может перейти из блока 5.2 с информацией о пациенте к блоку 6 с возможностью изменить или добавить дополнительную информацию. Затем пользователь может или вернуться к поиску, или завершить работу с системой.

Рассмотрим более подробно блок 6. Его декомпозиция представлена на рис. 2.



Лист 1



Лист 2

Рис. 1. Модель действий пользователя в нотации IDEF3

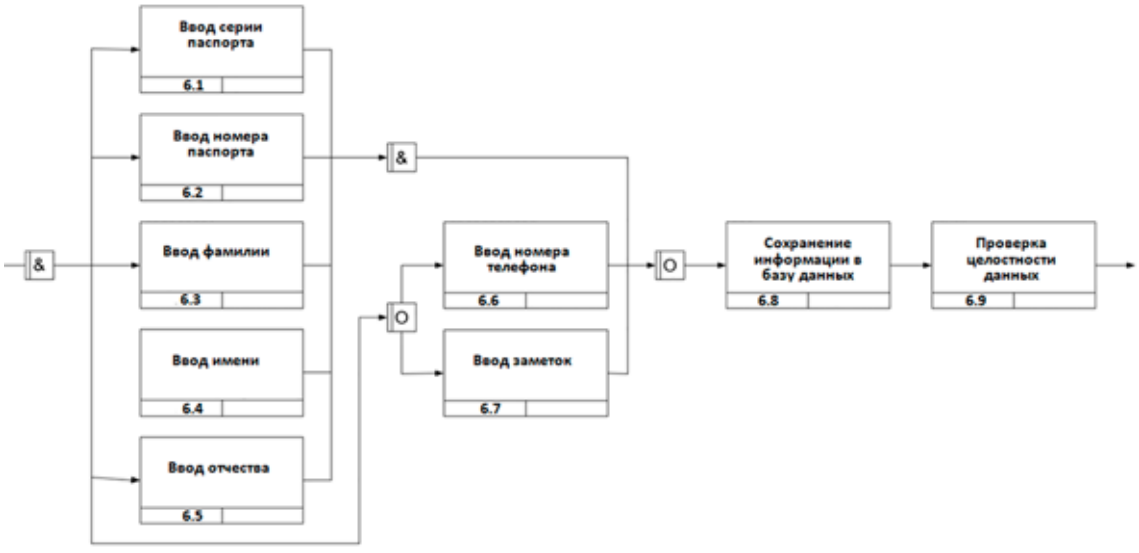


Рис. 2. Декомпозиция блока 6 IDEF3

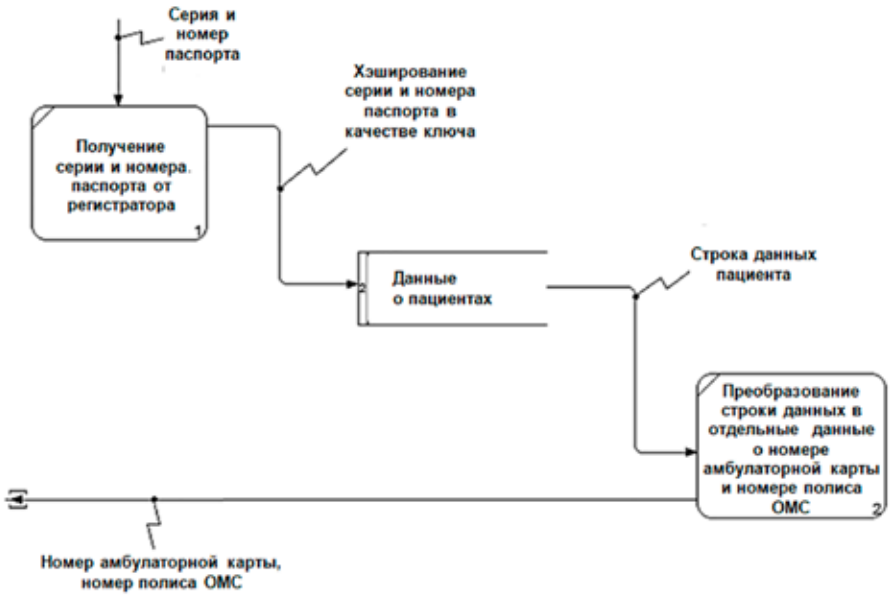


Рис. 3. Модель потоков данных «ИС Пациент»

Структура интерфейса блока 6, предлагающего пользователю добавить или изменить информацию о новом или существующем пользователе, представляет собой совокупность обязательных и необязательных для заполнения полей. К первой группе относятся поля с блока 6.1 до 6.5, ко второй – с блока 6.6 до 6.7, это демонстрирует совокупность операторов «асинхронное И» и «асинхронное ИЛИ». После заполнения всех обязательных полей пользователь может сохранить информацию в базу данных, нажав одноименную кнопку, после этого система автоматически проверит целостность данных в базе. Проверка целостности данных происходит каждый раз при обращении к базе, а именно при загрузке (блок 1), при поиске (блок 4) и при сохранении (блок 6.9).

Для понимания процессов потока данных в системе использована нотация DFD. Модель потоков данных «ИС Пациент» представлена на рис. 3. Регистратор передает в систему данные о серии и номере паспорта пациента, данные попадают в процесс 1, который получает их и преобразует в электронную форму. Далее он пропускает данные через хеш-функцию и отправляет получившийся ключ в базу данных «Данные о пациентах». База данных по ключу находит строку данных пациента и передает ее в блок 2. Этот блок разбивает строку данных на удобный для пользователя вид и возвращает результат пользователю.

В системе предусмотрена файловая модель базы данных для реализации контроля целостности данных, состоящая из одного файла. Сущностью является «Пациент» с атрибутами: серия паспорта; номер паспорта; фамилия; имя; отчество; номер телефона; заметки. Данные сущности фиксируются в файле с расширением .txt в виде строки.

Прежде всего, был создан стандартный двусвязный список средствами языка C++. Детально рассмотрим составляющие его элементы. Структура (класс) «node»:

```
typedef struct node {
        #ifndef FAST_LIST
        int _can1;
        #endif
        type _data;
        link _next;
        link _prev;
        #ifndef FAST_LIST
        size_t _checksum;
        int _can2;
        #endif
        node(typedata = 0):
          _data(data)
        {
                _next = _prev = NULL;

                #ifndef FAST_LIST
                  _can1 = _can2 = CAN;
                #endif
        }
        size_t Checksum();
        void OK();
        ~node()
        {
                _data = 0;
                _next = _prev = NULL;
        }
        friend ofstream&operator<<(ofstream
&out, const node&data);
        friend   ifstream&operator>>(ifstream
&in, node&data);
}item;
```

С помощью директивы ifndef можно либо включить, либо выключить «Быстрый список», другими словами, если нужно, при создании списка программа будет игнорировать все, что находится внутри директивы.

Одной из особенностей программы является переполнение буфера стека и опасность (или возможность, если эксплуатируется данная уязвимость) перезаписи данных в стеке.

В настоящее время появилось множество способов защиты от переполнения буфера стека, которые были внедрены в компиляторы и операционные системы. Рассмотрим метод с «канарейкой», который используется в качестве базовой защиты от переполнения. Такое название метод получил из-за аналогии с назначением канареек для безопасности в угольной шахте [3]. Как канарейка в угольной шахте, списковые канарейки предупреждают программу о том, что что-то не так, что позволяет программе завершить работу до того, как произойдет какая-либо вредоносная операция. Это делается следующим образом: выделяется место для локальных переменных; в начало и в конец элемента списка добавляется длинное случайное число, это число невозможно узнать, если не иметь исходный код программы; перед возвратом функции происходит проверка случайного числа, если канарейка стека была перезаписана атакой переполнения буфера стека, программа завершится.

Конечно, этот метод защиты далек от идеала. Существует множество атак, которые потенциально позволяют обойти эту защиту, например, уязвимость форматированной строки, однако в случае регистратора небольшого медицинского центра такие скоординированные атаки маловероятны.

После записи первой канарейки идет запись строки данных, затем указатели на предыдущий и следующий элемент. Указатели нужны для того, чтобы в итоге получился именно двусвязный список.

Как и односвязный список, двусвязный допускает только последовательный доступ к элементам, но при этом дает возможность перемещения в обе стороны [4]. В этом списке проще производить удаление и перестановку элементов, так как легкодоступны адреса тех элементов списка, указатели которых направлены на изменяемый элемент. В качестве преимуществ двусвязного списка выделим эффективное (за константное время) добавление и удаление элементов, то, что размер ограничен только объемом памяти компьютера и разрядностью указателей, а также динамическое добавление и удаление элементов [5].

Далее записывается контрольная сумма, которая показывает, изменялись эти данные или нет. Если понимать, что это такое и как этим пользоваться, можно проверить подлинность файла и обезопасить себя от подделок, вирусов и шпионов. Если известны контрольная сумма и алгоритм ее нахождения, можно проверить файл на целостность – загрузился ли файл целиком и вообще тот ли это файл, что нужен.

Рассмотрим функцию, реализующую проверку всех этих контрольных данных, функцию OK:

```
void node::OK()
{
    #ifndef FAST_LIST
    if (_can1 != CAN || _can2 != CAN || 
Checksum() != _checksum)
    {
        cout<<”Error not OK\n”;
        exit(1);
    }
    #endif
    return;
}
```

В данной функции тоже используется директива ifndef, внутри нее сравниваются значения канареек с контрольными, а также вызывается функция Checksum(), результат выполнения которой должен сойтись с контрольным значением. Исследуем Checksum() более подробно:

```
size_t node::Checksum()
{
    #ifndef FAST_LIST
    if (_next != NULL && _prev != NULL)
    {
        size_t a = (size_t)_next, b = 
(size_t)_prev;
        return ((a * b) / (a + b)) + 21442;
    }
    elseif (_next != NULL)
    {
        size_t c = (size_t)_next;
        return ((c * c) / (c + 15616)) + 21442;
    }
    elseif (_prev != NULL)
    {
        size_t d = (size_t)_prev;
        return ((d * d) / (d + 15616)) + 21442;
    }
    #endif
    return 0;
}
```

Функция проверяет ссылки на целостность. Если существует ссылка на прошлый или следующий элемент списка, она

не должна изменить свое значение, иначе произойдет критический сбой, который может привести к потере данных. В функции отдельно рассматриваются случаи, когда элемент является первым в списке или же последним.

Теперь рассмотрим частный случай ошибки, если список пустой. Для обнаружения этой проблемы использована специальная функция ListOK():

```
void list::ListOK()
{
      if (_head != NULL&& _tail != NULL)
            return;
      cout<<»Список не найден.\n»;
      exit(1);
}
```

Данная функция относится к методам класса list. Вернемся к методам класса node и рассмотрим, как работает оператор вывода в выходной поток для работы с файлами:

```
ofstream&operator<<(ofstream&out, const
node&data)
{
      out<<data._can1 <<'&'<<data._data<<'&'
<<data._checksum<<'&'<<data._can2 <<'\n';
      return out;
}
```

Этот оператор берет константную ссылку на экземпляр класса node, далее выводит в поток данные в соответствии со специальным порядком, это необходимо, чтобы потом можно было корректно загрузить данные из файла.

Оператор ввода из входного потока для работы с файлами:

```
ifstream&operator>>(ifstream&in,
node&data)
{
      char buf[50] = {};
      in.getline(buf, 50, '&');
      data._can1 = atoi(buf);
      in.getline(buf, 50, '&');
      string str = buf;
      char* cstr = new char[str.length() + 1];
      strcpy(cstr, str.c_str());
      data._data = cstr;
      in.getline(buf, 50, '&');
      data._checksum = 0;
      in.getline(buf, 50, '&');
      data._can2 = atoi(buf);
      return in;
}
```

Происходит считывание данных именно в том порядке, в котором они были записаны в файл. Перейдем к структуре list. Эта структура представляет собой двусвязный список, элементами которого являются экземпляры класса node. Класс list:

```
typedef struct list {
      link _head;
      link _tail;
      int _size;
      list():
      _size(0)
      {
            _head = _tail = newnode();
      }
      void ListOK();
      void PrintList();
      void InsertAfter(int index, type data);
      void InsertBefore(int index, type data);
      void ArrayToList(type *Array, int size);
      void Delete(int index);
      int FindString(type data);
      link FindPtr(type data);
      int FindIndex(type data);
      void Swap(link ptr1, link ptr2);
      void BubbleSort();
      ~list()
      {
            link tmp = _head;
            link next = NULL;
            while (tmp != NULL) {
                  next = tmp->_next;
                  delete tmp;
                  tmp = next;
            }
            _head = _tail = NULL;
            _size = 0;
      }
}list;
```

В данном фрагменте кода уделим внимание конструктору и деструктору класса. Так как для элементов списка память выделяется посредством оператора new, то для того, чтобы избежать утечки памяти, требуется после удаления списка освободить всю выделенную память. Эту задачу и берет на себя деструктор, поэлементно освобождая память.

Перейдем к описанию проектирования хеш-таблицы. Класс hash_table:

```
const int table_size = 597;
typedef class hash_table
{
public:
      list *hashed[table_size];
      size_t(*hash_f)(char*);
//public:
      hash_table(size_t(*hash_funk)(char*)) :
            hash_f(hash_funk)
      {
```

```
        for (int i = 0; i<table_size; i++)
        {
            hashed[i] = new list;
        }
    }
    void Register(char* data);
    void Remove(char* data);
    void DumpToFile(const char *file);
    void SaveToFile();
    void LoadFromFile();
    ~hash_table()
    {
        for (int i = 0; i<table_size; i++)
        {
            delete hashed[i];
        }
    }
}hash_table;
```

Как видно, важной частью конструктора является передача в него хеш-функции.

Размер таблицы был выбран случайно, так как за обработку коллизий будет отвечать двусвязный список. Более подробно рассмотрим методы этого класса, hash_table::Register:

```
void hash_table::Register(char* data)
{
    int x = hash_f(data) % table_size;
    if (hashed[x]->FindString(data) == 0)
hashed[x]->InsertAfter(1, data);
    cout<<data<<»\n»;
}
```

Данный метод осуществляет стандартную проверку на наличие добавляемого элемента в таблице, а затем реализует вставку в начало методом списка. Опустим некоторые методы, так как они реализованы стандартным образом и не имеют большого значения для данной программы.

Сохранение и загрузка данных являются методами хеш-таблицы. Рассмотрим подробно сохранение таблицы в файл, hash_table::SaveToFile():

```
void hash_table::SaveToFile()
{
/* пишем в файл целым классом как одним
блоком */
ofstream out(“base.txt”, ios::trunc);
for (size_t i = 0; i<table_size; i++)
   {
if (hashed[i]->_size != 0)
      {
link cur = hashed[i]->_head;
while (cur)
         {
if (cur != NULL) cur->OK();
out<< *cur;
cur = cur->_next;
         }
      }
else
      {
out<<’\n’;
      }
   }
out.close();
}
```

Сохранение в файл начинается с открытия этого самого файла, затем запускается цикл по всем элементам таблицы. Если элемент содержит данные, другими словами, его размер больше нуля, тогда начинается запись в файл элементов списка, хранящегося в данной ячейке таблицы. После прохода цикла по всей таблице программа закрывает поток вывода и заканчивает выполнение функции вывода.

Рассмотрим метод, позволяющий загружать данные из файла, hash_table::LoadFromFile():

```
void hash_table::LoadFromFile()
{
ifstream in(«base.txt»);

if (!in) // проверка - существует ли такой
файл
throw(“File does not exist!”); // файл
не существует
else
   {
char ch = ‘\0’;
while (true)
      {
ch = in.get();
if (in.eof() || in.fail()) break;
else if (ch != ‘\n’) {
in.putback(ch);
link nodeOK = new node();
in>> *nodeOK;
nodeOK->OK();
Register(nodeOK->_data);
delete nodeOK;
         }
      }
in.close();
}
}
```

Данный метод использован для проверки, существует ли открываемый файл, если он существует, то запускается цикл, на каждом шаге которого считывается один символ из потока входа. Функция проверяет, получилось ли считать этот символ, таким образом, она сможет выйти из цикла при достижении конца файла.

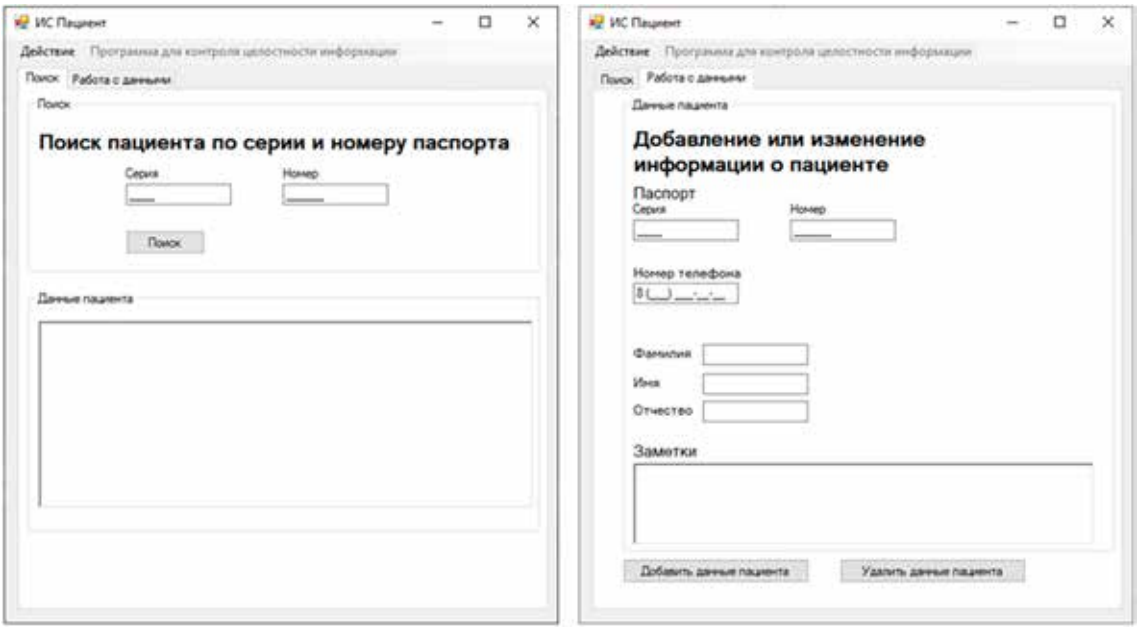


Рис. 4. Скриншоты вкладок UI системы

Далее, в соответствии с функцией сохранения в файл, программа выполняет цикл до тех пор, пока не считает символ, отличный от символа переноса строки. Когда это происходит, программа возвращает символ в поток входа и создает экземпляр класса node, после этого запускается перегруженный оператор ввода из входного потока, который записывает данные из файла в созданный экземпляр класса node. После этого запускается метод OK(), который проверяет, не изменились ли канарейки и контрольная сумма. Далее считанные данные регистрируются в хеш-таблицу с помощью одноименного метода. Перед завершением шага цикла память, выделенная для экземпляра класса node, освобождается.

Для создания интерфейса использовались стандартные компоненты, предоставляемые Windows Forms. Вкладки интерфейса системы представлены на рис. 4.

Как видно, пользователь может ввести серию и номер паспорта в специальные поля ввода. Эти поля контролируют ввод, чтобы пользователь не вводил ничего, кроме цифр, при этом количество цифр ограничено в соответствии с длиной серии (4 цифры) и номером (6 цифр) паспорта. После ввода пользователь может нажать кнопку «Поиск», и результат будет выведен в поле вывода снизу.

Форма для добавления серии и номера паспорта аналогична форме на главном экране. После этого следует поле для ввода номера телефона. Это поле тоже поддерживает ввод только цифр в необходимом количестве. Далее идут поля для ввода фамилии, имени и отчества. И последним является поле «Заметки», в которое можно вводить любую необходимую информацию о пациенте, например номер полиса ОМС, наличие аллергии и т.д. Далее идут кнопки «Добавить данные пациента» и «Удалить данные пациента», выполняющие одноименные действия.

Заключение

В процессе выполнения исследования на основе анализа предметной области решены поставленные задачи: выбраны программные средства разработки информационной системы хранения данных, определена архитектура и средства разработки программного продукта; для разработки системы хранения данных с контролем целостности и возможностью поиска информации о пациентах спроектированы модели: действий пользователя в нотации IDEF3, диаграмма потоков данных (DFD); реализованы функции сохранения и загрузки данных, создан интерфейс системы, кроме того, проведено тестирование и отладка программного продукта. Таким образом, получен практический результат – создан прототип информационной системы хранения данных с контролем целостности и возможностью поиска информации о пациентах.

В перспективе планируется расширение функций информационной системы хранения данных с контролем целостности, а также применение методики в случае использования реляционной базы данных.

Список литературы

1. Янченко И.В. Формирование карьерной компетентности студентов в профессиональном образовании: дис. ... канд. пед. наук. Красноярск, 2013. 255 с.

2. Приказ Министерства здравоохранения РФ от 23 июля 2010 г. № 541н «Об утверждении Единого квалификационного справочника должностей руководителей, специалистов и служащих» (с изменениями и дополнениями). [Электронный ресурс]. URL: https://base.garant.ru/12178397/ (дата обращения: 11.08.2022).

3. Эксплуатация уязвимостей исполняемых файлов для новичков: привилегии и обработка исключений. [Электронный ресурс]. URL: https://tproger.ru/translations/stack-protection-bypass/ (дата обращения: 11.08.2022).

4. Даниленко А.Н. Структуры данных и анализ сложности алгоритмов: учебное пособие. Самара: Издательство Самарского университета, 2018. 76 с.

5. Контрольная сумма: что это и почему это важно. [Электронный ресурс]. URL: https://thecode.media/crc32/ (дата обращения: 11.08.2022).