МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный электротехнический
университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C++

Учебно-методическое пособие

Санкт-Петербург
Издательство СПбГЭТУ «ЛЭТИ»
2021

УДК 004.42
ББК З 973.2-018я7
Ч74

Чмиленко Ф. В., Бондарь А. С., Хоршев А. А.

Ч74 Основы программирования на языке С++: учеб.-метод. пособие СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2021. 40 с.

ISBN 978-5-7629-2889-2

Содержит материалы по дисциплинам «Введение в информационные технологии» и «Информационные технологии» к лабораторным занятиям и курсовой работе для приобретения студентами базовых навыков программирования на языке высокого уровня С++.

Предназначено для студентов, обучающихся по направлениям подготовки бакалавров 13.03.02 «Электроэнергетика и электротехника» и 27.03.04 «Управление в технических системах».

УДК 004.42
ББК З 973.2-018я7

Рецензент – канд. техн. наук, доц. В. С. Добуш (СПбГУ «Горный»).

Утверждено
редакционно-издательским советом университета
в качестве учебно-методического пособия

ISBN 978-5-7629-2889-2

Лабораторная работа 1.
ЗНАКОМСТВО С ИНТЕГРИРОВАННОЙ СРЕДОЙ ПРОГРАММИРОВАНИЯ

Цель работы – получение базовых навыков разработки программного обеспечения в интегрированной среде программирования C++ Builder.

1.1. Порядок выполнения работы

1.1.1. Создание каталогов

На диске необходимо организовать структуру каталогов (папок) для сохранения данных своих проектов. В своем рабочем каталоге создайте каталог Projects. В нем следует создавать подкаталоги по мере выполнения лабораторных работ, к примеру Lab 1, Lab 2 и т. д. Если при выполнении одной лабораторной требуется создать несколько проектов, то подкаталоги для них можно назвать Lab 1, Lab 1_2, Lab 1_3 и т. д. Рекомендуется использовать только символы латинского алфавита, цифры, пробелы и подчеркивания. Эта рекомендация относиться также к именам родительских каталогов, вплоть до корневого.

1.1.2. Создание проекта

Запустите программу C++ Builder. Для создания консольного проекта нужно в меню File выбрать пункт New → Console Application, после чего откроется окно New Console Application. В этом окне можно выбрать язык кода (*C* или *C++*), поддержку многозадачности (Multi Threaded), а также подключить к проекту невизуальные элементы визуальных библиотек (VCL и FireMonkey). Кроме того, если консольная программа была создана в другой среде программирования, то ее можно импортировать в новый проект, указав в поле Specify Project Source.

Для создания своего консольного приложения выберите язык *C++*, отключите поддержку многозадачности, а поле Target Framework установите в None и нажмите кнопку OK. После выполнения этих действий раскроются окна среды программирования.

Для консольного приложения основными являются окно редактора кода (расположено в центре экрана) и менеджер проектов Projects (расположенный в правом верхнем углу). В редакторе кода по умолчанию создается шаблон консольного приложения с именем модуля File1.cpp. Менеджер проектов управляет созданием программ, он позволяет выбрать отладочный режим,

платформу для исполняемого кода, а также объединить ряд программ в программную группу.

Для минимизации проекта удалите с помощью менеджера проектов вспомогательный файл Project1PCH1.h. Для этого щелкните по нему правой клавишей мыши и выберите во всплывающем меню команду Remove From Project и подтвердите свое действие.

Далее следует сохранить проект в каталоге Lab 1. Для этого в меню File надо вызвать команду Save Project As…. Среда программирования вначале запросит новое имя и путь для несохраненного модуля File1.cpp. Следует сохранить этот файл в каталоге Lab 1, дав ему некоторое наименование, например Lab1. Далее будет запрошено имя проекта для записи, его можно тоже сохранить под именем Lab1.

1.1.3. Написание кода программы в стиле языка C

Перед созданием своей программы следует полностью удалить предложенный средой программирования шаблон программы в редакторе кода.

Для отображения информации в разрабатываемой программе необходимо подключить стандартную библиотеку ввода/вывода, поэтому в редакторе кода в первой строке надо написать следующую директиву препроцессора.

```
#include <stdio.h>
```

Далее следует поместить код функции main. Это обязательная функция для языков программирования *C/C++*, она соответствует точке входа в программу.

```
int main()                      //точка входа в программу
{
    printf("Hello world!");     //вывод приветствия
    return 0;                   //выход из функции
}
```

1.1.4. Компиляция, построение и запуск программы

Компиляция модуля осуществляется с помощью команды меню Project → Build имя файла или набором клавиш Alt+F9. Во время компиляции среда программирования выводит информацию о текущем статусе процесса. После успешной компиляции в рабочих каталогах появляется объектный файл с расширением .obj или .o, в противном случае будет выведена информация об ошибках компиляции данного модуля.

Запустить построение исполняемого файла (программы) можно с помощью команды меню Project → Make имя проекта или набором клавиш Ctrl+F9. Если какие-то модули не были скомпилированы или в них произошли изменения, для них будет вызвана компиляция. Во время построения образа задачи среда программирования выводит информацию о текущем статусе процесса. После успешного построения программы в рабочих каталогах появятся вспомогательные файлы, порожденные процессом сборки, и исполняемый файл с расширением .exe.

Существует вариант построения исполняемого файла с обязательной компиляцией всех модулей с помощью команды меню Project → Build имя проекта или набором клавиш Shift+F9.

Запустить исполняемый файл можно с помощью команды меню Run → Run или клавишей F9. Если для запуска исполняемого файла требуется выполнить и компиляцию, и построение, то они будут выполнены автоматически при нажатии клавиши F9.

Так как среда программирования C++Builder не делает паузы после выполнения исполняемого файла, то приведенный пример программы отработает и быстро закроется. Поэтому приведенный ранее программный код требует доработки.

1.1.5. Доработка программы

Для анализа результатов работы программы можно сделать принудительную остановку с помощью функции system с аргументом pause. Для этого перед функцией main следует с помощью директивы препроцессора #include добавить описание стандартной библиотеки:

```
#include <stdlib.h>  //разрешает вызов функции system
```

Перед инструкцией return вставить вызов системной паузы:

```
system("pause");
```

После внесенных изменений соберите и запустите исполняемый файл с помощью команды меню Run → Run или клавишей F9.

Чтобы разделить на экране выводимый текст на строки, необходимо добавить к выводимому тексту управляющую последовательность *n* для перевода строки:

```
printf("Hello world!\n");     //добавлен перевод строки
```

1.1.6. Работа с кириллицей

Для использования кириллицы в консольном приложении нужно воспользоваться функцией system, которой в качестве параметра передается кодировка 1251:

```
system("chcp 1251");
```

Вставьте эту строчку первой внутри функции main. Чтобы проверить работу кириллицы добавьте ниже внутри функции main еще строчку, выводящую кириллический текст:

```
printf("Привет Мир!\n");
```

Если вместо кириллицы на экране будут выводиться непонятные символы, то надо настроить шрифт окна консоли. Для этого нажмите правой клавишей мыши на верхней рамке окна консоли и выберете команду меню Свойства. В открывшемся окне выберите закладку Шрифт и задайте другой шрифт.

1.1.7. Создание проектной группы

Среда программирования C++ Builder позволяет несколько проектов объединять в проектные группы. Для этого в окне менеджера проектов надо нажать кнопку Add new project… (кнопка с плюсом) и выбрать в открывшемся окне CBuilder → Console Application…, после чего появится окно New Console Application из п. 1.1.2., в котором надо нажать кнопку OK.

Далее, как и в п. 1.1.2., для минимизации проекта надо удалить с помощью менеджера проектов вспомогательный файл Project1PCH1.h и записать новый проект с помощью команды меню File → Save Project As… в каталог Lab 1_2. Модуль File1.cpp и имя нового проекта можно задать как Lab 1_2. Среда разработки затребует имя для проектной группы, которую можно назвать FirstLab и записать в каталог первого проекта Lab 1.

1.1.8. Написание кода программы в стиле языка C++

Также как в п. 1.1.3. следует полностью удалить предложенный средой программирования шаблон в модуле Lab 1_2.cpp и вставить вариант предыдущей программы в стиле *C++*:

```
#include <iostream>  //библиотека ввода/вывода C++
#include <stdlib.h>

using namespace std; //использовать пространство имен
                     //стандартной библиотеки std
```

```
int main()
{
    system("chcp 1251");
    cout << "Hello World!" << "\n";
    cout << "Привет Мир!" << endl;
    system("pause");
    return 0;
}
```

1.1.9. Поиск и исправление синтаксических ошибок в программе

Закомментируйте первую строчку с директивой #include, таким образом искусственно внеся ошибку в текст программы. Затем запустите компиляцию и проанализируйте сообщение об ошибках. Верните программу в исходное состояние, повторите эксперимент, но теперь закомментируйте строчку с инструкцией using namespace. Далее снова повторите эксперимент удалив точку с запятой после следующей строки:

```
cout << "Привет Мир!" << endl
```

Запустите компиляцию программы, попробуйте понять смысл сообщения об ошибке. Спозиционируйте редактор на строку с ошибкой с помощью двойного щелчка на сообщении об ошибке.

1.2. Дополнительные задания

1.2.1. Онлайн компиляторы

Найдите в сети интернет несколько онлайн компиляторов для языка *C++*. Протестируйте их с помощью простых программ, предложенных в этой работе. Сравните возможности и поддерживаемые стандарты языков *C* и *C++*.

1.2.2. Другие среды разработки

Сравните C++ Builde*r* с какой-нибудь другой средой разработки, поддерживающей язык *C++*. Создайте в ней проект и добавьте в него файл с написанным в C++ Builder программным кодом (без применения copy/paste). В качестве альтернативной среды программирования может выступать MS Visual Studio, Embarcadero Dev-C++ или любая другая среда, поддерживающая язык программирования *C++*.

Содержание отчета

В отчете приведите выводы о назначении и возможностях всех изученных в работе элементов среды C++ Builder. Просмотрите содержимое каталогов, которые порождаются средой программирования и приведите перечень файлов с их расширениями. Укажите назначение известных файлов.

Приведите сравнительный анализ возможностей протестированных онлайн компиляторов языка *C++*.

Приведите сравнительный анализ C++ Builder с другой средой разработки. Укажите найденные различия при создании консольных приложений.

Лабораторная работа 2.
ПЕРЕМЕННЫЕ И ВСТРОЕННЫЕ ТИПЫ ДАННЫХ

Цель работы – исследование встроенных типов данных языка *C++*.

2.1. Порядок выполнения работы в программе datatypes

2.1.1. Диапазоны целых чисел

С помощью программы datatypes исследуйте представление целых чисел для различных типов данных (2 – 3 для одного типа). Для этого в самом нижнем поле программы введите целое число и затем переведите его в двоичное представление (кнопка стрелочка вверх).

Определите минимальные и максимальные допустимые значения для целых типов данных (со знаком и без знака), а также их двоичное представление. Для этого в соответствующих полях программы следует изменить значение битов с 1 на 0 или с 0 на 1.

Затем проверьте переполнение при увеличении максимального или уменьшении минимального допустимых значений. Для этого следует ввести исследуемое число в двоичном или десятичном формате (в последнем случае необходимо перевести число в двоичный формат кнопкой стрелочка вверх). Далее для максимального значения следует добавить единицу, а для минимального вычесть единицу, используя соответствующие кнопки. Полученные результаты объяснить.

2.1.2. Числа с плавающей точкой одинарной точности

С помощью программы datatypes исследуйте представление чисел с плавающей точкой типа float. У чисел этой точности 1 бит отвечает за знак числа, 8 бит за порядок и 23 бита составляют мантиссу. Проверьте формулу

вычисления значения числа с плавающей точкой для одинарной точности (мантисса и порядок должны быть приведены к десятичному виду):

$$(-1)^{\text{знак}}*(1+\text{мантисса}/2^{23})*2^{(\text{порядок}-127)}.$$

Определите минимальное и максимальное значения и их двоичное представление.

Исследуйте влияние разрядности мантиссы на точность для 4–5 чисел. Для этого введите числа с плавающей точкой и большим числом значащих цифр, например 123456789, 222.222222, 0.444555666 и т. п. Преобразуйте их в двоичный вид, а затем обратно в десятичный (кнопки стрелочка вверх и стрелочка вниз). Сравните полученный результат с исходным представлением. Определите максимальное количество значащих цифр.

2.1.3. Числа с плавающей точкой двойной точности

С помощью программы datatypes проведите исследование представление чисел с плавающей точкой типа double аналогичные предыдущим. Для этого типа 1 бит отвечает за знак числа, 11 бит за порядок и 52 бита составляют мантиссу. Формула для вычисления чисел двойной точности:

$$(-1)^{\text{знак}}*(1+\text{мантисса}/2^{52})*2^{(\text{порядок}-1023)}.$$

2.1.4. Представление символов

С помощью программы datatypes выполнить исследование представления символов в целочисленном виде. Для этого выберите тип unsigned char (символы). Следует зафиксировать внутреннее представление для нескольких символов, а также перевести его в десятичный вид. Желательно чтобы в исследовании были представлены и буквы, и цифры.

2.2. Порядок выполнения работы в среде программирования

2.2.1. Определения характеристик общих типов переменных

Создайте новый проект консольного приложения. Не забудьте добавить в него заголовочный файл библиотеки ввода/вывода языка *C++* <iostream> и включите пространство имен std. Далее с помощью директивы #include добавьте заголовочный файл limits.h, описывающий определения характеристик общих типов переменных.

Затем напишите в теле функции main код, выводящий информацию об основных типах данных.

Пример кода:

```
int main() {
    system("chcp 1251");
    cout << "Мин. и макс. значение типа char: "
         << SCHAR_MIN << "\t" << SCHAR_MAX << endl;
    ...
    system("pause");
    return 0;
}
```

Изучите характеристики других типов данных. Информацию о константах файла limits.h можно подчерпнуть в википедии по адресу https://ru.wikipedia.org/wiki/Limits.h.

2.2.2. Определение и инициализация локальных переменных

Добавьте новый проект и разработайте программу, в которой в самом начале тела функции объявите переменные различных типов (bool, char, short, long, float, double). Некоторые из них следует инициализировать. Внутри функции main выведите значение объявленных переменных.

Пример кода:

```
int main(){
int a;                      //объявление переменной a
double b = 25.34123495;   //инициализация переменной b
   cout << "a = " << a << endl;
//для setprecision добавьте заголовочный файл <iomanip>
   cout << setprecision(10) << "b = " << b << endl;
   ...;
}
```

Используйте оператора sizeof() для определения размера внутреннего представления в байтах для созданных переменных:

```
cout << "размер переменной a = " << sizeof(a)
    << " байт(а)" << endl;
```

2.2.3. Область видимости и время жизни переменных

Создайте новый проект. Объявите и инициализируйте три переменные в начале функции main. Ниже создайте два внутренних блока {...} внутри тела функции main.

Внутри первого блока объявите и инициализируйте локальную переменную с именем, которое совпадает с именем одной из переменных, объявленных сначала. Внутри второго блока проделайте такие же действия, только имя локальной переменной теперь должно совпадать с другой переменной.

С помощью оператора присваивания выполните изменение первоначальных значений переменных в каждом внутреннем блоке и между ними. Выведите значения всех переменных после изменения их значений. Разработанный программный код и результаты вывода представить в отчете.

Пример кода:

```
int main() {
int a = 1;            //объявление и инициализация
   ...
   cout << "a = " << a << endl;
   {              //первый внутренний блок
      int a = 0; //объявление и инициализация
      ...
      cout << "a = " << a << endl;
   }
   cout << "a = " << a << endl;
   ...
   {              //второй внутренний блок
      a = 3;      //изменение значения
      ...
   }
   cout << "a = " << a << endl;
   ...
}
```

Дополнительное задание

При использовании оператора присваивания, если операнды, входящие в выражения имеют различные типы, происходит явное или неявное преобразование типов. Для некоторых комбинаций типов, в этом случае, возможна потеря точности или даже значения. Допишите в разработанную программу 3–4 примера, демонстрирующих результат оператора присваивания с потерей значения или точности для разного сочетания типов. Для вывода вещественных чисел воспользуйтесь манипулятором setprecision().

Содержание отчета

Привести в отчете в разделе «Наблюдения» результаты исследования типов данных в программе datatypes, в том числе двоичные представления различных чисел и символов. Выводы по этой части работы должны содержать обоснование возможностей применения в программах исследованных типов данных, а также описание их недостатков.

Приведите в отчете выводы, в каких случаях необходима информация о характеристиках встроенных типов переменных, а также информация возвращаемая оператором sizeof().

Объясните, по какому правилу разрешаются противоречия для переменных с одинаковыми именами.

В отчет включить код всех разработанных программ.

Лабораторная работа 3.
ОПЕРАТОРЫ ЯЗЫКА С++

Цель работы – исследование стандартных операторов языка *C++*.

3.1. Порядок выполнения работы

3.1.1. Арифметические операторы

Создайте программу с вводом чисел и их обработкой с применением арифметических операторов (+, –, *, /, %) и выводом результата. Программа должна демонстрировать работу с различными арифметическими операторами как с целочисленными переменными, так и с вещественными.

Пример кода:

```
int x;            //объявление целочисленной переменной
cout << "Введите целое число: ";
cin >> x;                       //ввод целого числа
int y = (x*3)%17;               //арифметическое выражение
cout << "Ответ y = " << y << endl;//вывод целого числа
```

3.1.2. Операторы отношения

Добавьте новый проект и разработайте программу, показывающую работу операторов отношения (==, !=, <, >, <=, >=). Создайте примеры (не мене трех), в которых осуществляется ввод двух целых чисел, сравнение с применением необходимых операторов и вывод результата. Для разнообра-

зия результаты можно присваивать целочисленным или логическим (тип bool) переменным:

```
int y = (x1 >= x2);    // пример сравнения
```

Логические данные (типа bool) можно вводить и выводить как в численном виде, так и в строковом, используя манипуляторы ввода/вывода такие, как boolalpha и noboolalpha:

```
bool b = true;
cout << boolalpha << b << endl;   //результат "true"
cout << b << endl;                //результат "1"
```

3.1.3. Логические операторы

Добавьте новый проект и разработайте программу, демонстрирующую возможности логических операторов. Придумайте несколько примеров, где вводятся с клавиатуры целые числа, к которым затем применяются логические операторы: && (и), || (или), ! (отрицание). Для конструирования сложных выражений используйте дополнительно скобки и операторы отношения. Выведите результаты работы примеров на консоль.

3.1.4. Побитовые операторы

Добавьте новый проект и разработайте программу, в которой надо продемонстрировать возможности побитовых операторов. Для этого надо реализовать ввод целых чисел с клавиатуры, к которым затем применяются следующие побитовые операторы: & (и), | (или), ^ (исключающее или), ~ (инверсия), << (побитовый сдвиг влево), >> (побитовый сдвиг вправо) и вывести результат на консоль (не менее шести примеров). Среди примеров необходимо реализовать такие, в которых оба операнда вводятся с клавиатуры.

3.1.5. Операторы инкремента и декремента

Добавьте новый проект и разработайте программу, для демонстрации различий между префиксной и постфиксной формой операторов инкремента и декремента. Для этого реализуйте несколько примеров (не менее четырех), где целое число вводиться с клавиатуры, далее применяется оператор инкремента или декремента, совместно с оператором присваивания, а затем результат выводиться на консоль.

3.2. Знакомство с отладчиком

Добавьте новый проект и вставьте в начало функции main приведенный код:

```
int a = 0;
a = 3;
int b = 2;
a = b+2;
a = 3;
a++;
a = a << b;
```

Зафиксировать точку останова на строке с кодом *a = 3*. Для этого левой кнопкой мыши щелкнуть левее номера указанной строки. Строка будет выделена серым цветом, а слева от номера строки появится красный кружок.

Скомпилируйте и запустите программу (кнопка F9). Наблюдайте остановку программы в указанной точке. Слева в окне отладчика Local Variables будут отображаться имена локальных переменных и их текущие значения.

Вызовите диалоговое окно Evaluate/Modify командой меню Run → Evaluate/Modify или набором клавиш Ctrl + F7. В поле Expression введите имя переменной *«a»*, в поле New Value задайте новое значение.

Далее пройти в пошаговом режиме (кнопка F8) последующие строки программы. Наблюдать изменения значения переменной *«a»*.

В дальнейших лабораторных работах надо стараться использовать отладчик для поиска логических (несинтаксических) ошибок в программах.

Дополнительное задание

Допишите в разработанные программы примеры с использованием всех составных операторов присваивания (+=, –=, *=, =/, =%, &=, |=, ^=, <<=, >>=).

Содержание отчета

В отчет включить код всех разработанных программ. Привести примеры их работы. Сделать выводы по возможностям и особенностям применения изученных операторов языка *C++*.

Лабораторная работа 4.
ИНСТРУКЦИИ ЯЗЫКА C++

Цель работы – исследование инструкций языка *C++* для управление программным потоком.

4.1. Порядок выполнения работы

4.1.1. Инструкция условного перехода if

Создайте программу с инструкциями if, if - else. Реализуйте несколько примеров с разной структурой вложенности инструкции if - else.

Чтобы работа программы была более разнообразной, используйте ввод с клавиатуры. Для демонстрации того, что происходит управления программным потоком, выводите информацию на консоль.

4.1.2. Оператор ?:

Расширьте свою программу примерами с тернарным оператором ?:. Приведите несколько примеров, когда этот оператор может быть более оптимальным, чем инструкция if - else.

4.1.3. Инструкция выбора switch

Добавьте новый проект и разработайте программу с инструкцией switch. Обязательно используйте ключевые слова break и default. Приведите примеры, когда применение инструкции switch более предпочтительно, чем вложенная конструкция if - else.

4.1.4. Инструкции циклов while *и* do while

Добавьте новый проект и разработайте программу с инструкцией while. Приведите также несколько примеров с использованием инструкций continue и/или break. Для демонстрации итераций выведите необходимую информацию на консоль в теле цикла.

Расширьте свою программу примерами с инструкцией do while.

4.1.5. Инструкции цикла for

Добавьте новый проект и разработайте программу с инструкцией for. Приведите варианты с увеличением счетчика цикла, а также с его уменьшением. В качестве примера можно использовать расчет суммы ряда и т. п. Желательно в некоторых примерах использовать инструкции continue и/или break.

Расширьте свою программу и перепишите все варианты, разработанные с помощью инструкции for, с помощью инструкции while.

4.1.6. Инструкция безусловного перехода goto

Добавьте новый проект и разработайте программу с применением инструкции goto (реализовать совместно с инструкцией if - else). Попробуйте также привести пример с «обоснованным» применением инструкции goto.

Дополнительное задание

Допишите в разработанные программы примеры, демонстрирующие возможности задавать объявление переменной прямо в условии инструкции if и в выражении инструкции switch, которые появились в стандарте *C++17*. Объясните преимущества, которые дают эти нововведение.

Содержание отчета

В отчет включить код всех разработанных программ. Привести примеры их работы. Сделать выводы по возможностям и особенностям применения изученных инструкций языка *C++*.

Лабораторная работа 5.
СОСТАВНЫЕ ТИПЫ ДАННЫХ

Цель работы – исследование составных типов данных языков *C* и *C++*.

5.1. Порядок выполнения работы

5.1.1. Перечисления

Разработайте программу, в которой необходимо определить свой перечисляемый тип данных, наподобие представленного:

```
enum Animal {Cat, Dog, Tiger, Fox, Elephant};
```

Создайте переменную этого типа и выведите значение переменной на консоль. Объясните полученное значение. Определите с помощью оператора sizeof() размер перечисляемого типа данных.

Расширьте свою программу примером с явной инициализацией констант перечисляемого типа. Для большей наглядности преимуществ перечисляемого типа создайте пример с использованием инструкции switch.

5.1.2. Одномерные массивы

Добавьте новый проект и разработайте программу, в которой надо создать и частично инициализировать одномерный массив одного из типов данных. Далее в цикле while необходимо организовать вывод на консоль значения элемента массива, индекс которого задается с клавиатуры. Следует также предусмотреть случай некорректно заданного индекса и выход из цикла.

Пример кода:

```
int index;
cout << "Введите индекс: ";
cin >> index;
cout << numbers[index] << endl; //numbers – имя массива
```

Объясните выводимое значение для неинициализированных элементов массива.

Расширьте свою программу примером с использованием цикла for для выполнения некоторых действий со всеми элементами одномерного массива. К примеру, можно найти минимальное значение либо сумму всех элементов, либо добавить ко всем элементам единицу.

5.1.3. Многомерные массивы

Добавьте новый проект и разработайте программу, в которой надо создать и инициализировать двухмерный массив одного из типов данных. Далее в цикле while необходимо организовать вывод на консоль значения элемента массива, два индекса которого задаются с клавиатуры. Необходимо предусмотреть выход из цикла, а также случай, когда какой-нибудь из индексов задан некорректно.

Примеры инициализации:

```
int numbers[3][2] = {{11, 12}, {21, 22}, {31, 32}};
```

или

```
int numbers[3][2] = {11, 12, 21, 22, 31, 32};
```

Расширьте свою программу примером с использованием вложенной конструкции из двух циклов for для выполнения некоторых действий со всеми элементами двумерного массива.

5.1.4. Структуры

Добавьте новый проект и разработайте программу, в которой следует определить составной тип данных – структуру, внутри которой используется не менее трех встроенных типов. Создайте переменные данного типа. В одних примерах продемонстрируйте инициализацию полей структуры, а в других заполнение полей с помощью ввода с клавиатуры. Определите с помощью оператора sizeof() размер, занимаемый структурой.

Пример кода:

```
struct Data {short a; long b; bool c;};
Data data;
cout << sizeof(data) << '\t' << sizeof(data.a) << '\n';
```

Расширьте свою программу примерами (не менее двух) с похожими структурами, но в которых поля расположены в другом порядке (упорядоченных и не упорядоченных по размеру типа). Определите с помощью оператора sizeof() размер этих структур. Для объяснения различий, можно использовать результаты работы оператора alignof(), который был введен в *C++11*.

5.1.5. Объединения

Добавьте новый проект и разработайте программу, в которой следует определить сложный тип данных – объединение, где внутри используется не менее трех встроенных типов. Создайте переменную данного типа и реализуйте заполнение ее полей, выводя информацию о значении всех полей объединения.

Пример кода:

```
union Data { short a; double b; long c;};
Data data;
data.a = 2;
cout << data.a << endl;
data.b = 5.23;
cout << data.b << endl;
data.c = -3;
cout << data.a << endl;
cout << data.b << endl;
cout << data.c << endl;
```

Определите с помощью оператора sizeof() размер, занимаемый объединением и объясните полученные результаты.

5.2. Дополнительные задания

5.2.1. Инструкция цикла range based for

Для задания с одномерными массивами допишите программный код, демонстрирующий применение нового варианта цикла for, который был добавлен в стандарт *C++11*.

5.2.2. Инициализация полей структуры

Для задания со структурами добавьте примеры, показывающие новые возможности стандартов *C++11* и *C++20*. А именно, инициализацию по умолчанию полей структуры (direct member initializers) и назначенную инициализацию (designated initializers).

*5.2.3. Применение структурных привязок (*structured bindings*)*

Для задания со структурами добавьте создание массива структур. Далее напишите пример, использующий цикл range based for для перебора элементов этого массива. Для удобства работы внутри тела цикла с полями структуры используйте структурные привязки, добавленные в *C++17*.

Содержание отчета

В отчет включить код всех разработанных программ. Привести примеры их работы. Сделать выводы по исследованным возможностям составных типов данных языка *C++*.

Лабораторная работа 6.
ФУНКЦИИ

Цель работы – поучение навыков разработки пользовательских функций и освоение модульного принципа программирования.

6.1. Порядок выполнения работы

6.1.1. Объявление, определение и использование функций

Разработайте программу, состоящую из одного модуля, в которой надо определить свою функцию, получающую несколько параметров (по значению) и возвращающую рассчитываемое значение. Во время расчетов необходимо использовать какую-либо константу, наподобие числа π. Примером

может служить функция, рассчитывающая площадь поверхности цилиндра по заданному радиусу основания и высоте $S = 2\pi R^2 + 2\pi RH$.

Объявление функции расположите до функции main.

Пример объявления:

```
double Surface(double R, double H);
```

Включите вызов функции в тело функции main.

Пример вызова функции:

```
int main()
{
   double r, h;
   cin >> r;
   cin >> h;
   double s = Surface(r, h); //вызов функции
   cout << s;
}
```

Определение функции расположите после функции main.

Пример определения:

```
double Surface(double R, double H)
{
   double s1 = 2*3.14*R*R;
   double s2 = 2*3.14*R*H;
   return s1+s2;
}
```

Исследуйте необходимость объявления функции. Закомментируйте объявление и скомпилируйте программу заново. Затем попробуйте перенести определение своей функции выше функции main и снова скомпилируйте программу. По наблюдениям сделайте выводы.

6.1.2. Модульный принцип программирования

Добавьте новый проект и разработайте программу, аналогичную предыдущей, но состоящую из нескольких модулей main.cpp и func.cpp. Для модуля func.cpp потребуется еще файл заголовка func.h, который будет содержать объявление разработанной функции. Добавить новый модуль можно с помощью команды меню File → New → Unit - C++ Builder.

Пример файл заголовка func.h:

```
#pragma once     //защита от повторного использования
double Surface(double R, double H);
```

Модуль func.cpp содержит только определение разработанной функции.

Тело функции main можно взять из предыдущего задания и разместить его в модуле main.cpp. Теперь этот модуль не должен содержать объявление и определение разработанной функции. Вместо этого в него следует добавить директиву сопроцессора #include, подключающую файл заголовка func.h. Сделать это лучше в самом начале файла main.cpp, таким образом:

```
#include "func.h"
```

Обратите внимание, что для подключения пользовательского заголовка в директиве #include используются двойные кавычки, а не угловые скобки, как для библиотечных файлов.

Тело функции main можно взять из предыдущего задания и разместить его в модуле main.cpp.

6.1.3. Глобальные переменные

Доработайте программу из п.6.1.1. В программе вне каких-либо функций, сразу после директив препроцессора, объявите глобальную переменную, соответствующую типу вашей константы и сразу инициализируйте ее, как показано, одним из двух способов:

```
double PI = 3.14;
const double PI = 3.14;  //константная переменная
```

Замените в своей функции значение константы в явном виде на глобальную переменную.

Исследуйте, доступна ли глобальная переменная в функции main (к примеру, попробуйте задать ей более точное значение)? Добавьте еще глобальных переменных без инициализации. Выведите их значения в функции main. Чем глобальные переменные отличаются от локальных?

6.1.4. Доступ к глобальным переменным в разных модулях

Доработайте программу из п.6.1.2. Чтобы иметь возможность использовать глобальную переменную в двух модулях, ее следует объявить в каждом, но определить необходимо только в одном. Для этого в файл заголовка func.h после директивы #pragma once необходимо вставить

следующую конструкцию с использованием инструкций условной компиляции:

```
#ifdef FUNC  //если объявлен макрос FUNC
double PI = 3.14;     //определение + инициализация
#elseif
extern double PI;     //определение в другом модуле
#endif
```

Далее в начале модуля func.cpp следует добавить директиву сопроцессора #include, подключающую файл заголовка func.h. А перед ней определить макрос FUNC:

```
#define FUNC
```

Замените в своей функции значение константы в явном виде на глобальную переменную и скомпилируйте заново проект.

6.1.5. Передача параметров по ссылке

Доработайте программу и разработайте функцию типа void, которая получает по ссылке две переменные (любого типа данных) и обрабатывает их в соответствии с некоторым правилом, например, если первая переменная больше второй переменной, то функция меняет их значения местами. Для объявления и определения функции используйте файлы из предыдущего задания func.h и func.cpp.

6.1.6. Передача и возврат сложных объектов

Доработайте программу и разработайте функцию, которая получает по ссылке две переменные пользовательского типа – комплексное число MyComplex – и производит с ними некоторое математическое действие (сложение, вычитание и т. п.). Результат из функции должен возвращаться с помощью инструкции return. Пример объявления структуры:

```
struct MyComplex
{
   double real, imag;
};
```

Для объявления и определения функции создайте файлы mycomplex.h и mycomplex.cpp. Объявление структуры поместите в файл mycomplex.h. Так как пользовательский тип используется для определения функции, необхо-

димо подключить пользовательский заголовок mycomplex.h через директиву #include и в файле с функцией main, и в файле mycomplex.cpp.

6.1.7. Передача указателя в функцию

Доработайте программу и разработайте несколько функций для работы со строками в стиле *C*, которые будут получать первым параметром указатель на строку символов, а также, возможно, другие параметры, и выполнять некоторые действия. К примеру, подсчет всех символов в строке или поиск заданного символа.

Затем разработайте функцию, которая будет получать указатель на одномерный массив и размер массива (по значению) и вычислять некоторые параметры (среднее значение, минимальное значение, дисперсию, квадратичное отклонение и т. п.).

Дополнительное задание

Доработайте программу и продемонстрируйте применение лямбда-функций, которые были добавлены в стандарт *C++11*.

Содержание отчета

В отчет включить весь разработанный код. Привести примеры его работы. Сделать выводы по разным способам передачи параметров в функции. Описать преимущества модульного принципа построения программ.

Лабораторная работа 7.
ФОРМАТНОЕ ПРЕОБРАЗОВАНИЕ ДАННЫХ

Цель работы – исследование библиотечных функций для ввода/вывода и преобразования данных языка *C*.

7.1. Порядок выполнения работы

7.1.1. Преобразование целых чисел

Реализовать в программе ввод с клавиатуры и вывод на экран (ввод/вывод с консоли) целых чисел в десятичном (без знака и со знаком), восьмеричном и шестнадцатеричном форматах. Используйте для этого функции scanf и printf и форматы ”%d”, ”%u” и ”%x”. Научитесь при выводе форматировать числа (выравнивание, дополнительные пробелы или нули), для чего используйте такие компоненты формата, как флаги, ширина и точность.

7.1.2. Преобразование чисел с плавающей точкой

Реализовать в программе ввод с клавиатуры и вывод на экран вещественных чисел в обычном и экспоненциальном форматах. Используйте для этого функции scanf и printf. Научитесь работать с форматами ”%f”, ”%e” и ”%g”, задавая требуемое число разрядов до и после десятичной точки.

Реализуйте вариант, когда ширина и точность задаются пользователем с клавиатуры. Для этого в строке формата функции printf для поля «ширина и точность» примените ‘*’.

7.1.3. Преобразование символов и строк

Реализовать в программе ввод с клавиатуры и вывод на экран отдельных символов и строк. Используйте для этого функции scanf и printf и форматы ”%c” и ”%s”. Научитесь при выводе выравнивать текст (влево или вправо), усекать текст до нужного размера поля, дополнять пробелами.

Реализуйте вариант, когда в один символьный массив вводится строка, состоящая из нескольких слов, разделенных пробелом. Не забывайте контролировать переполнение массива, который вы выделили для ввода строки.

Также реализуйте ввод строки через формат ”%c”, используя ввод символов в цикле.

7.1.4. Форматированный вывод в строку

Выберите одно из заданий для пунктов 7.1.1 и 7.1.2 и реализуйте его с помощью функции sprintf. Для вывода сформированной строки используйте функцию puts.

7.1.5. Форматированный ввод из строки

Выберите одно из заданий для пунктов 7.1.1 и 7.1.2 и реализуйте его с помощью функции sscanf. Для ввода строки с данными (которая будет преобразовываться), а также для ввода строки со спецификаторами формата используйте функции gets или gets_s.

7.1.6. Форматирование таблицы строк из данных разных типов

Разработайте программу с выводом массива структур в виде таблицы. Для этого имеет смысл в программе использовать цикл. Строки и столбцы таблицы следует разграничить с помощью символов ‘|’ и ‘-‘.

Чтобы задача форматирования была сложнее, обязательно для арифметических типов проинициализируйте начальные данные положительными и отрицательными значениями.

Пример структуры:

```
struct A
{
   short a;
   double b;
   char c[16];
};

A data[] =
{
   { 200,  -25.178, "Язык C++"},
   { -34, 1234.520, "Язык FORTRAN"},
   {   0,    0.000, "Язык Pascal"}
};
```

Пример таблицы:

```
*-------------------------------*
| 200 | -25.178  | Язык C++     |
---------------------------------
| -34 | 1234.520 | Язык FORTRAN |
---------------------------------
|   0 | 0.000    | Язык Pascal  |
*-------------------------------*
```

Усложните задание. Реализуйте в цикле «ввод данных» в массив структур с консоли. Чтобы избежать переполнения массива, можно у пользователя запросить количество вводимых строк. Учтите, что их количество может быть меньше или больше, чем в статическом массиве. Самый гибкий способ реализовать эту задачу – использовать динамическую память new/delete.

7.1.7. Функции преобразования данных

Разработайте программу для исследования работы различных функций языка *C* для преобразования строк в числа (atof, atoi, atol, strtod, strtol, strtoul). Имеет смысл сравнить работу этих функций с функцией sscanf, особенно для

случаев, когда данные для преобразования заданы некорректно (не все строки можно конвертировать в численные данные).

Для ввода данных используйте функцию gets_s, а для вывода преобразованных данных printf.

Дополнительное задание

Добавьте новый проект и разработайте программу, в которой надо реализовать задание 7.1.6 с помощью форматирования потокового вывода языка *C++*.

Содержание отчета

В отчет включите весь разработанный код и приведите выводы по исследованным возможностям библиотечных функций для ввода/вывода и преобразования данных, которые вы использовали.

Лабораторная работа 8.
РАБОТА С ФАЙЛАМИ ДАННЫХ

Цель работы – освоение техники работы с файлами данных помощью библиотеки языка *C*.

8.1. Порядок выполнения работы

8.1.1. Форматированное сохранение массива сложной структуры в текстовый файл

Определите в программе свой тип структуры, включающий 5–6 полей (поля с целыми, вещественными и символьными данными). Создайте и инициализируйте в программе статический одномерный массив с этой структурой, размером около 10 записей.

Разработайте программу, которая откроет файл данных в текстовом формате с помощью функции open и сохранит в него созданный массив. Для записи данных используйте функцию fprintf внутри тела цикла for.

Первоначально имя файла можно задать статически в виде строковой константы. После успешного тестирования добавьте в программу возможность задавать имя файла с консоли (клавиатуры). Для этого используйте функции scanf, gets или fgets.

Пример структуры:

```
struct A {
   short a;
   double b;
   char c[16];
};

A data[] = {
   { 200,  -25.178, "Язык C++"},
   { -34, 1234.520, "Язык FORTRAN"},
   {   5,    0.000, "Язык Pascal"}
};
```

8.1.2. Чтение массива сложной структуры из текстового файла

Отредактируйте полученный в предыдущем задании файл с помощью текстового редактора. Добавьте или удалите строки, измените данные, не нарушая при этом структуру файла (ширину полей).

Разработайте программу для чтения отредактированного файла в массив структур с помощью функции fscanf. В связи с тем, что массив статический, его размер следует задать заведомо больше, чем количество строк в файле данных.

Имя файла следует задавать с консоли, используя функции scanf, gets или fgets. Предусмотрите ситуацию, когда запрашиваемого файла не окажется на диске.

Поскольку предполагается, что неизвестно количество записей внутри файла, читать данные лучше всего в цикле while, используя функцию проверки конца файла feof. В этом случае важную роль играет символ перевода строки ‘\n’ в конце строки форматирования функции fscanf.

Следует учесть, что для подсчета прочитанных записей необходимо определить переменную, с помощью которой следует индексировать доступ к массиву.

Пример цикла:

```
FILE* inp = fopen("data.txt", "r");
    ...
while(!feof(inp)){
   fscanf(inp, "ВАШ ФОРМАТ\n", ВАШИ ПЕРЕМЕННЫЕ);
}
```

Для вывода на консоль (дисплей) прочитанного из файла массива, добавьте в программу функцию, которой будет передаваться указатель на массив и количество считанных записей. Вызов функции вставьте сразу после закрытия файла.

8.1.3. Определение размера файла

Доработайте предыдущее задание. Определите размер выбранного для чтения файла. Для этого после того, как файл будет открыт, установите указатель на конец файла с помощью функции fseek. Получите размер файла, воспользовавшись функцией ftell, и выведите его на консоль. Обратите внимание, что для последующего чтения данных из файла указатель надо установить в начало.

8.1.4. Запись в файл в двоичном формате

Разработайте программу, которая сохраняет массив структур в бинарном (двоичном) файле (аналогично п. 8.1.1, но без преобразования в символьный формат) с помощью функции fwrite. Сравните полученный размер файла с размером файла текстового формата.

8.1.5. Чтение из файла двоичного формата

Разработайте программу, которая читает данные в массив из файла двоичного формата с помощью функции fread. Так как предполагается, что неизвестно, сколько информации содержится в файле, то возможно два варианта решения этой задачи.

В первом варианте чтение данных можно осуществить по одному элементу в цикле while с проверкой конца файла (п. 8.1.2), используя статический массив заведомо большого размера.

Второй вариант сложнее, но надежнее. Чтобы избежать переполнения массива необходимо использовать динамическую память new/delete. Для определения требуемого размера динамического массива следует определить размер файла (п. 8.1.3) и поделить его на размер структуры. В этом случае для чтения данных цикл не потребуется, и можно обойтись одним вызовом функции fread.

Для вывода на консоль прочитанного массива добавьте функцию как в п. 8.1.2, которой будет передаваться указатель на массив и количество считанных записей. Вызов функции вставьте сразу после закрытия файла.

8.1.6. Чтение элемента массива из произвольного места файла двоичного формата

Разработайте программу, которая читает один заданный элемент массива из файла. Номер элемента введите с консоли. Для этого откройте файл в бинарном формате и установите указатель на заданное смещение от начала файла с помощью функции fseek. Смещение можно вычислить по заданному в консоли номеру строки и размеру структуры. Прочтите элемент массива с помощью функции fread и выведите его на консоль.

8.1.7. Запись элемента массива в произвольное место файла двоичного формата

Разработайте программу, которая записывает одну строку массива в заданное место файла. Для этого следует определить переменную с типом вашей структуры. Введите в поля этой переменной данные с консоли, который вы потом запишите в файл с помощью функции scanf. Откройте файл и установите указатель на заданное смещение от начала файла с помощью функции fseek. Смещение можно вычислять по задаваемому в консоли номеру строки и размеру структуры. Запишите элемент массива с помощью функции fwrite.

Для проверки результата используйте программу, разработанную в п. 8.1.5.

Дополнительное задание

Добавьте новый проект и разработайте программу, в которой надо реализовать задания 8.1.1 и 8.1.2 с помощью потоковой библиотеки работы с файлами языка *C++*.

Содержание отчета

В отчет включите весь разработанный код и приведите выводы по исследованным возможностям библиотечных функций для работы с файлами, которые вы использовали.

КУРСОВАЯ РАБОТА

Цель работы – получение навыков постановки задачи, алгоритмизации, модульного принципа разработки и отладки приложений на примере создания программы для работы с базой данных в виде типизированного файла.

Задание: Создать программу, которая работает с базой данных в виде типизированного файла. Код программы должен поддерживать модульный принцип разработки (состоять не менее чем из трех модулей). При реализации программы необходимо использовать функции, массивы, структуры, указатели, выполнять форматное преобразование данных, чтение и запись в файлы.

Ваша СУБД (система управления базой данных) должна уметь выполнять следующие действия:

а) чтение данных из файла, имя которого задано пользователем;

б) просмотр данных на консоли в табличном виде;

в) редактирование записи, номер которой задан пользователем;

г) добавление записи в конец списка;

д) удаление записи, номер которой задан пользователем;

е) сортировку записей по возрастанию и убыванию;

ж) поиск записей по названию;

з) запись данных в файл, имя которого задано пользователем;

и) численную обработку данных по выбору учащегося (подсчет среднего, поиск максимального и т. п.).

Предусмотрите обработку ошибочных ситуаций:

а) на диске нет файла, имя которого задано для чтения;

б) для новой записи нет места в массиве;

в) введен неправильный номер для удаляемой или редактируемой записи.

Информация для базы данных выбирается на усмотрение учащегося. Примеры тем приведены далее. Следует отметить, что разработанная структура данных должна содержать не менее четырех разнородных полей:

1. Книжный магазин (название автора, название книги, цена, количество, кодовый номер).

2. Анализ метеоданных (страна, город, время, температура, влажность).

3. Анализ радиопередач (радиостанция, название передачи, время, продолжительность, рейтинг).

4. Расписание занятий (преподаватель, дисциплина, курс, семестр, аудитория).

5. Расписание поездов (место отправления, место прибытия, номер, тип места, цена).

6. Учет товаров на складе (поставщик, товар, кодовый номер, количество, цена).

7. Каталог художественных фильмов (режиссёр, название, год, количество копий, цена).

8. Картотека чертежей (деталь, формат, автор, год, кодовый номер).

9. Производство листового металла (завод, марка стали, объем, код продукции, цена).

10. Осветительная продукция (производитель, тип лампы, мощность, количество, цена).

11. Меню ресторана (ресторан, блюдо, тип блюда, вес, цена).

12. Поставщики электронных компонентов (производитель, изделие, код продукции, количество, цена).

Принципы оценки курсовой работы

Для повышения объективности оценки курсовой работы применяется принцип самоконтроля знаний. Студенты сами выбирают степень проработки курсового задания, от которого зависит максимальная оценка.

Удовлетворительно – это базовый уровень, который соответствует описанной постановке задачи.

Хорошо – это расширенный уровень, в котором надо реализовать все возможности базового уровня.

Дополнительные задания:

1. Для хранения считанных данных использовать в программе динамическую память. С учетом того, что в программе должен работать режим вставки, имеет смысл память выделять в программе одним большим массивом с запасом. Однако надо не забыть реализовать возможность выделения нового объема памяти, когда не осталось места для добавления новой записи. С точки зрения эффективности разработки программы, следует вначале использовать статическую память, а потом переделать ее на динамическую;

2. Реализуйте запись и чтение файла в csv-формате (данные разграничены каким-нибудь символом, например, запятой);

3. При выходе из программы предупреждать, если данные были изменены, но не сохранены.

Отлично – это продвинутый уровень, в котором надо реализовать все возможности базового и расширенного уровней.

Дополнительные задания:

1. Предусмотрите какой-нибудь вариант простого шифрования данных так, чтобы их нельзя было просмотреть в текстовом редакторе.

2. Используйте аргументы командной строки, которые доступны в функции main. Допустим, что в командной строке передается имя файла, который надо открыть сразу после запуска программы.

3. Реализовать «защиту от дурака». Программа должна адекватно реагировать на все попытки ввести неправильные данные.

4. Уменьшить повторяемость кода программы: по возможности все дублирующие места реализовать в виде вспомогательных функций.

Несоответствие проявленных при защите знаний уровню курсовой приводит к понижению оценки до «неудовлетворительно» и последующей пересдаче с возможностью получить не более 3 баллов.

Одинаковые отчеты наказываются. «Штраф» в виде понижения оценки будет применен и к тому, кто первым принес отчет (оригинал), и к тому, кто принес отчет вторым (дубликат).

Чем выше оценка, на которую претендует студент, тем больше оригинальности и креативности должно быть проявлено при разработке программы.

Рекомендации по разработке программного кода

Основной модуль программы File1.cpp

В данном примере предлагается немного избыточный подход, реализующий принцип управления посредством обработки команд. Для этого в файле заголовка File1.h описан тип enum eCMD для объявления констант соответствующих командам меню.

Кроме того, в файле заголовка определена структура TRec, которая должна содержать необходимые для описания выбранной задачи поля. Также в данном файле задана константа MAX_SIZE, с помощью которой задается размер статического массива структур (резервируется память для определенного количества записей).

Пример заготовки для файла File1.h:

```
//тип enum для реализации режима обработки команд
enum eCMD {CMD_EXIT = -1, CMD_NONE, CMD_READ, CMD_SHOW,
           CMD_EDIT, CMD_ADD, CMD_DELETE, CMD_SORT,
           CMD_FIND, CMD_SAVE};

//константа задает размер статического массива
const int MAX_SIZE = 10;

//основная структура данных
struct TRec {
   short a;
   double b;
   char c[16];
};
```

Если программа реализована с использованием статической памяти, то массив структур лучше сделать глобальным. Для удобства отладки имеет смысл его инициализировать некоторым количеством записей. Глобальной следует объявить и переменную, которая указывает на текущее количество актуальных записей в массиве.

Файл заголовка должен содержать обрамление из директив препроцессора, чтобы обеспечить включение заголовочного файла только один раз:

```
#ifndef File1H
#define File1H
...
#endif
```

Глобальные переменные в файле File1.cpp:

```
//Статический массив структур
TRec Data[MAX_SIZE] =    //размер определен константой
{
   { 200,  -25.178, "Язык C++"},
   { -34, 1234.520, "Язык FORTRAN"},
   {   5,    0.000, "Язык Pascal"}
};
//Переменная указывает количество записей в массиве
unsigned Count = 3;
```

Для удобства чтения текста программы имеет смысл основную функцию main не перегружать излишним кодом. Поэтому для выполнения основных действий, таких как вызов меню или выполнение выбранных пользователем команд, необходимо использовать разработанные для этого функции.

Пример заготовки для функции main:

```
int main(int argc, char* argv[])
{
   system("chcp 1251");
   eCMD cmd = CMD_NONE;
// цикл для обработки команд основного меню
   while(true)
   {
      if(cmd == CMD_EXIT)break; //выход из цикла
      cmd = Menu();  //отображение меню и ввод команд
      // вызов функции для соответствующей команды
      switch(cmd)
      {
        case CMD_SHOW: ShowData(Data, Count);
                       break;
        case CMD_READ: ReadDatabase(Data, Count);
                       break;
        ...
      }
   }
   puts("Работа закончена");
   system("pause");
   return 0;
}
```

Функцию Menu для вывода меню и ввода команд можно реализовать также в файле File1.cpp. Так как она вызывается только из функции main, то ее можно определить раньше, чтобы не писать отдельно объявление.

Пример заготовки для функции Menu:

```
eCMD Menu()
{
//цикл для вывода меню, если ввели неправильно команду
   while(true)
   {
   system("cls");                  //очистка экрана
   puts("Выберите действие:");     //отображение меню
   puts("1 - Открыть файл");
   puts("2 - Просмотр данных");
   ...
   puts("4 - Выход из программы");
   unsigned opt;
   fflush(stdin);          //обнуление входного потока
   scanf("%u", &opt);
   switch(opt) { //возврат из функции команды
      case 1:  return CMD_READ;
      case 2:  return CMD_SHOW;
      ...
      case 4:  return CMD_EXIT;
      default: puts("Вы ввели неправильную команду");
      system("pause");
      }
   }
}
```

Модуль для файлового ввода/вывода программы File2

Во втором модуле предлагается реализовать две функции: одну для чтения, а другую для записи данных. Для этого их надо объявить в заголовке File2.h, который следует также включить в File1.cpp.

Чтобы функции из File2.cpp знали об объявленной структуре данных в этот файл, надо включить заголовок File1.h.

Далее приводятся примеры заглушек (определение функций без рабочего тела) для функций ввода/вывода из файла File2.cpp.

```
#include "File1.h" //содержит информацию о TRec*
void ReadDatabase(TRec* data, unsigned &count)
{
//Читаем базу данных...
}
void SaveDatabase(TRec* data, unsigned count)
{
//Сохраняем базу данных...
}
```

Модуль для работы с данными File3

В третьем модуле предлагается реализовать функции для работы с данными, такие как просмотр данных, редактирование, удаление или добавление записи. Для этого их надо объявить в заголовке File3.h, который следует также включить в File1.cpp.

Чтобы функции из File3.cpp знали об объявленной структуре данных и максимальном размере массива, в этот файл надо включить заголовок File1.h.

Пример объявлений функций из заголовка File3.h:

```
void ShowData(TRec* data, unsigned &count );
void EditRecord(TRec* data, unsigned count);
void AddRecord(TRec* data, unsigned &count);
```

Реализацию в файле File3.cpp функции ShowData (отображения всех записей и меню для работы с данными) можно организовать также с обработкой команд.

Пример заготовки для функции ShowData:

```
void ShowData (TRec* data, unsigned &count)
{
   eCMD cmd = CMD_NONE;
   while( cmd != CMD_EXIT)
   {
      system("cls");

//здесь должно быть отображение всех данных
```

```
        //отображение меню и ввод команд
        cmd = MenuShow();
        //вызов функции для соответствующей команды
        switch(cmd)
        {
         case CMD_EDIT:   EditRecord(data, count); break;
         ...
        }
    }
}
```

В этом случае функция для отображения меню и ввода команд может иметь следующий прототип.

Пример заготовки для функции MenuShow:

```
eCMD MenuShow()
{
    while(true)
    {
      puts("\nВыберите действие:");
      puts("1 - Редактировать; ...); //отображение меню
    unsigned opt;
    fflush(stdin);         //обнуление входного потока
    scanf("%u", &opt);
    switch( opt ) {   //возврат из функции команды
        case 1: return CMD_EDIT;
        ...
        case 6: return CMD_EXIT;
        default: puts("Вы ввели неправильную команду");
                 system("pause");
        }
    }
}
```

Предложенная структура и функции для программы не являются обязательными. Не стоит повторять название переменных и функций: используйте названия близкие к вашей теме по смыслу. Приветствуется креативный подход к курсовой работе.

Если вы используете предложенные заготовки, то следует разбираться в назначении каждой строки. В этом случае надо понимать:

а) почему в некоторых функциях аргумент unsigned count передается по значению, а в некоторые по ссылке?

б) зачем перед вводом информации посредством scanf вызывается функция fflush(stdin)?

г) почему предложено массив Data объявлять глобальным, а не локальным в функции main?

Содержание отчета

Общий объем работы 15–20 страниц.

Требуемые разделы и их содержание:

– содержание;

– введение (цель работы, основные задачи);

– описание разработки (один или несколько разделов);

– заключение (краткое описание полученного результата);

– список использованных источников;

– приложения (содержат полный программный код, крупные рисунки и таблицы).

Использование ссылок на литературу в разделе, посвященном описанию разработки, обязательно.

СОДЕРЖАНИЕ

Чмиленко Федор Викторович,
Бондарь Алексей Сергеевич,
Хоршев Алексей Алексеевич

Основы программирования на языке C++

Учебно-методическое пособие

Редактор О. Р. Крумина

Подписано в печать 16.11.21. Формат 60×84 1/16.
Бумага офсетная. Печать цифровая. Печ. л. 2,5.
Гарнитура «Times New Roman». Тираж 170 экз. Заказ.

Издательство СПбГЭТУ «ЛЭТИ»
197376, С.-Петербург, ул. Проф. Попова, 5