ЛАБОРАТОРНАЯ РАБОТА №2.

Строки, потоки, функции.

Цель работы: Изучение новых возможностей C++ (текстовые строки, потоковый ввод/вывод, динамическое выделение памяти) и языковых конструкций для обработки структурных данных с помощью пользовательских функций.

Теоретические сведения.

2.1) Текстовые строки в C++. Класс string.

В языке C для хранения текстовой информации традиционно используются символьные массивы (char []). Признаком окончания текстовой строки в таком массиве является нулевой символ '\0' (нуль-терминированные строки). При этом часто при написании программ размер символьного массива задается на этапе компиляции, тогда как содержание текстовой строки определяется на этапе выполнения программы (например, вводится с клавиатуры). В результате одной из частотных ошибок при работе со строками C является выход за пределы массива. Кроме того, нуль-терминированные строки оказываются неудобными для копирования текста из одной переменной в другую, при конкатенации (слиянии) строк, сравнении двух строковых переменных.

Одним из новых компонентов языка C++ является стандартный класс string для работы с текстовыми строками. Класс содержит большой набор функций, облегчающий работу с текстом и исключающий большую часть ошибок.

Для использования класса string в программе C++ необходимо подключить библиотеку <string> и сделать класс видимым в пространстве имен std. Следующий фрагмент кода демонстрирует основные приемы работы с текстовыми строками.

```
string s1;                          // создание пустой строки
string s2 = "Иван";                 // создание и инициализация
string s3("Петрович");              // другой вид инициализации

s1 = "Сидоров";                     // операция присваивания
string s4 = s1 + s2 + s3;           // слияние строк

s1[2] = 'Д';                        // работа с отдельными символами строки
if (s1 == s2)                       // сравнение строк между собой
     printf("Строки равны!");
```

Переопределим далее структуру book, используя новый тип данных

```
struct book
{
    string title;
    string authors;
    string publisher;
    int year;
    unsigned int pages;
};
```

При таком определении пропадает необходимость явно указывать максимальное количество символов в строке, так как размер строки всегда соответствует содержимому.

2.2) Потоковый ввод/вывод с консоли. Объекты cin и cout.

Ввод/вывод в C++ основан на концепции потоков. Под потоком понимается последовательность символов, для которой определены операции вставки элемента (символа, байта) и извлечения элемента.

Для использования стандартных потоков, связанных с консолью (клавиатура и текстовый экран), в программе C++ требуется подключение заголовочного файла библиотеки <iostream> и использования пространства имен std. В библиотеке <iostream>, в частности, определены потоковые объекты cin и cout, связанные с клавиатурой и дисплеем, соответственно. Объект cin использует операцию "<<" для занесения (записи) значения в поток, объект cout – операцию ">>" для извлечения (чтения) из потока. Рассмотрим в качестве примера следующий фрагмент.

```
cout << "\nВВЕДИТЕ ДАННЫЕ О КНИГЕ:\n";
cout << "   Название    : ";
getline(cin, collection[i].title);
cout << "   Автор(ы)    : ";
getline(cin, collection[i].authors);
cout << "   Издатель    : ";
getline(cin, collection[i].publisher);
cout << "   Год выпуска : ";
cin  >> collection[i].year;
cout << "   Страниц     : ";
cin  >> collection[i].pages;
```

Обратите внимание на то, что поток cout допускает использование ESC-последовательностей для управления выводом (символы перевода строки ‘\n’ и др.). Для ввода текстовых строк, содержащих пробелы, необходимо использовать функцию getline вместо оператора >> (см. пример).

2.3) Ввод/вывод данных из текстовых файлов. Файловые потоки.

В C++ ввод/вывод данных из текстовых файлов возможен как с помощью функций стандартной библиотеки языка C, так и с использованием файловых потоков. Файловый поток рассматривается как последовательность байтов, причем по умолчанию процедуры чтения выполняются с начала файла, а процедуры записи - в конец файла.

Библиотека <fstream> предоставляет пользователю класс ofstream для операций файлового вывода (Output File stream) и класс ifstream для операций ввода данных из файла (Input File stream). Сами операции ввода и вывода выполняются также, как и для других потоковых объектов - с помощью операторов >> и <<, соответственно.

Рассмотрим простейший пример организации ввода данных из текстового файла

```
ifstream infile;                // Создаем потоковый объект infile ...
                                // ... класса ifstream для чтения из файла
infile.open("my_books.txt");    // Открываем поток, связывая его с файлом

infile >> N;                    // Считываем данные в переменную N

infile.close();                 // Закрываем файловый поток
```

Предполагается, что файл с именем my_books.txt существует и находится на диске в одном каталоге с разрабатываемым проектом C++. В приведенном фрагменте кода производится одиночное считывание из файла. Значение помещается в переменную N.

Аналогичным образом может быть организована запись данных в файл на диске. В этом случае создается объект класса ofstream, далее файл открывается с помощью операции open, производится чтение данных, и файл закрывается командой close.

```
ofstream outfile;
outfile.open("my_collection.dat");
outfile << N;
outfile.close();
```

Отметим, что приведенные выше примеры не дают полного представления о возможностях файловых потоков ifstream и ofstream. Подробное описание режимов доступа к файлам, методов работы с файлами, различных операций чтения и записи и т.д. можно найти в справочной литературе.

2.4) Структура как аргумент и/или возвращаемое значение функции

Структуры C++ позволяют упорядочить используемые в программе данные и упростить их обработку, что особенно важно в ситуациях, когда объем обрабатываемых данных велик, а они сами имеют сложную внутреннюю организацию. Вместе с тем, в больших программных проектах часто бывает необходимо структурировать не только данные, но и код программы. Одним из традиционных способов повышения эффективности кода является использование подпрограмм. В C/C++ подпрограммы реализуются в виде функций.

Важной особенностью C/C++ является возможность передачи в функцию целой структуры в качестве аргумента (вместо отдельных полей). Запись вызова функции при этом становится более лаконичной, а вероятность появления синтаксических ошибок уменьшается. Кроме того, передача внутрь функции целой структуры позволяет за один вызов обработать содержимое всех полей.

Рассмотрим функцию, выводящую на экран монитора информацию о некоторой книге. Данные передаются в функцию посредством структуры book, определенной выше. Функция названа print_book.

```
void print_book(book abook)
{
      cout << "  Название:     ";
      cout << abook.title << endl;
      cout << "  Автор(ы):     ";
      cout << abook.authors << endl;
      cout << "  Издательство: ";
      cout << abook.publisher << endl;
      cout << "  Год выпуска:  ",
      cout << abook.year << endl;
      cout << "  Страниц:      ",
      cout << abook.pages << endl;
}
```

Здесь abook – формальный параметр – представляет собой структуру типа book. Содержимое отдельных полей и комментарии к ним выводятся на экран через cout.

Следующий фрагмент кода демонстрирует пример вызова функции print_book из основной программы. Функция вызывается в цикле несколько раз, при этом ей последовательно передаются в качестве фактического параметра элементы массива collection. Это позволяет вывести на экран информацию обо всех книгах в коллекции.

```
for (int i = 0; i < N; i++)
      print_book(collection[i]);
```

Функция C++ может не только принимать структуру в качестве параметра, но и возвращать ее как результат своей работы. Такая функция, как правило, создает новую структуру внутри себя, заполняет поля необходимыми значениями, а затем возвращает структуру целиком как результат. В следующем примере показано определение функции get_book, которая запрашивает у пользователя информацию о новой книге и возвращает ее в виде структуры book.

```
book get_book(void)
{
      book newbook;

      cout << "\nВВЕДИТЕ ДАННЫЕ О НОВОЙ КНИГЕ:\n";
      cout << "   Название    : ";
      getline(cin, newbook.title);
      cout << "   Автор(ы)    : ";
      getline(cin, newbook.authors);
      cout << "   Издатель    : ";
      getline(cin, newbook.publisher);
      cout << "   Год выпуска : ";
      cin >> newbook.year;
      cout << "   Страниц     : ";
      cin >> newbook.pages;
      cin.get();

      return newbook;
}
```

С помощью функции get_book теперь можно легко заполнить массив collection

```
for (int i = 0; i < N; i++)
      collection[i] = get_book();
```

2.5) Динамическое выделение памяти. Операторы new и delete.

C++ поддерживает динамическое выделение и освобождение памяти с использованием операторов new и delete. Эти операторы выделяют память для объектов из пула памяти, называемого свободным хранилищем.

Синтаксис операторов new и delete демонстрируют следующие примеры.

```
float *p1;              // объявляем указатель на вещественную переменную
p1 = new float;         // и динамически выделяем для нее память

int *p2 = new int[5];   // выделяем память под массив из 5-ти элементов int

book *p3 = new book[7]; // выделяем память под массив из 7-ти структур book

...

delete p1;       // освобождаем память под указателем p1 (4 байта)

delete[] p2;     // освобождаем память под массивом p2 (4x5=20 байт)

delete[] p3;     // освобождаем память под массивом страктур
```

Из приведенных примеров видно, что для динамического выделения памяти под массив в операторе new достаточно указать тип элементов и их количество в квадратных скобках. С помощью этого оператора в памяти могут размещаться элементы как встроенных, так и пользовательских типов. В случае неудачи динамического размещения (например, при нехватке оперативной памяти) оператор new возвращает пустой указатель (NULL) или выбрасывает исключение.

Обратим внимание на синтаксис оператора delete в случаях, когда освобождается память, занимаемая массивом. В этих случаях перед указателем должны быть обязательно записаны квадратные скобки (см. примеры выше).

2.6) Пример приложения Visual C++.

Рассмотрим пример приложения, в котором используются рассмотренные выше языковые конструкции. Приложение считывает информацию о книгах из текстового файла, выводит ее на экран, и производит поиск нужной книги по ее названию. Для хранения текстовых строк приложение использует класс string. Ввод данных с клавиатуры и вывод на экран реализован через потоковые объекты cin и cout. Некоторые операции со структурами и массивами структур оформлены в виде функций. Оперативная память под массив распределяется динамически с помощью оператора new.

```
========================= ЛИСТИНГ 2.1 =========================
```

```
#include "stdafx.h"
#include <locale>           // поддержка русского алфавита
#include <iostream>         // потоковый ввод/вывод с консоли
#include <fstream>          // файловые потоки
#include <string>           // текстовые строки C++
#include <Windows.h>        // решение проблем кодировки текста

using namespace std;

struct book                 // определение структуры book
{
	string title;
	string authors;
	string publisher;
	int year;
	unsigned int pages;
};

// Функция read_book реализует считывание информации
// о некоторой книге из файлового потока. Ссылка на
// файловый поток передается как параметр. Считанная
// информация записывается в новую структуру book.
// Структура возвращается в вызывающую программу.

book read_book(ifstream &file)
{
	book abook;                         // создаем новую структуру
	getline(file, abook.title);         // считываем данные из файла
	getline(file, abook.authors);
	getline(file, abook.publisher);
	file >> abook.year;
	file >> abook.pages;
	file.get();
	return abook;                       // возвращаем результат
}
```

```
// Функция print_book выводит на экран информацию
// о книге, переданную ей в качестве параметра.
// Никакого результата не возвращает.

void print_book(book abook)
{
	cout << "\n  Название:     ";		// используем поток cout
	cout << abook.title << endl;		// для вывода на экран
	cout << "  Автор(ы):     ";		// содержимого всех полей
	cout << abook.authors << endl;		// структуры abook
	cout << "  Издательство: ";
	cout << abook.publisher << endl;
	cout << "  Год выпуска:  ",
	cout << abook.year << endl;
	cout << "  Страниц:      ",
	cout << abook.pages << endl;
}

// Функция find_book производит поиск книги в массиве по ее названию.
// Аргументами функции являются: 1) массив структур book, 2) число
// элементов в массиве, 3) название искомой книги.
// Функция не возвращает результата. В случае, если поиск завершен
// успешно, на экран выводится полная информация о найденной книге.
// В случае неудачи на экран выводится соответствующее сообщение.

void find_book(book acollection[], int num, string atitle)
{
	bool found = false;			// флаг found показывает, найдена ли книга
	for (int i = 0; i < num; i++)		// перебираем все элементы массива
	{
		if (acollection[i].title == atitle)	// книга найдена?
		{
			found = true;			// устанавливаем флаг
			cout << "\n\nНАЙДЕНА КНИГА:";	// и выводим на экран
			print_book(acollection[i]);	// информацию о книге
		}
	}
	if (!found)					// если книга не найдена
		cout << "\n\nКНИГИ С ТАКИМ НАЗВАНИЕМ НЕ НАЙДЕНЫ!\n";
}


int N;				// глобальная переменная - число книг в коллекции
book *collection;		// указатель для размещения массива


int _tmain(int argc, _TCHAR* argv[])
{
	setlocale(LC_ALL, "rus");

	ifstream infile;			// файловый поток для ввода данных
	infile.open("my_books.txt");		// открываем файл my_books.txt

	infile >> N;				// считываем из файла число книг N
	infile.get();				// очищаем буфер для чтения

	collection = new book[N];		// распределяем в памяти массив из N структур

	for (int i = 0; i < N; i++)			// в цикле считываем из файла
		collection[i] = read_book(infile);	// данные о каждой книге

	infile.close();				// закрываем файловый поток

	for (int i = 0; i < N; i++)			// в цикле выводим на экран
		print_book(collection[i]);		// информацию о всех книгах
```

```
        string find_title;                  // новая текстовая строка
        cout << "\n\n Введите название искомой книги - ";
        SetConsoleCP(1251);                 // переключаем кодовую страницу
        getline(cin, find_title);           // считываем название искомой книги
        SetConsoleCP(866);                  // возвращаем кодовую страницу

        find_book(collection, N, find_title);       // вызываем функцию поиска книги

        delete[] collection;                // освобождаем выделенную ранее память

        system("pause");                    // задержка завершения программы
        return 0;
}
```

==========

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ №2.

1) Реализуйте описанное в лабораторной работе №1 индивидуальное задание с помощью новых возможностей языка C++. А именно, используйте:
   - класс string для работы с текстовыми строками,
   - потоковые объекты cin и cout для ввода/вывода данных с консоли,
   - классы ifstream и ofstream для ввода/вывода данных из файлов,
   - динамическое выделение памяти под массив структур с помощью new и delete.

2) Реализуйте ввод исходных данных (массива структур) из текстового файла. Текстовый файл должен содержать информацию о количестве записей и, последовательно, значения всех полей. Оформите чтение данных из файла в виде отдельной процедуры.
3) Реализуйте вывод на экран всего массива структур, оформив в виде отдельной процедуры.
4) Реализуйте дополнительные функции из своего варианта задания (см. работу №1), оформив их в виде отдельных функции.

Содержание отчета по лабораторной работе №2.

Отчет по лабораторной работе №2 (в тетради студента) должен содержать:
1) Стандартную «шапку» 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
   - определение используемой структуры,
   - размещения массива структур в динамической памяти,
   - функций, реализующих индивидуальное задание к Вашему варианту (2 функции).