СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4

1. ТЕМА КУРСОВОЙ РАБОТЫ 5

2. МЕТОДИКА ВЫПОЛНЕНИЯ РАБОТЫ 5

3. ТРЕБОВАНИЯ К ПРОГРАММЕ 20

4. ТИПОВАЯ СТРУКТУРА ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ 20

5. РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ РАЗДЕЛОВ КУРСОВОЙ РАБОТЫ 22

6. ТРЕБОВАНИЯ К ОФОРМЛЕНИЮ ОТЧЕТА 23

7. ПОРЯДОК ЗАЩИТЫ РАБОТЫ 24

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ 24

Приложение 1 Образец оформления титульного листа 25

Приложение 2 Пример описания интерфейса программы 26

Приложение 3 Пример описания классов 27

Приложение 4 Пример спецификации состава программы 29

Приложение 5 Пример описания модулей программы 30

Приложение 6 Варианты заданий 1-20 31

ВВЕДЕНИЕ

Методические указания представляют требования к выполнению курсовой работы дисциплины «Объектно-ориентированное программирование» для направления обучения бакалавриата 09.03.01 «Информатика и вычислительная техника», 09.03.02 – «Информационные системы и технологии».

Курсовая работа является частью программы изучения дисциплины, проводится согласно графику учебного процесса и имеет целью оценить:

- теоретические знания, практические умения и навыки в области объектно-ориентированного программирования, теории автоматов и структур данных, технологий программирования;
- умение творчески решать поставленную задачу, применяя теоретические, практические знания и умения;
- умение выполнить самостоятельно и в срок заданную работу; умение самостоятельно работать со специальной литературой.

Для выполнения курсовой работы обучающиеся должны быть знакомы с математическими основами программирования, структурами данных, теорией автоматов и формальных языков.

В рамках выполнения курсовой работы необходимо разработать программу в соответствии с предложенным вариантом задания, подготовить пояснительную записку о проделанной работе, и защитить работу перед преподавателем.

График выполнения курсовой работы соответствует графику аттестации семестра. Оценка за курсовую работу выставляется в зачетную книжку.

1. ТЕМА КУРСОВОЙ РАБОТЫ

Цель работы - научиться разрабатывать и применять шаблон класса для решения задач.

Задание

A. Создать шаблон класса «двусвязный линейный список». Реализовать в нем следующие методы:

- все виды конструкторов – по умолчанию, с параметрами, копирования; деструктор;
- итераторы класса – перемещают указатель узла на следующий Next() или предыдущий Pred() узел списка, и возвращают значение информационного поля списка. Например, Node* temp=temp->next;
- добавление в конец и в начало списка; вставка элемента в список по правилу;
- присваивание нового значения элементу списка и/или перегрузка операции (=);
- удаление элемента в списке; очистка списка;

- поиск элемента в списке по номеру - перегрузка операции ([ ]) - и по значению информационного поля;
- просмотр и вывод всех элементов списка в прямом и обратном порядке;
- просмотр и вывод указанного элемента;
- дополнительные операции, обеспечивающие необходимую функциональность шаблона как контейнерного класса.

B. Решить вариант задачи, используя шаблон класса «двусвязный линейный список »

Варианты заданий в Приложении 6. Варианты заданий дают описание информационного класса, его объекты необходимо организовать в динамический двусвязный список посредством шаблона класса (задание А) и продемонстрировать работу всех методов двусвязного списка на информационном объекте класса. И требуется организовать также обработки списка объектов по варианту задачи.

2. МЕТОДИКА ВЫПОЛНЕНИЯ РАБОТЫ

Синтаксис определения шаблонного класса

Язык C++ поддерживает метод повторного использования структуры класса. Этот инструмент называется шаблонный класс. Вместо класса с фиксированным типом компонентов, создается класс, где тип компонентов интерпретируется как параметр класса.

Шаблоны классов предоставляют создавать параметризированные классы. Параметризированный класс создает семейство родственных классов, которые можно применять к любому типу данных, передаваемому в качестве параметра.

Синтаксис описания шаблона:

```
template <class имя_параметра, … >
class   имя {              private:
…          public:
            …
};
```

Каждый параметр шаблона в угловых скобках представляет собой "заполнитель" для типа. Параметризованный класс (родовой класс) может иметь любое количество параметров типа. Несколько шаблонных параметров в списке разделяются запятыми.

```
template <class T1, class T2, class T3> //три параметра типа
class Triple;                           // объявление класса
```

Как и в других классах C++ , зарезервированное слово **class** в списке параметров имеет иное значение, чем в других контекстах. Оно показывает, что

идентификатор, расположенный за ним, является "заполнителем" фактического типа. Этот тип не обязательно должен быть классом. Он также может быть любым встроенным типом. Кроме того, допускается использование конкретных типов параметров выражений.

```
template <class Type, int size>   // тип и значение  class
Array;
```

Это подобно параметрам функций — в момент реализации класса Array значение указанного типа (в данном случае int) должно предоставляться клиентской программой.

В качестве параметров могут использоваться типы, шаблоны, переменные. Область действия параметра шаблона – от точки описания до конца шаблона, поэтому параметр можно использовать для описания следующих за ним, например:

```
template <class T, T* p, class U = T>
class X {
…
};
```

В данном примере типу *U* по умолчанию определен тип *T*, а указатель имеет ранее определенный тип *T*.

Пример 1. Шаблон класса «стек». Тип параметра обозначается именем Type, определенным программистом.

```
template <class Type> class
Stack
{
private:
Type* items;                    // компонент класса, где тип – это параметр шаблона
int size; int top; public:
Stack(int sz);                  //конструктор
Stack();                        // конструктор по умолчанию
Stack(const Stack<Type> &L);    //конструктор копирования
~Stack();                       // деструктор
// основные операции
       void push(Type element);
Type pop(); Type
peek();
int Top() { return top+1; }; bool
isEmpty();
// перегрузка операции сложения - конкатенации двух стеков
Stack<Type>& operator + (Stack<Type> &L);
// перегрузка операции > bool
operator > (Stack<Type> &L);
Type* info() { return items;}; };
```

Методы шаблона класса автоматически становятся шаблонами функций. Если метод описывается вне шаблона, то его заголовок – это заголовок шаблона-функции.

```
//-------------- конструктор ---------------------------
 template <class Type>
Stack<Type>::Stack(int sz)
```

```
{ size =
sz; top =
-1;
items = new Type[size];
}
//----- конструктор копирования (операция присваивания =)------ template
<class Type>
Stack<Type>::Stack(const Stack<Type> &L)
{
items = new Type[L.size];
size = L.size; top =
L.top;
for (int i = 0; i <= top; i++) items[i]
= L.items[i];
}
//-------------- конструктор по умолчанию -------- template
<class Type>
Stack<Type>::Stack()
{ size =
100; top =
-1;
items = new Type[size];


}
//---------------деструктор-----------------


template <class Type> Stack<Type>::~Stack()
{ delete[]
items;
}
//----------------вывод вершины стека--------
template <class Type> Type Stack<Type>::peek()
{
return items[top];
}
//----------------вставка  значение в стек----


template <class Type>
void Stack<Type>::push(Type element)
{ if (top == size-1)
return; items[++top] =
element;
}
//---------------удаление значения из стека----
 template  <class  Type>
Type  Stack<Type>::pop()
{
if (!isEmpty()) return items[top--];
}
//----метод - предикат, проверка стека на пустоту--------
 template <class Type> bool
Stack<Type>::isEmpty()
{
return (top == -1);
}
//---- перегруженная операция  сложения двух стеков------


template <class Type>
```

```
Stack<Type>& Stack<Type>::operator + (Stack<Type> &L)
{ int i =
0;
Stack<Type>* S = new Stack<Type>(size + L.size);
 for (; i <= top;
i++)
S->push(items[i]);


for (i = 0; i <= L.top; i++)
S->push(L.items[i]);


return *S;
}
//---------перегруженная операция  сравнения двух стеков------


template <class Type>
bool Stack<Type>::operator > (Stack<Type> &L)
{ return (top >
L.top);
}
```

Создание объекта шаблонного класса называется реализацией (instantiation). Оно подобно созданию объекта любого класса в C++. Когда клиентская программа приписывает значения конкретному объекту шаблонного класса, она должна предоставить для каждого шаблонного параметра аргумент фактического типа. Шаблон Stack не является классом, это всего лишь шаблон. Он не поддерживает прямое создание объектов Stack. Объект Stack без указания фактического типа при вызове функции является ошибкой.

Фактический тип определяется при реализации шаблона как имя типа в угловых скобках, которые добавляются к имени шаблонного класса. Имя объекта задается таким же образом, как и для обычных классов. Параметры конструкторов используются там, где необходимо.

```
Stack<int> is(50); // стек целых длиной 50 значений
Stack<char> cs(200); // стек символов длиной в 200 символов
```

Рассмотрим использование шаблона стека на разных типах данных от простых базовых типов до более сложных объектных типов.

Программа – клиент для класса стек, реализованного на типе int и double представлена ниже.

```
int main()
{
Stack<int>  *stk2; stk2 = new Stack<int>(10);
stk2->push(56);   stk2->push(67);   cout <<
stk2->peek()<<endl;


Stack<double> stk1(20);         //отрабатывает конструктор с параметрами
stk1.push(23.5); stk1.push(-34.5);   cout << stk1.Top() << endl;
```

```
Stack<double>  stk5 = stk1;    //отрабатывает конструктор копирования
Stack<double> st;              //отрабатывает конструктор по умолчанию
stk5.push(45.1);
  st = stk1 + stk5;              //операция сложения для стека
st.push(56.1); double* m = st.info();
  for (int i = 0; i <st.Top(); i++) cout << m[i] << endl;
cout << st.Top() << endl;


  if (stk1 > st) cout << " stk1 -размер стека
больше";   else  cout << " stk1 -размер стека меньше
";
}
```

Более сложная будет программа – клиент для класса стек, реализованного на объектном типе Point. Описание объектного типа потребует перегрузку операции вывода, в общих случаях требуется перегрузить также операцию сравнения для поиска и разработать методы обработки полей объекта.

```
 class Point
{ private:
int x, y;
//  перегрузку операции помещения в поток вывода объявляем дружественной классу, //  она
должна иметь доступ к закрытым полям класса x и y  friend ostream& operator <<(ostream&
out, const Point& p);  public:
       Point () { }              // конструктор по умолчанию: пустой
       Point (const Point &p)    // конструктор копирования
         {x = p.x; y = p.y;}
       void set (int a, int b)  // установить координаты Point
       { x = a; y = b; }
};
// перегрузка операции помещение в поток вывода ostream&
operator << (ostream& out, const Point& p)   { out << "(" <<
p.x << "," << p.y << ")";    return out ; }
 int
main() {
Point data[5];      // объект Point     Stack<Point>
s(4);  // объект Stack
data[0].set(1, 2); data[3].set(7, 8); data[1].set(3, 4);
data[2].set(5, 6); data[4].set(9, 0);  int n = sizeof
(data)/sizeof (Point);  cout << "Initial data: ";
for (int j = 0; j < n; j++ ) { cout <<data[j] << " "; } cout << endl;
for (int i = 0; i < n; i++) { s.push(data[i]); }  cout << " Inversed
data: ";
while ( !s.isEmpty()) cout << s.pop() << " "; cout << endl;


}
```

C++ поддерживает концепцию специализации при работе с параметрами типа, которые требуют специальной обработки. Для каждого специального класса должен предусматриваться отдельный **специализированный шаблон класса.** Синтаксис описания специализации представляет собой объединение синтаксиса для самого шаблонного класса (в списке параметров шаблона) и инициализации шаблона в клиентской программе (в списке фактических типов).

После описания шаблона класса, помещается полное описание специализированного класса, при этом требуется заново определить все его методы.

```
template <class T>          // удалить класс T из скобок
class Array {…};
                            // добавить <char*> к имени класса
template <>                 // пустой список параметров шаблона
class Array<char*>{ … };  // список фактических типов
```

В методах специализации шаблонов описывается, что следует выполнить для конкретного типа. Специализация шаблона реализуется с использованием того же синтаксиса, что и для объекта шаблонного класса. Объявление специализированного объекта шаблона:

```
Array<char*> a1;
```

Каждая версия класса или функции, создаваемая по шаблону, содержит одинаковый базовый алгоритм. При этом эффективность этого алгоритма в зависимости от типа данных может сильно меняться. Если для какого-либо типа существует более эффективный алгоритм (программный код) можно предусмотреть специальную реализацию отдельных методов, т. е. **специализировать метод.**

```
template <class Data> void
List<Data>::print(){ …};


//специализированное описание
void List<char *>::print({ …};
```

в описании класса

Если создан объект

```
List <char*> Str;
```

то вызван будет метод специализированный.

Методика выполнения работы

Выполнение задания работы предполагает решение двух задач.

A. Создать шаблон класса «двусвязный линейный список». Реализовать в нем указанные методы.

B. Решить задачу по варианту, разработав информационный класс, который использует шаблон класса «двусвязный линейный список» для организации списка своих объектов. Это задание позволит определить качество функциональности шаблона класса двусвязного списка, проверив его на более сложных типах данных. В результате выполнения варианта, возможно, потребуется внести дополнительные методы в класс двусвязного списка, но они должны носить универсальный характер и не зависеть от конкретного типа для решения задачи.

Решение А. Разработка двусвязного динамического списка основано на реализации шаблона класса узла списка Node (Листинг 1) и класса двусвязного списка узлов ListNode (Листинг 2). Проект такого решения представлен на Листинге 1-2.

Листинг 1

```
// Модуль DList.h - шаблон класса узла списка
template<typename T>
class Node {
public:
    T d;
    Node* prev;
    Node* next;
    Node() {
        prev = NULL;
        next = NULL;
    };
};
```

После кода классов в модуле расположите описание методов, которые не определены в классе. Это самостоятельная разработка, основанная на знаниях структур данных и алгоритмов их обработки. Создайте для шаблона класса отдельный модуль и отладьте его работу, проверив все методы на разных типах данных. Модуль в C++ состоит из двух файлов:
- заголовочного, C++ Header file (.h), где располагаются все объявления; -
файла реализации, C++ Source file (.cpp), где представлено описание кодов программ, объявленных в заголовочном файле.

Между собой они связаны командами #include <имя файла>. Этой же командой он подключаются к модулю, где требуются коды данного модуля. Можно модуль реализовать в одном из этих файлов.

Работа с много модульным проектом – необходимый навык программирования и с ним необходимо самостоятельно разобраться. Среда разработки предоставляет инструменты создания многомодульного решения, см. справку и документацию по этим вопросам на соответствующем сайте среды разработки.

Листинг 2

```
// шаблон класса двусвязного списка
template <typename T>
class ListNode {
private:
    Node<T>* head;                          //голова списка
    Node<T>* tail;                          //хвост списка
    Node<T>* current;                       // указатель на текущий узел
    Node<T>* FindNodePos(int i);            // поиск узла по позиции
    Node<T>* FindNode(T d);                 // поиск узла по информационному значению
public:
    // конструктор по умолчанию
    ListNode() :head(NULL), tail(NULL), current(NULL) {};
    // конструктор с параметром информационного значения узла
    ListNode(T d) { AddHead(d); };
    // конструктор с параметрами для заполнение узлов значениями из массива
    ListNode(T arr[], int length);
    // деструктор
    ~ListNode() { Clear(); }

    // методы добавления и вставки узлов, вывода прямого и обратного, удаления,
    // очистки списка и дополнительные
    void AddHead(T d);
    void AddTail(T d);
    void AddNode(T d, int pos);
    void ListHead();
    void ListTail();
    void DeleteHead();
    void DeleteTail();
    void DeleteNode(int i);
    void Clear();
    bool isEmpty();
    // методы работы с итераторами
    T currentNode() { return current->d; }
    T Next();
    T Pred();
    T operator[](int pos) {
        Node<T>* it = FindNodePos(pos);
        return it->d;
    };
    // дополнительные методы, расширяющие функциональность класса
    // чтение/запись в файл элементов списка
    void ReadFromFile(string NameOfFile);
    void WriteToFile(string NameOfFile);
};
```

Решение В. Пусть вариант задания будет следующий. Составить программу, которая содержит динамическую информацию об архиве газет. Сведения о каждой газете содержат:

- название газеты;
- номер газеты;
- дата издания;
- перечень статей (не менее 5);
- тираж;
- ФИО ответственного редактора.

Программа должна обеспечивать:

- начальное формирование архива в виде списка из указанного файла года;
- начальное формирование списка изданий газет текущего года;
- при выпуске очередного номера вводится информация в список текущего года, если наступил новый год, то список сохраняется в файл как архив;
- по запросу выдается перечень статей, тираж, ответственный редактор из архива указанного года или из текущего;
- обеспечить переход от выбранного номера газеты к предыдущему или последующему.

Для данного решения разработаем класс определяющий заданную информационную структуру и дополнительные структуры данных, если они необходимы.

Описание информационной структуры в виде класса позволит определить операции в составе класса, которые сделают информационные данные отдельным типом, который можно обрабатывать контейнерным классом двусвязного списка. *Шаблон класса двусвязного списка должен остаться универсальным для любых типов*.

Проект решения задачи представлен на Листинге 3.

Двусвязный список подключен как модуль DList.h, он остался таким же как описан в Листинге 1-2. Созданы два класса для описания даты Data и описания газеты NewsPaper с набором переопределенных операций помещение (<<) и извлечения (>>) из потока ввода/вывода, сравнения, присваивания, определены методы этих классов. В шаблон класса двусвязного списка добавлены дополнительные методы чтения и записи в указанный файл списка.

Кроме того, консольное приложение имеет меню и позволяет вызывать функции решения.

Листинг 3

```
#include <iostream>
#include <stdlib.h>
#include <vector>
#include <string>
#include <fstream>
#include "DList.h"

using namespace std;

#pragma region Data

class Data {
private:
    int day;
    int month;
    int year;
public:
    Data() :day(1), month(1), year(2019) {}
    Data(int d, int m, int y) :day(d), month(m), year(y) {}
    ~Data() {}
    string data() {
        return  to_string(day) + '.' + to_string(month) + '.' + to_string(year);
    }
    friend ostream& operator << (ostream& out, const Data& obj)
    {
        out << obj.day << " " << obj.month << " " << obj.year;
        return out;
    }
    int getYear() {
        return year;
    }
    friend istream& operator >> (istream& in, Data& obj)
    {
        in >> obj.day;
        in >> obj.month;
        in >> obj.year;

        return in;
    }
    Data& operator = (Data obj) {
        day = obj.day;
        month = obj.month;
        year = obj.year;
        return *this;
    }
};
#pragma endregion
```

```
#pragma region NewsPaper

class NewsPaper {
private:
    string name;
    int num;
    Data date;
    int numArticles;
    vector<string> article;
    int edition;
    string FIOEditor;
```

```
public:
    NewsPaper() : name(""), num(0), date(1, 1, 2019), article(), edition(1), FIOEditor("") {};
    ~NewsPaper() {};
    NewsPaper& operator = (NewsPaper obj) {
        name = obj.name;
        num = obj.num;
        date = obj.date;
        numArticles = obj.numArticles;
        for (size_t i = 0; i < numArticles; i++)
            article.push_back(obj.article[i]);
        edition = obj.edition;
        FIOEditor = obj.FIOEditor;
        return *this;
    };
    friend ostream& operator << (ostream& out, const NewsPaper& obj)
    {
        out << obj.name << endl <<
            obj.num << endl <<
            obj.date << endl <<
            obj.numArticles << endl;
        for (size_t i = 0; i < obj.numArticles; i++) {
            out << obj.article[i].c_str() << endl;
        }
        out << obj.edition << endl <<
            obj.FIOEditor << endl;
        return out;
    }
    friend istream& operator>> (istream& in, NewsPaper& obj)
    {
        in >> obj.name;
        in >> obj.num;
        in >> obj.date;
        for (size_t i = 0; i < obj.article.size(); i++) {
            in >> obj.article[i];
        }
        in >> obj.edition;
        in >> obj.FIOEditor;

        return in;
    }
    void GetInfo();
    void SetName(string def);
    void SetNum(int def);
    void SetData(Data def);
    void SetNumArticles(int def);
    void SetArticle(vector<string>);
    void SetEdition(int def);
    void SetFIOEditor(string def);
    int getYear() {
        return date.getYear();
    }
};
```

```
void NewsPaper::GetInfo() {
    cout << "Name - " << name << endl <<
        "Number - " << num << endl <<
        "Date - " << date.data() << endl <<
        "Article - " << endl <<
        "Edititon - " << edition << endl <<
        "FIO Editor - " << FIOEditor << endl;
};

void NewsPaper::SetName(string def) {
    name = def;
};

void NewsPaper::SetNum(int def) {
    num = def;
};

void NewsPaper::SetData(Data def) {
    date = def;
};

void NewsPaper::SetArticle(vector<string> def) {
    for (size_t i = 0; i < def.size(); i++)
        article.push_back(def[i]);
};

void NewsPaper::SetEdition(int def) {
    edition = def;
};

void NewsPaper::SetFIOEditor(string def) {
    FIOEditor = def;
};

void NewsPaper::SetNumArticles(int def) {
    numArticles = def;
};
#pragma endregion
```

```
void createNewsPaper() {
    NewsPaper paper;
    string line;
    int num, month, year;
    vector<string> articles;

    cout << "Input name";
    cin >> line;
    paper.SetName(line);
    cout << "Input number";
    cin >> num;
    paper.SetNum(num);
    cout << "Input day";
    cin >> num;
    cout << "Input month";
    cin >> month;
    cout << "Input year";
    cin >> year;

    Data dateOf(num, month, year);
    paper.SetData(dateOf);

    cout << "Input number of articles";
    cin >> num;
    paper.SetNumArticles(5);

    for (int i = 0; i < num; i++) {
        cin >> line;
        articles.push_back(line);
    }
    paper.SetArticle(articles);

    cout << "Input edition";
    cin >> num;
    paper.SetEdition(num);
        cout << "Input FIO Editor";
    cin >> line;
    paper.SetFIOEditor(line);

    int currentYear = list.currentNode().getYear();
    if (paper.getYear() == currentYear)
        list.AddHead(paper);
    else {
        line = to_string(currentYear);
        string name = "C:\\Users\\Desktop\\" + line + ".txt";
        list.WriteToFile(name);
        list.Clear();
        list.AddHead(paper);
    }
}
```

```
void loadArchive() {
    if (!list.isEmpty()) list.Clear();
    NewsPaper paper;

    string year;
    cout << "Please input year:" << endl;

    cin >> year;
    ifstream in("C:\\Users\\Desktop\\" + year + ".txt");
    string line;
    int num;
    vector<string> articles;
    Data dateOf;
    while (!in.eof())
    {
        getline(in, line);
        paper.SetName(line);

        in >> num;
        paper.SetNum(num);

        in >> dateOf;
        paper.SetData(dateOf);

        in >> num;
        paper.SetNumArticles(num);

        for (size_t i = 0; i < num + 1; i++) {
            getline(in, line);
            articles.push_back(line);
        }
        paper.SetArticle(articles);

        in >> num;
        paper.SetEdition(num);

        getline(in, line);
        getline(in, line);
        paper.SetFIOEditor(line);
        list.AddHead(paper);
        getline(in, line);
    }
    in.close();
}
```

```
int main()
{
    char key;
    do {
        cout << endl << "Input 1 to create newspaper"
            << endl << "Input 2 to load archive"
            << endl << "Input 3 to save newspapers"
            << endl << "Input 4 to view all newspapers"
            << endl << "Input 5 to view current newspaper"
            << endl << "Input 'n' to go to next newspaper"
            << endl << "Input 'p' to go to previous newspaper" << endl;

        cin.get(key);
        switch (key)
        {
        case('1'):
            createNewsPaper();
            break;
        case('2'):
            loadArchive();
            break;
        case('3'): {
            string line = to_string(list.currentNode().getYear());
            string NameOfFile = "C:\\Users\\Desktop\\" + line + ".txt";
            list.WriteToFile(NameOfFile);
            break;
        }
        case('4'):
            list.ListHead();
            break;
        case('5'):
            if (list.isEmpty())
                cout << "No newspapers" << endl;
            else
                cout << list.currentNode();
            break;
        case('n'):
            list.Next();
            break;

        case('p'):
            list.Pred();
            break;
        default:
            break;
        }
        cin.get();

    } while (key != 'q');
}
```

Внимательно изучите решение и приступайте к самостоятельному решению задачи по варианту.

3. ТРЕБОВАНИЯ К ПРОГРАММЕ

В программе предусмотреть ввод и вывод исходных данных и полученных результатов с необходимыми поясняющими текстами, использовать графический пользовательский интерфейс, иметь необходимые элементы управления, меню, подсказки, быть законченной. Предусмотреть контроль ввода. Подготовить тесты или контрольный пример.

Программа должна быть разработана в системе программирования - Visual Studio C++.

Программа должна выполнять все указанные в задании функции. Набор дополнительных функций для удобства работы с программой необходимо определить самостоятельно.

Тестирование контейнерных классов должно быть выполнено на разных типах данных.

В программе рекомендуется предусмотреть возможность сохранения данных в файл и их чтение из файла. Это может быть как запись в файл по команде пользователя, так и автоматическое сохранение/чтение данных при завершении/запуске программы. Используемые операции для записи данных в файл и их дальнейшего чтения зависят от выбранного языка программирования. Так же для работы с файлом в рамках курсового проекта могут использоваться любые другие операции, способы, изученные самостоятельно.

Во время защиты курсовой работы выполнение программы должно быть продемонстрировано на контрольном примере и/или подготовленных тестах.

4. ТИПОВАЯ СТРУКТУРА ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

Пояснительная записка состоит из глав и разделов:

Оглавление
Введение
1 Постановка задачи
- 1.1 Описание предметной области задачи
- 1.2 Функциональное назначение программы
- 1.3 Описание входных и выходных данных
- 1.4 Характеристики программы

2 Разработка программы
- 2.1 Диаграмма иерархии классов и ее описание
- 2.2 Описание классов
- 2.3 Описание алгоритмов и методов решения
- 2.4 Интерфейс программы

3 Описание программы
- 3.1 Общие сведения
- 3.2 Описание модулей программы
- 3.3 Текст программы

3.4 Спецификация состава программы

3.5 Тестирование программы

Источники, использованные при разработке

Приложение 1 Исходный текст программ

Пояснения к главам и разделам Введение

Цель работы и задачи, которые потребуется решить для достижения цели, указать номер варианта и текст задания.

Постановка задачи

Решение задачи начинается с ее постановки. Дается точное описание исходных данных, условий задачи и результатов ее решения.

Описание предметной области задачи

Текст задание и пояснения. Описываются возможные пути решения задачи с указанием их достоинств и недостатков. Выбирается и обосновывается метод решения задачи.

Функциональное назначение

Должны быть указаны требуемые функции разрабатываемой программы и сведения об ограничениях на применение.

Описание входных и выходных данных

Выходные данные, результаты. Описание выходных данных и их форматы.

Входные данные, исходные данные. Организация и предварительная подготовка входных данных, формат и смысловое описание.

Обязательно описываются ограничения, накладываемые на входные/выходные данные, необходимая разрядность и точность представления исходных данных и результатов решения. Указываются возможные пределы изменения входных/выходных параметров задачи.

В этом разделе могут быть описаны типы данных и основные приемы их программирования. Например, если в задаче используются динамические структуры, то перечисляются виды динамических структур данных и основные функции по работе с динамическими структурами. Если задача заключается в формировании массива и дальнейшей работе с массивом, то приводится описание типовых алгоритмов обработки массива. Если применяются файлы - приемы работы с файлами и т.д.

Характеристики программы

В этом разделе описывают нефункциональные и эксплуатационные требования к операционной системе, среде разработки, технологиям, методам и языкам разработки, требования к интерфейсам и др. Может быть описан концептуальный подход к методам решения.

Описание классов

Пример описания классов см. в Приложении 3.

Описание алгоритмов и методов решения

Представление алгоритмов решения в методах классов. Излагаются основные требования к алгоритму и пути их реализации. Приводится схема или псевдокод алгоритма, дается пояснение.

Интерфейс программы

Должен быть представлен проект интерфейса пользователя (консоль или Windows-окно (или окон)). Элементы интерфейса должны быть пронумерованы. Ниже приводится расшифровка пронумерованных элементов интерфейса: название(тип) и имя; назначение в программе; события, на которые данный компонент откликается. Для каждого компонента графического интерфейса должны быть указаны свойства, изменяемые при проектировании окна (Приложение 2).

Описание программы

Общие сведения. Должны быть указаны наименование программы, идентификатор исполняемого файла, программное обеспечение, необходимое для функционирования программы, язык программирования, методология и технология написания программы.

Описание модулей программы

Пример описания модулей программы см. в Приложении 5.

Текст программы

В разделе представлен исходный текст программы или ссылка на Приложение к Пояснительной записке. Перед заголовками функций, процедур, модулей должны быть комментарии, описывающие назначение подпрограммы/модуля, назначение входных и выходных параметров.

Спецификация состава программы

Пример описания спецификации состава программы см. в Приложении 4.

Тестирование программы

Кратко описывается среда программирования, перечисляются и описываются средства отладки в данной среде. Перечисляются требования, подлежащие проверке при испытании программы. Приводятся тесты или контрольный пример и ожидаемые результаты.

Прилагаются сканы решения контрольного примера и результатов тестирования.

Источники, использованные при разработке

В разделе перечисляется литература и другие информационные ресурсы, использованные при выполнении курсовой работы

5. РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ РАЗДЕЛОВ КУРСОВОЙ РАБОТЫ

Для решения поставленной задачи необходимо предварительно ознакомиться с литературой и другими источниками, посвященных теме

задания. При этом следует обратить внимание на средства, используемые для решения аналогичных задач или для решения каких-либо ключевых моментов задачи. Этап работы с литературой и другими источниками, должен закончиться описанием предметной области задачи, где будут собраны полученные сведения из литературы, дан их анализ с точки зрения приложения к поставленной задаче.

Лаконичная спецификация вариантов задания на курсовую работу предоставляют обучающемуся простор для творчества. В вариантах заданий умышленно опущены некоторые детали и необходимые требования. После ознакомления с литературой обучающейся должен оценить возможности языка программирования и вычислительной техники, на которой предлагается реализовать решение. Результатом этой работы должна быть точная формулировка задачи со всеми ограничениями и требованиями.

При решении задач необходимо придерживаться техники пошаговой детализации, использовать стандартные структуры, не забывая при этом о развитии программного окружения программиста, расширяя возможности языка за счет включения новых процедур и функций.

При разработке алгоритма необходимо предусмотреть средства проверки и тестирования программы, удобство работы пользователя, возможные модификации.

При написании программы не следует забывать о хорошем стиле программирования, о таких понятиях, как читабельность, эффективность, надежность. Необходимо искать наиболее простые и естественные приемы и методы решения.

6. ТРЕБОВАНИЯ К ОФОРМЛЕНИЮ ОТЧЕТА

Текст отчета должен быть напечатан на одной стороне стандартного листа белой бумаги формата 210 x 297 мм (А4) в редакторе MS Word. Шрифт – Times New Roman, размер – 14 pt, междустрочный интервал одинарный, выравнивание по ширине. Размеры полей: слева и справа – 2,5 см, сверху – 2 см, снизу – 3 см. Абзацный отступ – 1,25 см. Нумерация страниц – внизу посередине листа. На титульном листе номер не ставится.

Заголовки оформляются заглавными буквами шрифтом размера 14 pt, начертание – полужирное. Точка в конце заголовка не ставится.

Тексты программ необходимо набирать шрифтом Courier New 14 pt прямого начертания.

Изображения (рисунки) должны сопровождаться названиями. Названия размещаются под рисунком и оформляют обычным шрифтом, размер – 14 pt. Нумерация рисунков – сквозная в пределах всего документа. В конце названия точка не ставится. Например: «Рисунок 1. Главное окно программы». В тексте должны быть ссылки на все рисунки. Например: «Программа имеет графический интерфейс (Рис. 1)».

Названия таблиц размещаются над таблицами, например «Таблица 1 – Название таблицы», выравнивание вправо. Нумерация таблиц – сквозная в пределах всего документа. Шрифт – Times New Roman, размер – от 11 pt до 14 pt. В тексте должны быть ссылки на все таблицы.

Текст, изображения, таблицы не должны выходить за поля документа.

7. ПОРЯДОК ЗАЩИТЫ РАБОТЫ

Для защиты курсовой работы должны быть представлены преподавателю следующие материалы:

1) программа в виде исходного кода и исполняемого файла с набором тестовых случаев для проверки корректной работы;
2) Пояснительная записка в электронном и печатном виде. Электронный вариант должен быть отправлен в систему поддержки учебного процесса предварительно, не менее чем за 3 дня до защиты.

Защита включает в себя:

1) демонстрацию выполнения программы на тестах и/или контрольном примере, подготовленных заранее;
2) демонстрацию исходного кода;
3) ознакомление преподавателя с Пояснительной запиской ;
4) ответы на вопросы преподавателя (например, «почему было реализовано именно таким образом», «имело ли смысл предусмотреть в программе такие-то функции» и т.п.);

При выставлении баллов за курсовую работу оценивается программа (до 50 баллов), Пояснительная записка (до 30 баллов), качество защиты работы и ответы на вопросы (до 20 баллов).

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. C/C++. Программирование на языке высокого уровня [] : учебник для студентов вузов, обучающихся по направлению "Информатика и вычислительная техника" / Т. А. Павловская. - М. [и др.] : Питер, 2008. - 461 с. : ил. - (Учебник для вузов)

2. C/C++. Структурное и объектно-ориентированное программирование [] : практикум / Т. А. Павловская, Ю. А. Щупак. - Москва [и др.] : Питер, 2010. - 348 с. : ил. - (Учебное пособие).

3. Ахо Альфред В., Хопкрофт Джон, Ульман Джеффри Д., Структуры данных и алгоритмы. : Пер. с англ. : Уч. пос. — М. : Издательский дом "Вильямс", 2000. — 384 с. : ил.

4. Введение в теорию автоматов, языков и вычислений, 2-е изд. / Джон Хопкрофт, Раджив Мотвани, Джеффри Ульман. – Пер. с англ. - М. : Издательский дом «Вильямс». 2002 – 528с. : ил.

Приложение 2 Пример описания интерфейса программы

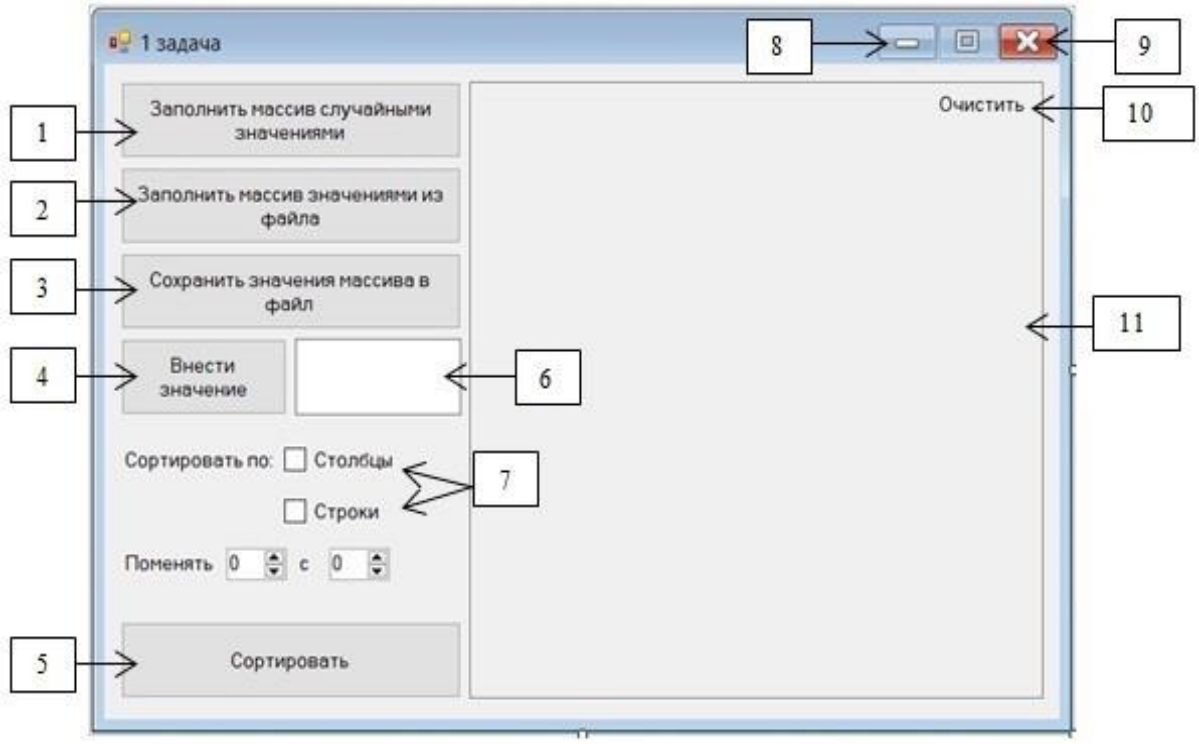

Рисунок 1. Главное окно программы

1. Button1:Button срабатывает при событии Click и заполняет массив случайными значениями.
2. Button2:Button срабатывает при событии Click и вызывает диалоговое окно для выбора файла.
3. Button3:Button срабатывает при событии Click и вызывает диалоговое окно для выбора файла, в который будут сохранены значения массива.
4. Button4:Button срабатывает при событии Click и вносит элемент в массив.
5. Button5:Button срабатывает при событии Click и меняются местами указанные строки/столбцы.
6. Textbox1:TextBox - Поле ввода нового значения массива.
7. CheckBox1,2:CheckBox - Выбираем, что необходимо поменять местами.
8. Свернуть программу.
9. Закрыть программу.
10. Label1:Label - Очистить поле вывода.
11. TextBox2:TextBox - Поле вывода, куда выводятся значения массива.

Приложение 3 Пример описания классов

Таблица 1 – Класс Kurator

Параметр	Значение
Комментарий	Класс, представляющий собой куратора отделения
Поля	IDKurator : integer – номер куратора FamilyK : string – Фамилия куратора NameK : string – Имя куратора PatronK : string – Отчество куратора ShNameK : string – Короткое ФИО куратора loginK : string – логин куратора password : string – пароль куратора Все атрибуты имеют модификатор доступа – private.
Методы	search login() – поиск логина check login() – проверка логина check password() – проверка пароля get curator() – получить список кураторов determine curator() – определить выбор куратора в интерфейсе create a new curator() – создать нового куратора save attribute() – сохранить атрибут delete curator() – удалить запись о кураторе Все операции имеют модификатор доступа – public.

Таблица 2 – Класс Razdel

Параметр	Значение
Комментарий	Класс, представляющий собой раздел тематики цикла
Поля	NameR : string – наименование раздела тематики ShNameR : string – краткое наименование раздела TimeRTeoriya : integer – количество часов, выделенных на лекционные занятия по разделу (суммируется по входящим в раздел темам) TimeRPractic : integer – количество часов, выделенных на практические занятия по разделу (суммируется по входящим в раздел темам) TimeRZaoch : integer – количество часов выделенное на самостоятельное обучение слушателям IDRazdel : integer – номер раздела Temas : Класс Tema – список тем раздела

	Все атрибуты имеют модификатор доступа – private.
Методы	get razdel for tematic() – получить разделы для тематики add razdel() – добавить новый раздел в тематику delete razdel() – удалить раздел в тематике change attribute() – изменить атрибут раздела change time() – пересчитать время выделенное на раздел Все операции имеют модификатор доступа – public.

Приложение 4 Пример спецификации состава программы

Таблица 1 – Спецификация состава программы

Обозначение	Наименование	Примечание
Курсовая.pas	Файл проекта	Содержит в себе исходный код программного обеспечения
Курсовая.pabcproj	Файл проекта	Исполнительный файл. Pabcproj — PascalABC.NET Project.
Курсовая.opt	Файл модуля всего проекта	Хранит в себе различные параметры, применяемые для обработки выходных значения.
Unit4.pas	Файл 4 модуля	Содержит в себе исходный код программного обеспечения
Unit4.Form.inc	Файл с необходимыми библиотеками	Содержит все необходимые библиотеки
Unit3.Form3.resources	Данные ресурсов	Двоичный файл, содержащий данные ресурсов .NET.
Unit3.pas	Файл 3 модуля	Содержит в себе исходный код программного обеспечения
Unit3.Form.inc	Файл с необходимыми библиотеками	Содержит все необходимые библиотеки
Unit1.pas	Файл 1 модуля	Содержит в себе исходный код программного обеспечения
Unit1.Form.inc	Файл с необходимыми библиотеками	Содержит все необходимые библиотеки
Unit2.pas	Файл 2 модуля	Содержит в себе исходный код программного обеспечения
Unit2.Form.inc	Файл с необходимыми библиотеками	Содержит все необходимые библиотеки
Massiv.txt	Текстовый файл	Содержит значения массива

Приложение 5 Пример описания модулей программы

Таблица 1 – Модули программы

Имя файла	Выполняемые функции
Unit_Login.pas	Экранная форма «Вход в систему». Реализует интерфейс, предоставляющий пользователю возможность ввода логина и пароля. Основная функция является авторизация пользователя и открытие главной формы программы.
Unit_main.pas	Главная форма программы. Реализует интерфейс. Основными функциями являются переход к формам программы: формы для ввода данных для оказания услуги клиенту, формы для расчета с клиентом, просмотр отчетов. Также реализует интерфейс предоставляющий возможность выбора типа клиента и ввода Ф.И.О клиента задач.
Unit_Input1.pas	Экранная форма «Прием почтовых отправлений». Реализует интерфейс, представляемый пользователю ввод данных для оказания услуги клиенту.
Unit_Input2.pas	Экранная форма «Вручение почтовых отправлений». Реализует интерфейс, представляемый пользователю ввод данных для оказания услуги клиенту.
Unit_Input3.pas	Экранная форма «Прием коммунальных платежей». Реализует интерфейс, представляемый пользователю ввод данных для оказания услуги клиенту.
Unit_Input4.pas	Экранная форма «Продать ТМЦ». Реализует интерфейс, представляемый пользователю ввод данных для оказания услуги клиенту.
Unit_Input5.pas	Экранная форма «Прием денежных переводов». Реализует интерфейс, представляемый пользователю ввод данных для оказания услуги клиенту.
Unit_Input6.pas	Экранная форма «Выбор объекта почтовой связи». Реализует интерфейс, представляемый пользователю выбор объекта почтовой связи.
Unit_Calc.pas	Экранная форма «Расчет с клиентом». Реализует интерфейс для осуществления расчета с клиентом.

Unit_QuickReport.pas	Экранная форма «Просмотр отчета». Реализует интерфейс, предоставляемый пользователю для просмотра отчета

Приложение 6 Варианты заданий 1-20

Вариант 1. Составить программу, которая содержит информацию о наличии троллейбусов в парке.

Сведения о каждом троллейбусов содержат:

- номер;
- фамилию водителя; номер маршрута.

Программа должна обеспечивать:

- начальное формирование данных о троллейбусах в парке в виде динамического списка;
- формирование списка данных о троллейбусах на маршруте в виде динамического списка;
- при выезде троллейбуса из парка программа удаляет данные об этом троллейбусе из списка «в парке», и записывает эти данные в список «на маршруте»;
- при въезде каждого троллейбуса в парк программа удаляет данные из списка троллейбусов на маршруте, и записывает эти данные в список, находящихся в парке;
- по запросу выдаются сведения о троллейбусах, находящихся в парке, или о троллейбусах, находящихся на маршруте.

Программа должна обеспечивать диалог и контроль ошибок при вводе.

Вариант 2. Составить программу, которая содержит текущую информацию о заявках на туристические маршруты.

Каждая заявка содержит:

- номер туристического маршрута;
- географические пункты маршрута;
- фамилию и инициалы туриста;
- желаемую дату начала маршрута.

Программа должна обеспечивать:

- хранение всех заявок в виде списка;
- добавление заявок в список;
- удаление заявок;
- вывод всех заявок по заданному номеру туристического маршрута и его дате;
- вывод всех заявок.

Программа должна обеспечивать диалог и контроль ошибок при вводе.

Вариант 3. Составить программу, которая содержит данные о наличии семян в магазине для садоводов.

Сведения о каждом пакете семян содержат:

- название растения;
- количество пакетов семян;
- цена пакета;
- дата выпуска.

Программа должна обеспечивать:

- начальное формирование данных о всех семенах в виде алфавитных списков по категориям (овощи, декоративные растения, деревья и кустарники);
- при продаже семян изменять значение количества пакетов;
- вывод всего списка данной категории семян и другой информации о них;
- поиск по всем спискам указанного растения и переход к предыдущему и последующему.

Программа должна обеспечивать диалог и контроль ошибок при вводе.

Вариант 4.Составить программу, которая содержит информацию об изданиях журнала в редакции.

О номере журнала содержат следующие сведения:

- номер;
- список статей;
- список авторов статей;
- количество проданных экземпляров;
- год издания;
- общее количество изданных экземпляров номера. Программа должна обеспечивать:
- начальное формирование данных о всех номерах журнала в виде списка;
- при выводе, указывается номер журнала и по номеру выводятся все сведения о журнале;
- при вводе номера журнала можно изменить количество проданных экземпляров;
- удаление данных о всех журналах заданного года, перемещаемых в архив;
- по запросу выдается процент проданных журналов заданного номера и общий процент проданных журналов этого года.

Программа должна обеспечивать диалог и контроль ошибок при вводе.

Вариант 5.Составить программу, которая содержит информацию о заказах такси.

Сведения о каждом такси содержат:

номер такси;

- фамилию и инициалы водителя;
- номер заказа;
- признак того, где находится такси — на заказе или свободно. Программа должна обеспечивать:
- начальное формирование данных обо всех такси в виде списка;
- при выезде каждого такси по заказу, программа устанавливает значение признака «такси на заказе»;
- при выполнении заказа, и программа устанавливает значение признака «такси свободно»;
- по запросу выдаются сведения о свободных такси, или о такси, находящихся на заказе;
- список всех заказов и соответствующих им такси и водителях.

Программа должна обеспечивать диалог и контроль ошибок при вводе.

Вариант 6. Составить программу, которая содержит текущую информацию о бронирование мест в гостинице.

Каждая заявка на бронь содержит:

- номер брони;
- вид забронированного номера гостиницы (одноместный, двухместный);
- фамилию и инициалы гостя;
- дата въезда;
- признак отказа от брони.

Программа должна обеспечивать:

- хранение заявок в виде списка;
- добавление заявки на бронирование;
- по признаку отказа, вывод всех заявок с их последующим удалением; вывод забронированных номеров по виду и дате въезда; вывод всех заявок на дату въезда.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 7. Система заявок на услуги сервисной компании организована как линейный список. Для каждой заявки содержатся следующие сведения:

- номер заявки;
- дата создания;
- вид услуги (1, 2, 3, …)
- отметка о выполнении услуги.

Составить программу, которая обеспечивает:

- начальное формирование списка заявок;

- вывод невыполненных заявок на указанную дату;
- удаление заявок с отметкой о выполнении, дата создания которых меньше заданной;
- выборку услуги с наибольшим количеством обращений.

Программа должна обеспечивать диалог и контроль ошибок при вводе **Вариант 8.** Картотека бюро проката бытовых приборов. Карточка каждого прибора содержит:

- номер;
- название;
- стоимость проката в сутки;
- статус (в прокате, доступен).

Составить программу, которая обеспечивает:

- начальное формирование картотеки;
- установку поля статуса о прокате прибора;
- поиск в картотеке подходящего варианта по стоимости и названию: при равенстве названий, при различии в стоимости в пределах 10% и доступном статусе выводится соответствующая карточка прибора для выдачи в прокат;
- вывод всех доступных приборов;
- вывод всего списка приборов.

Программа должна обеспечивать диалог и контроль ошибок при вводе.

Вариант 9. Регистратура поликлиники имеет картотеку пациентов для извещения о диспансеризации, организованную как линейный список. Карточка каждого пациента содержит:

- номер медицинского полиса пациента;
- номер телефона пациента;
- возраст пациента;
- статус извещения (извещен, не извещен).

Составить программу, которая:

- обеспечивает начальное формирование картотеки;
- выводит номера телефонов, не извещенных пациентов заданного возраста;
- вводит и выводит статус извещения по номеру телефона или номеру полиса;
- выводит всю картотеку.

Программа должна обеспечивать диалог и контроль ошибок при вводе.

Вариант 10. Система автовокзала содержит сведения об отправлении междугородних автобусов в виде линейного списка. Для каждого автобуса указывается:

- номер маршрута;
- город назначения;
- время отправления.

Составить программу, которая:

- обеспечивает первоначальный ввод данных и формирование линейного списка;
- производит вывод всего списка;
- вводит номер маршрута и выводит все данные о маршруте;
- вводит название города назначения и выводит данные обо всех автобусах, отправляющихся в этот город.

Программа должна обеспечивать диалог и контроль ошибок при вводе.

Вариант 11. Составить программу, которая содержит текущую информацию о книгах в библиотеке.

Сведения о книгах содержат:

- номер УДК;
- фамилию и инициалы автора;
- название;
- год издания;
- количество экземпляров данной книги в библиотеке.

Программа должна обеспечивать:

- начальное формирование данных о всех книгах в библиотеке в виде списка;
- добавление данных о книгах, вновь поступающих в библиотеку;
- удаление данных о списываемых книгах;
- по запросу выдаются сведения о наличии книг в библиотеке, упорядоченные по годам издания.

Вариант 12. Составить программу, которая содержит текущую информацию о заявках на авиабилеты. Каждая заявка содержит:

- пункт назначения;
- номер рейса;
- фамилию и инициалы пассажира;
- желаемую дату вылета.

Программа должна обеспечивать:

- хранение всех заявок в виде списка;
- добавление заявок в список;
- удаление заявок;
- вывод заявок по заданному номеру рейса и дате вылета; вывод всех заявок.

Вариант 13. Картотека в бюро обмена квартир организована как линейный список. Сведения о каждой квартире содержат:

- количество комнат;
- этаж;
- площадь; адрес.

Составить программу, которая обеспечивает:

- начальное формирование картотеки;
- ввод заявки на обмен;
- поиск в картотеке подходящего варианта: при равенстве количества комнат и этажа и различии площадей в пределах 10% выводится соответствующая карточка и удаляется из списка, в противном случае поступившая заявка включается в список;
- вывод всего списка.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 14. Составить программу, моделирующую заполнение магнитного диска.

- Сведения о диске:
- Общий объем памяти на диске *Size* Кбайт.
- Файлы имеют имя и произвольную длину в Кбайтах.

В процессе работы файлы либо записываются на диск, либо удаляются с него.
Программа должна обеспечить:

- В начале работы файлы записываются подряд друг за другом.
- После удаления файла на диске образуется свободный участок памяти, и вновь записываемый файл либо размещается на свободном участке, либо, если файл не вмещается в свободный участок, размещается после последнего записанного файла. В случае, когда файл превосходит длину самого большого свободного участка, выдается аварийное сообщение.
- Требование на запись или удаление файла задается в строке, которая содержит имя файла, его длину в байтах, признак записи или удаления. Программа должна выдавать по запросу сведения о занятых и свободных участках памяти на диске.
- Создать список занятых участков и список свободных участков памяти на диске.

Вариант 15. Предметный указатель организован как линейный список.

Каждая компонента указателя содержит слово и номера страниц, на которых это слово встречается. Количество номеров страниц, относящихся к одному слову, от одного до десяти.
Составить программу, которая обеспечивает:

- начальное формирование предметного указателя;
- вывод предметного указателя;
- вывод номеров страниц для заданного слова.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 16. В файловой системе каталог файлов организован как линейный список. Для каждого файла в каталоге содержатся следующие сведения:

- имя файла;
- дата создания;
- количество обращений к файлу.

Составить программу, которая обеспечивает:

- начальное формирование каталога файлов;
- вывод каталога файлов;
- удаление файлов, дата создания которых меньше заданной;
- выборку файла с наибольшим количеством обращений.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 17. Текст помощи для некоторой программы организован как линейный список.
Каждая компонента текста помощи содержит термин (слово) и текст, содержащий пояснения к этому термину. Количество строк текста, относящихся к одному термину, от одной до пяти.
Составить программу, которая обеспечивает:

- начальное формирование текста помощи в виде списка;
- вывод текста помощи;
- вывод поясняющего текста для заданного термина.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 18

Англо-русский словарь построен как список.
Каждая компонента содержит английское слово, соответствующее ему русское слово и счетчик количества обращений к данной компоненте.

Первоначально список формируется согласно английскому алфавиту. В процессе эксплуатации словаря при каждом обращении к компоненте в счетчик обращений добавляется единица.

Составить программу, которая:

- обеспечивает начальный ввод словаря с конкретными значениями счетчиков обращений;
- формирует новое представление словаря в виде двоичного дерева по следующему алгоритму: а) в старом словаре ищется компонента с наибольшим значением счетчика обращений; б) найденная компонента заносится в новый словарь и удаляется из старого; в) переход к п. а) до исчерпания исходного словаря;
- производит вывод исходного и нового словарей.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 19

Информационная система на железнодорожном вокзале содержит сведения об отправлении поездов дальнего следования. Для каждого поезда указывается:

- номер поезда;
- станция назначения;
- время отправления.

Данные в информационной системе организованы в виде линейного списка. Составить программу, которая:

- обеспечивает первоначальный ввод данных в информационную систему и формирование линейного списка;
- производит вывод всего списка;
- вводит номер поезда и выводит все данные об этом поезде;
- вводит название станции назначения и выводит данные обо всех поездах, следующих до этой станции.

Программа должна обеспечивать диалог с помощью меню и контроль ошибок при вводе.

Вариант 20

Составить программу, которая содержит текущую информацию о книгах в библиотеке.

Сведения о книгах содержат:

- номер УДК;
- фамилию и инициалы автора;
- название;
- год издания;

- количество экземпляров данной книги в библиотеке.

Программа должна обеспечивать:

- начальное формирование данных о всех книгах в библиотеке в виде дерева;
- добавление данных о книгах, вновь поступающих в библиотеку;
- удаление данных о списываемых книгах и помещение этих данных в вектор;
- по запросу выдаются сведения о наличии книг в библиотеке, упорядоченные по годам издания.

Исходные данные могут вводиться как из файла, так и из диалога.