ЛАБОРАТОРНАЯ РАБОТА №3.

Классы и объекты.

Цель работы: Изучение языковых конструкций для работы с классами и объектами, приобретение навыков разработки и использования классов.

Теоретические сведения.

3.1) Объектно-ориентированное программирование (ООП) и классы.

Понятие класса является одним из ключевых для объектно-ориентированной разработки программ. ООП дает программисту ряд возможностей, которые невозможно реализовать в рамках процедурной парадигмы, то есть с помощью функций C++. Одной из таких возможностей является защита обрабатываемых программой данных от некорректных действий пользователя. Кроме того, использование классов часто делает структуру программы более прозрачной и легко модифицируемой. По этой причине в настоящее время ООП – это основной метод создания «больших» программных пакетов, содержащих десятки тысяч (и более) строк исходного кода и требующих усилий нескольких групп разработчиков.

С точки зрения внутренней организации, класс представляет собой объединение данных (вообще говоря, разнотипных) с функциями, которые эти данные обрабатывают. Можно говорить о том, что класс C++ является расширением понятия структуры C++. Вместе с тем, объединение данных и функций не является механическим, так как при таком объединении реализуется разграничение доступа к данным.

В общем виде объявление класса выглядит следующим образом

```
class MyClass    // объявление класса MyClass
{
  private:       // закрытая часть класса (по умолчанию)
    // элементы в этой части доступны только из класса
  protected:     // защищенная часть класса
    // элементы в этой части доступны из класса и его потомков
  public:        // открытая часть класса
    // элементы в этой части доступны из любой части программы
};
```

Объявление начинается с ключевого слова class, вслед за которым указывается имя разрабатываемого класса (в приведенном примере – MyClass). Далее в фигурных скобках идет список *полей* (элементов данных) и *методов* (функций обработки). При этом поля и методы класса разделены на три группы – закрытую (записывается после спецификатора доступа private), защищенную (после спецификатора protected) и открытую (после спецификатора public). Если спецификатор доступа не указан, то по умолчанию элемент(ы) класса считаются закрытыми.

Расположение элемента в той или иной группе (private, protected, public) определяет режим доступности этого элемента. К примеру, поля и методы, объявленные в public-области класса, являются видимыми из любой части программы. С другой стороны, поля и метод из private-области видимы только «внутри» класса, но невидимы «извне». Этот механизм позволяет защищать данные, скрывая их от внешнего пользователя. Механизм

часто называют *инкапсуляцией данных*. Подробнее о разграничении доступа к полям и методам класса – см. лекции и справочные материалы по языку C++.

Рассмотрим далее следующий пример

```
class book              // объявление класса book
{
  private:              // закрытая часть класса
    string title;
    string authors;
    string publisher;
    int year;
    unsigned int pages;

  public:               // открытая часть
    void read_from(ifstream &file);    // чтение из файла
    void display();                    // вывод на экран
};
```

Здесь объявляется класс book, который содержит несколько полей данных (они повторяют поля структуры из предыдущей лабораторной работы), а также прототипы компонентных функций read_from и display. Обратим внимание на то, что все поля данных расположены в закрытой части класса, а прототипы функций – в открытой.

Метод read_from будем использовать для считывания информации о книге из текстового файла. Структура файла my_books.txt описана в предыдущей работе. Файловый поток, из которого будут считываться данные, передается функции в качестве параметра. Метод display будем использовать для вывода полей book на экран.

Определим далее компонентные функции read_from и display, то есть запишем их программную реализацию. Так как они являются методами класса book, определения имеют следующий вид

```
void book::read_from(ifstream &file)
{
    getline(file, title);
    getline(file, authors);
    getline(file, publisher);
    file >> year;
    file >> pages;
    file.get();
}

void book::display()
{
    cout << "======================================\n";
    cout << "  Название:     " << title << endl;
    cout << "  Автор(ы):     " << authors << endl;
    cout << "  Издательство: " << publisher << endl;
    cout << "  Год выпуска:  " << year << endl;
    cout << "  Страниц:      " << pages << endl;
}
```

Запись вида void book::display() сообщает компилятору о том, что далее определяется метод display класса book. Согласно указанной сигнатуре, эта компонентная функция не принимает параметров и не возвращает результата.

3.2) Использование класса. Создание объектов.

В рамках ООП классы часто называют *абстрактными типами данных* (АТД). В рассмотренном примере класс book может считаться описанием некоторой абстрактной книги, которая обладает своими свойствами и поведением. Свойства книги здесь определяются ее полями (название, авторы, издательство и т.д.), а поведение – методами класса (считать данные из файла, вывести на экран и др.).

Реальное использование разработанного класса C++ возможно только после того, как в программе будет создан один или несколько экземпляров класса. Эти экземпляры называют *объектами*. Объект выступает воплощением абстрактного класса и имеет конкретное содержание (конкретное название, конкретных авторов и т.д.). Кроме того, объект всегда имеет конкретный адрес в оперативной памяти компьютера. Отношения между классами и объектами в ООП во многом аналогичны отношениям между типами данных (int, float, char, …) и переменными (x, y, z, index, …) в стандартной не-ООП программе.

Создание объекта некоторого класса похоже на объявление переменной. В случае, если объект создается статически, синтаксис имеет вид

```
имя_класса имя_объекта;
```

Статический массив объектов объявляется в программе следующим образом

```
имя_класса имя_массива[число_элементов];
```

И наконец, для динамического расположения массива объектов в памяти используется оператор new

```
имя_класса *имя_массива = new имя_класса[число_элементов];
```

Освободить память, занятую массивом объектов, можно с помощью оператора delete

```
delete[] имя_массива;
```

Рассмотрим несколько примеров

```
book book1, book2, book3;       // создаем 3 объекта класса book
book library[10000];            // создаем массив из 10000 объектов

book *mylib = new book[n];      // выделяем память под массив из n объектов
delete[] mylib;                 // освобождаем выделенную память
```

3.3) Доступ к полям объектов и вызов методов.

Изменение данных в полях объекта и вызов его методов производится с помощью оператора «точка»

```
имя_объекта.поле = ...;
имя_объекта.метод(параметры);
```

Если в программе задано не имя объекта, а указатель на него, то для работы с отдельными компонентами используется оператор «->»

```
указатель_на_объект -> поле = ...;
указатель_на_объект -> метод(параметры);
```

Рассмотрим следующие операторы:

```
book1.title = "Приключения Робинзона Крузо";   // ошибка - доступ закрыт!
book1.year = 2001;                             // ошибка - доступ закрыт!
book1.pages = book2.pages + 1;                 // ошибка - доступ закрыт!
book1.display();
library[100].read_from(infile);
mylib[5].display();
```

В последнем примере использованы объявленные ранее объекты book1 и book2, а также массивы library и mylib. Все приведенные операторы соответствуют синтаксису языка C++, однако часть из этих операторов некорректна с точки зрения доступности используемых данных. Вспомним, что поля title, authors, publisher, year и pages объявлены нами в закрытой (private) части класса book. Это означает, что прямой доступ «извне» к этим полям для любого объекта невозможен. В результате попытка изменить, например, значение поля title объекта book1 (см. 1-ю строку листинга) приведет к ошибке времени компиляции.

Таким образом, в нашем случае работа с объектами класса book ограничена вызовом методов read_from и display. Только эти функции могут изменять значения полей объекта (путем ввода из файла) или просто считывать данные (для вывода на экран). Эта технология обеспечивает сохранность и целостность данных внутри объекта.

3.4) Методы геттеры и сеттеры.

В некоторых случаях полный запрет на доступ к полю класса оказывается неудобным решением. Предположим, что нам необходимо найти книгу с названием «Гиперболоид инженера Гарина» в массиве объектов book. В этом случае выполнить поиск с помощью цикла

```
for (int i = 0; i < N; i++)
{
   if (collection[i].title == "Гиперболоид инженера Гарина")  // ошибка!
          cout << "КНИГА НАЙДЕНА!";
}
```

не возможно, так как поле title закрыто и попытка доступа к нему через конструкцию collection[i].title приведет к сообщению об ошибке. Решением в данном случае может стать использование специального метода класса book, который будет возвращать значение поля title в качестве результата. Ниже приводится новое объявление класса book, в интерфейсную (открытую) часть которого добавлена функция get_title

```
class book
{
  private:
      string title;
      string authors;
      string publisher;
      int year;
```

```
    unsigned int pages;

  public:
    void read_from(ifstream &file);
    void display();
    string get_title();      // метод-геттер
};

// реализация метод get_title
string book::get_title()
{
     return title;
}
```

Единственным назначением метода get_title, как показывает этот пример, является передача названия книги. Вызывая эту функцию, мы получаем содержимое поля title из закрытой части объекта, не нарушая при этом его целостность. Методы такого вида часто называют *геттерами*, а в их названии используется слово get (*англ.* получать). Поиск нужной книги в массиве с помощью метода get_title иллюстрирует следующий фрагмент

```
for (int i = 0; i < N; i++)
{
   if (collection[i].get_title() == "Гиперболоид инженера Гарина")
          cout << "КНИГА НАЙДЕНА!";
}
```

В данном случае ошибка доступа не возникает, так как вызываемая функция находится в открытой части класса.

Из приведенного примера понятно, что геттеры могут использоваться только для считывания значения поля в закрытой части объекта. В ситуациях, когда программе время от времени необходимо изменять это значение, используют другой специальный метод – сеттер. Сеттер устанавливает значение поля объекта, ничего не возвращая. В названии этого метода часто используют слово set (*англ.* устанавливать). Для поля title класса book, к примеру, можно определить следующую функцию-сеттер

```
void book::set_title(string atitle)
{
      title = atitle;
}
```

Прототип этой функции, так же, как и функции get_title, должен быть включен в объявление класса book.

3.5) Указатель this.

Обратим теперь внимание на то, что все рассмотренные нами методы класса book (read_from, display, get_title, set_title) работают с полями объекта, но при этом не принимают объект в качестве аргумента. Вопрос можно поставить следующим образом. Если мы вызываем метод display() для конкретного объекта (скажем, book1), то каким образом функция display «узнает» о том, какие именно данные хранятся в полях объекта book1? Ответ прост: эти данные передаются в функцию *неявно* с помощью специального указателя с фиксированным именем this. Этот указатель делает объект доступным внутри принадлежащей классу функции.

Изменить значение this нельзя, так как это константный указатель. Он является дополнительным скрытым параметром любой нестатической компонентной функции. Вместе с тем, в большинстве случаев у программиста не возникает необходимость обращаться к указателю this напрямую, так как он задействуется даже в том случае, если не называется явным образом. В качестве иллюстрации к сказанному далее приведены два варианта реализации метода set_title. С точки зрения компилятора, оба эти варианта эквивалентны, хотя в одном из них указатель this задействован явно, а в другом – нет.

```
void book::set_title(string atitle)
{
    this -> title = atitle;
}
```

```
void book::set_title(string atitle)
{
    title = atitle;
}
```

В программах C++ чаще можно встретить второй вариант (без явного обращения через указатель this).

3.6) Использование программных модулей. Многофайловые проекты.

Одним из способов структурирования кода программы является его разделение на программные модули, которые хранятся в отдельных дисковых файлах. Модули позволяют разбивать сложные задачи на более мелкие и решать их последовательно. Обычно модули проектируются так, чтобы дать возможность их многократного использования.

Модули часто объединяются в пакеты и библиотеки. Удобство использования модульной архитектуры заключается в возможности обновления или замены модуля, без необходимости изменения остальной программы. Примером отдельного программного модуля может служить любой компонент стандартной библиотеки языка C/C++.

Программный модуль C++ как правило представляет собой пару файлов в одним именем, но разными расширениями – «*.h» и «*.cpp». Файл с расширением «*.h» называется заголовочным (header) и играет роль интерфейса для пользователя. В нем содержатся объявления структур, перечислений, классов и т.д., а также прототипы всех функций в библиотеке. Файл с расширением «*.cpp» называют файлом реализации, он содержит определения всех объявленных в заголовочном файле функций, включая методы классов.

Программные проекты, включающие два или более модулей, называются многофайловыми. Хорошей практикой программирования является выделение в отдельные модули логически связанных функций (например, функций для обработки текстовых строк), а также классов. Для удобства использования, имя создаваемого программного модуля должно соответствовать его фактическому содержанию.

В следующем пункте рассматривается пример проекта, реализованного в виде двух программных модулей. Модуль books является библиотечным и содержит объявление и реализацию класса book. Модуль lab3 используется в качестве основного, он содержит функцию main и использует модуль books.

3.7) Пример приложения Visual C++.

Рассмотрим пример проекта, включающего библиотечный модуль books и основной модуль lab3. Заголовочный файл модуля books содержит объявление класса book и функции find_book.

```
======================== файл books.h ========================
```

```
#ifndef BOOKS_H
#define BOOKS_H

#include <string>
#include <fstream>
using std::string;
using std::ifstream;

class book
{
  private:
       string title;
       string authors;
       string publisher;
       int year;
       unsigned int pages;

  public:
       void read_from(ifstream &file);
       void display();
       string get_title();
};

void find_book(book[], int, string);

#endif
```

```
================================================================
```

Файл books.cpp включает в себя заголовочный файл books.h с помощью директивы #include и, далее, содержит реализацию функций.

```
======================== файл books.cpp ======================
```

```
#include "stdafx.h"
#include "books.h"
#include <iostream>

using namespace std;

// Геттер, передающий значение поля title в вызывающую программу.
string book::get_title()
{
       return title;
}

// Метод read_from класса book считывает информацию о книге
// из файлового потока. Ссылка на файловый поток передается
// как параметр. Метод не возвращает каких-либо значений.
void book::read_from(ifstream &file)
{
       getline(file, title);
       getline(file, authors);
       getline(file, publisher);
```

```
	file >> year;
	file >> pages;
	file.get();
}

// Метод display класса book выводит на экран информацию
// о книге. Не возвращает результата.
void book::display()
{
	cout << "=====================================\n";
	cout << "  Название:     " << title << endl;
	cout << "  Автор(ы):     " << authors << endl;
	cout << "  Издательство: " << publisher << endl;
	cout << "  Год выпуска:  " << year << endl;
	cout << "  Страниц:      " << pages << endl;
}

// Функция поиска книги в массиве (см. описание в ЛР №2).
void find_book(book acollection[], int num, string atitle)
{
	bool found = false;
	for (int i = 0; i < num; i++)
	{
		if (acollection[i].get_title() == atitle)
		{
			found = true;
			cout << "\nНАЙДЕНА КНИГА:\n";
			acollection[i].display();
		}
	}
	if (!found)
		cout << "\nКНИГА С ТАКИМ НАЗВАНИЕМ НЕ НАЙДЕНА!\n";
}
```

==========

Основной модуль программы. Использует класс book и функцию find_book модуля books.

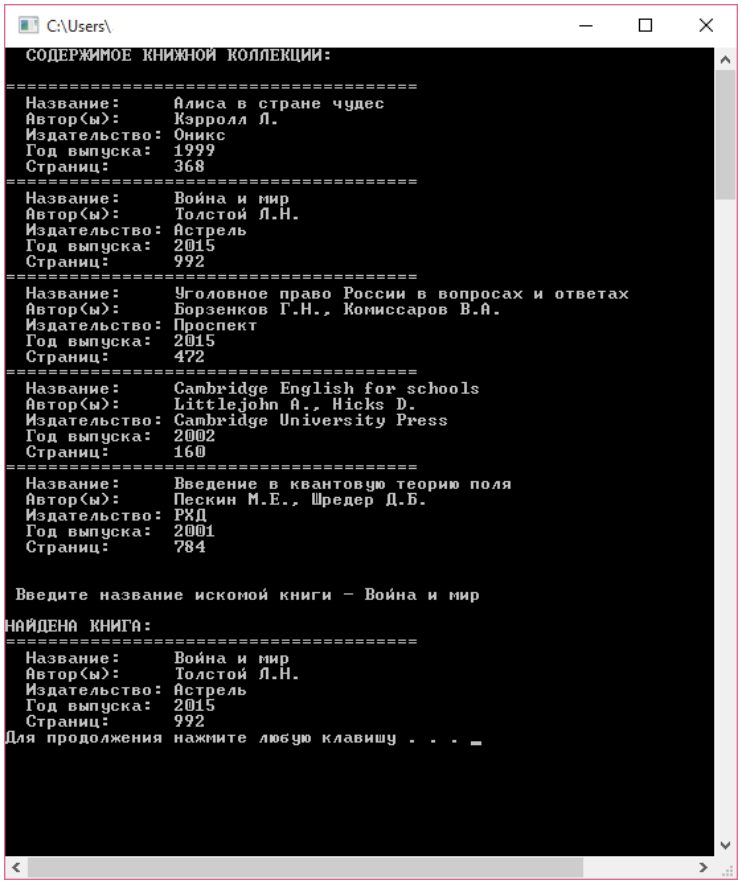
========== файл lab3.cpp ==========

```
#include "stdafx.h"
#include <locale>
#include <iostream>
#include <Windows.h>
#include "books.h"

using namespace std;

int N;                  // число книг в коллекции
book *collection;       // массив объектов класса book

void main(void)
{
	setlocale(LC_ALL, "rus");

	ifstream infile;
	infile.open("my_books.txt");    // открываем файл данных
	if (!infile.is_open())          // файл не найден?
	{
		cout << "Файл данных не найден!" << endl;
		system("pause");
		return;
	}
```

```
        infile >> N;
        infile.get();

        collection = new book[N];          // массив из N объектов класса book

        for (int i = 0; i < N; i++)             // в цикле вызываем метод read_from
               collection[i].read_from(infile);    // для каждого объекта коллекции

        infile.close();

        cout << "  СОДЕРЖИМОЕ КНИЖНОЙ КОЛЛЕКЦИИ:\n\n";
        for (int i = 0; i < N; i++)             // в цикле вызываем метод display
               collection[i].display();            // для всех объектов коллекции

        string find_title;
        cout << "\n\n Введите название искомой книги - ";
        SetConsoleCP(1251);
        getline(cin, find_title);
        SetConsoleCP(866);

        find_book(collection, N, find_title);      // поиск книги

        delete[] collection;

        system("pause");
}
```

==========

Ниже на скриншоте приведены результаты работы приложения.



ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ №3.

1) Разработайте программный класс, объединяющий поля и методы в соответствии со своим вариантом задания. Используйте возможности языка C++ для защиты данных. При необходимости используйте геттеры и сеттеры для доступа к закрытым данным. В качестве методов класса реализуйте:
   - функцию ввода данных с клавиатуры,
   - функцию чтения данных из текстового файла,
   - функцию вывода содержимого на экран,
   - функцию записи содержимого в текстовый или бинарный файл.
2) Разместите разработанный класс в отдельном программном модуле. В заголовочный файл модуля поместите объявление класса и прототипы всех используемых функций, в файл реализации – определения всех методов класса и отдельных функций.
3) В основном модуле программы:
   - создайте массив из нескольких объектов (динамически),
   - заполните массив данными, используя соответствующие методы,
   - выведите массив на экран, используя соответствующие методы,
   - продемонстрируйте работу дополнительных функций из своего задания.

Содержание отчета по лабораторной работе №3.

1) Стандартная «шапка» отчета
2) Цель: формулировка цели работы
3) Теория: краткие сведения о конструкциях C++, изученных в работе (объем 1-2 стр.)
4) Программа: код некоторых компонентов разработанного приложения
   - содержимое заголовочного файла (целиком), включая объявление класса,
   - код одной из компонентных функций класса,
   - фрагменты программы, демонстрирующие создание объектов и вызов методов.