Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
Санкт-Петербургский горный университет

Кафедра электронных систем

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

Методические указания к лабораторным работам для студентов бакалавриата направления 11.03.04

САНКТ-ПЕТЕРБУРГ
2021

УДК 519.682 (073)

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ: Методические указания к лабораторным работам / Санкт-Петербургский горный университет. Сост.: *А.А. Белицкий, А.В.Полукеева*. СПб, 2021. 22 с.

Методические указания предназначены для студентов бакалавриата направления 11.03.04 «Электроника и наноэлектроника», направленность (профиль) «Промышленная электроника», «Силовая электроника».

Научный редактор доц. *И.И. Растворова*

Рецензент доц. *С.Д. Дубровенский* (Санкт-Петербургский государственный технологический институт)

ГЛАВА 1. ТИПЫ ДАННЫХ. КОМАНДЫ ВВОДА-ВЫВОДА. РАБОТА С ФАЙЛАМИ

1.1 ТИПЫ ДАННЫХ

Типы данных языка С++ можно разделить на основные и составные. Используя первые данные, можно вводить описание вторых. Основные типы данных также иногда называются арифметическими, поскольку они могут использоваться в арифметических операциях. К таким типам данных относятся:

int – целые числа;

float – числа с плавающей точкой (также известные, как числа с плавающей запятой, дробные числа). Числа этого типа называются так из-за отдельного хранения мантиссы (значащих цифр числа) и показателя степени (позиции точки в этом числе);

double - числа с плавающей точкой, только размер занимаемой памяти в два раза больше, чем у типа данных *float*, что увеличивает диапазон принимаемых значений;

char – строковая переменная;

bool – логическая (булевская) переменная, описывающая истинность или ложность какого-либо утверждения (>0/= 0, true/false соответственно).

С помощью типов данных можно объявлять переменные как в теле *главной функции int main()*, т.е внутри *{ }* (локальные переменные), так и перед её началом – после подключения необходимых библиотек (глобальные переменные). Также в теле функции для остановки программы до момента нажатия любой клавиши используется команда *system("pause")*. Перед окончанием программы в конце тела основной функции используется *return 0* – команда возврата в систему целочисленного нуля.

Арифметическими операциями являются: сложение (+), вычитание (-), деление (\), умножение (*), инкремент (*a*++ – увеличение переменной а на 1), декремент (*a*-- – уменьшение переменной а на 1).

Важно также помнить, что любая команда программы (кроме строк с подключением библиотек, а также описанием ветвлений и циклов и пр.) должна заканчиваться “;”.

Примеры:

```
int a;          // объявление переменной a целочисленного типа
float b;        // объявление переменной b типа данных с плавающей запятой
char d = 's';   // инициализация переменной типа char
bool k = true;  // инициализация логической переменной k
```

1.2 КОМАНДЫ ВВОДА-ВЫВОДА

Взаимодействие приложения, написанного на любом языке программирования, с окружающим миром осуществляется посредством ввода-вывода информации. Обычно информация выводится в консоль (на монитор) или в файл.

Для осуществления ввода и вывода информаций используются специальные команды вводы-вывода:

cout – команда потокового вывода на консоль. Для работы этой команды после её написания необходимо поставить “<<”, на консоль будет выводиться то, что будет записано после этого знака. Если необходимо вывести значение имеющейся переменной, то требуется написание обозначения этой переменной. Если нужно вывести какой-либо текст, то он помещается в двойные кавычки после знака “<<”;

cin – команда потокового ввода с консоли. Для её работы после неё необходимо поставить знак “>>”. Данная команда считывает строки до первого пробельного символа (это пробел ’ ’, табуляция ’*t*’ и перевод строки ’*n*’).

Для работы этих объектов необходимо подключить библиотеку функций *iostream.h* перед телом главной функции и объявлением переменных.

Для перехода на новую строку помимо ’\n’ используется *endl*.

Примеры:

```
#include <iostream.h>          // подключение библиотеки iostream
cout << "Введите число A:";    // вывод текста на консоль
cin >> a;                      // ожидание ввода с клавиатуры и
                               запись символа в переменную
cout << a <<endl;              // вывод переменной на консоль и
                               переход на новую строку
```

1.3 РАБОТА С ФАЙЛАМИ

Иногда ввод или вывод получаются слишком объёмными или многочисленными, чтобы использовать для этого консоль. Вместо этого используют файлы, данные из которых программа должна считывать информацию, как будто это ввод с клавиатуры, или куда программа должна записывать вывод.

Для начала работы с файлом, необходимо подключить библиотеку работы с файловыми потоками *fstream.h*

Для удобства обращения к файлу путь к нему можно обозначить через строковую переменную char, заменяя “\” на “\\”.

Далее в теле главной функции создается объект класса *ofstream* (для работы с записью в файл) или *ifstream* (для работы со считыванием из файла) и через него открывается файл, в который необходимо произвести запись\считать с него данные. По окончанию работы с файлом файл необходимо закрыть.

Примеры:

```
#include <fstream.h>          // подключение библиотеки fstream
char *M="E:\\A\\1.txt" ;      // обозначение пути к файлу через переменную
ofstream out;                 // создание файлового потока с именем out для записи
ifstream in;                  // создание файлового потока с именем in для считывания
out.open(M);                  // открытие файла в потоке out
in.open(M);                   // открытие файла в потоке in
in>>a;                        // считывание переменной a с файла
out<<k<<”\n”;                 // запись переменной k и переход на новую строку в файле
out.close();                  // закрытие файла записи
in.close ();                  // закрытие файла считывания
```

Задание для самостоятельного выполнения:

1. Найти и вывести на экран и в файл:

A+B

A-B

A/B
A*B
Инкремент A, B
Декремент A, B
Переменные A и B вводится через консоль с клавиатуры.

ГЛАВА 2. ВЕТВЛЕНИЯ. ОПЕРАТОР МНОЖЕСТВЕННОГО ВЫБОРА SWITCH. ЦИКЛЫ

2.1 ВЕТВЛЕНИЯ

Иногда возникает необходимость выбора операции в ходе выполнения программы в зависимости от какого-либо условия. Для реализации этого процесса можно использовать ветвления.

Оператор *if* служит для того, чтобы выполнить операцию или последовательность команд, находящихся в фигурных скобках, в том случае, когда условие является верным. Условие всегда записывается в круглых скобках между оператором *if* и фигурными скобками. Последовательность команд, которые должны выполняться, если условие не выполняется, записывается в фигурных скобках после оператора *else* (туда, например, можно записать новое ветвление).

В качестве оператора сравнения следует использовать знак двойного равенства – “==”, так как знак одиночного равенства является знаком присваивания значения. Если поставить оператор присваивания в условии, то при проверке условия, значение переменной изменится и это условие в любом случае выполнится.

Если после операторов if или else должна выполняться только одна команда, фигурные скобки можно не ставить. Если же необходимо выполнение более одной команды, то фигурные скобки обязательны. Точка с запятой после фигурных скобок не ставится.

Пример:

```
if (num<0)                          //если переменная num меньше 0, на консоль выводится строка "negative number"
{
cout <<"negative number"<<endl;
}
```

```
else                                    // иначе, если num равно 0,
{                                       выводится строка "neutral
   if (num==0)                          number"
       {
cout<<"neutral number"<<endl;
       }
    else                                // иначе выводится строка
       {                                "positive number"
cout << "positive number" << endl;
       }
}
```

2.2 ОПЕРАТОР МНОЖЕСТВЕННОГО ВЫБОРА SWITCH

Запись операций множественного выбора (например, при значениях переменной 1,2,3 …) с помощью ветвлений выглядит очень громоздко. В этих случаях рациональнее использовать оператор множественного выбора *switch*. Это более эффективный условный оператор ветвления.

Для начала объявляется сам оператор *switch* и после него ставятся круглые скобки, внутри которых записывается выражение или переменная для дальнейшего множественного выбора. В *switch* не может использоваться тип данных *float*.

Далее записывается *case* и символ/число, которые должны провериться на равенство с выражением/переменной, записанной при *switch*. Ставится двоеточие, после чего записывается последовательность действий, которые должна выполнить программа при совпадении проверяемых значений, и оператор *break*. Этот оператор выполняет функцию прекращения работы *switch*, при его выполнении программа переходит на следующую строку после блока *switch*.

Если значение выражения/переменной, объявленной в *switch*, не совпадает ни с одним значением кейсов, то выполняется последовательность действий, записанных в блоке *default*.

Пример:

```
switch (i)                              // оператор будет работать с
{                                       переменной i
```

```
case 1:                          // в случае, если i равно 1, находится инкремент a и выводится
        a++;                     на консоль, работа switch прекращается
        cout << a << endl;
        break;

case 2:                          // в случае, если i равно 2, находится декремент a и выводится
        a--;                     на консоль, работа switch прекращается
        cout << a << endl;
        break;

default:                         // если i не равно ни 1, ни 2, то программа выводит сообщение
        cout << "Error" << endl; об ошибке, действие switch прекращается
        break;
}
```

2.3 ЦИКЛЫ

Иногда необходимо повторять одно и то же действие несколько раз подряд. Для этого используются циклы.

Простейшим и наиболее используемым циклом является *for*, его можно использовать в том случае, если мы знаем точное количество итераций (повторений) цикла. В круглых скобках после ключевого слова *"for"*, через точку с запятой, идет начальное условие, условие продолжения выполнения цикла и описание действия в конце каждой итерации цикла. Затем ставятся фигурные скобки и внутри них пишутся инструкции для цикла.

Цикл *while* – цикл с предусловием. Начинается с написания ключевого слова *"while"*, после которого следуют круглые скобки с условием. После них ставятся фигурные скобки, в которых пишется тело цикла. Этот цикл будет выполнятся до тех пор, пока условие в круглых скобках будет являться истиной.

Цикл *do… while* – цикл с постусловием. *Н*ачинается с *"do"* и фигурных скобок, в которые записываются действия цикла. После чего пишется *"while"* и ставятся круглые скобки, в которых описывается условие продолжения цикла, после скобок обязательно ставится точка с запятой. Особенность этого цикла в том, что одна итерация цикла будет выполнена независимо от условия.

В том случае, если потребуется досрочное прерывание цикла, можно использовать оператор *break*.

Примеры:

```
for (i=1; i<n+1; i++)                    //циклы выведут на консоль все
    {                                    числа от 1 до n
      cout<<i<<endl ;
    }
int i=1;
while (i < n+1)
    {
      cout<<i<<endl;
      i++;
    }
int i=1;
do
    {
      cout << i << endl;
      i++;
    }
while(i < n);
```

Задание для самостоятельного выполнения:

Уравнение: **Ai = A0 + (i - 1) * d**, где A0 – считывается из файла, n,d – вводятся с клавиатуры, n- максимальное количество членов, i - номер члена.

1. Цикл «For»

Найти и вывести на экран и в файл все члены последовательности Ai.

Найти и вывести на экран и в файл сумму всех членов последовательности и среднее арифметическое всех членов последовательности.

2. Цикл «While»

Найти и вывести на экран и в файл все члены последовательности Ai с номером элемента < n.

Найти и вывести на экран и в файл сумму всех членов последовательности и среднее арифметическое всех членов последовательности.

3. Цикл «Do…While»

Найти и вывести на экран и в файл все члены последовательности Ai, сумма которых меньше 100-150 (число выбрать самостоятельно).

Найти и вывести на экран и в файл сумму всех членов последовательности и среднее арифметическое всех членов последовательности.

В том случае, если i<0 – прекратить выполнение подпрограммы и вывести на экран фразу «Выполнение программы невозможно».

ГЛАВА 3. МАССИВЫ

3.1. ЗНАКОМСТВО С МАССИВАМИ

Массив – совокупность переменных одного типа, объединённых под общим именем. Каждая переменная в массиве называется *элементом*.

Массивы могут быть как *одномерными*, так и *многомерными*. Также массивы делятся на *фиксированного размера* и *динамические*. Их отличие заключается в том, что фиксированные массивы имеют фиксированную длину, которую нельзя изменить в процессе выполнения программы. Размер динамических массивов может зависеть от пользовательских значений, либо значений, которые вычисляются при работе программы.

Работа с массивом начинается с *объявления* самого массива: для массивов фиксированного размера сначала записывается тип данных его элементов, а затем имя самого массива. В квадратных скобках после имени записывается размерность (чтобы сообщить, что это не обычная переменная). Объявление динамического массива происходит с помощью *указателя* и оператора *new*, который необходим для выделения памяти.

Для массива, использующего класс *string,* необходимо подключить библиотеку *string.h*.

Важно помнить, что отсчёт элементов массива начинается с *нулевого*. Поэтому, чтобы при обращении, например, к пятому по порядку элементу одномерного массива, в квадратных скобках после имени массива необходимо записать число 4, а не 5.

Примеры:

```
#include <string.h>                    //подключение библиотеки string
string mas[5]={“a”, “b”, “c”, “d”, “e”}; //объявление одномерного массива фиксированного размера, класса string, где a, b, c, d, e – элементы массива
int mas1[5]={1, 2, 3, 4, 5};           //объявление одномерного массива фиксированного размера, где 1, 2, 3, 4, 5 – элементы массива целочисленного типа
int *mas2= new int[n];                 // выделение памяти для динамического массива размерности n
int mas3[5];                           //объявление одномерного массива из 5 элементов
int mas4[5][3];                        //объявление двумерного массива фиксированного размера – 5 строк, 3 столбца
cout<<mas4[4][2];                      //вывод элемента массива пятой строки, третьего столбца
```

3.2. ДЕЙСТВИЯ С МАССИВАМИ

Заполнять/выводить/считывать/записывать массив можно с помощью *цикла*. Используя *циклы*, можно проводить операции сложения, вычитания, умножения и деления элементов массивов. Для работы с двумерными массивами необходимо использовать структуру *цикл в цикле.* Для транспонирования массива (замены строк столбцами) используется цикл с заменой строк столбцами.

Если необходимо провести сравнение элементов, то внутри цикла используется *ветвление*.

Для лучшего восприятия двумерного массива и перехода на новую строку, в первом цикле используется *“\n”* или *endl*, для обеспечения табуляции во втором(вложенном) цикле используется *“\t”*.

Для использования математических функций при работе с массивами, нужно подключить заголовочный файл *math*.

Примеры:

```
#include <math.h>                    //подключение библиотеки math
for (int i = 0; i < 5; i++)          // заполнение и вывод членов дву-
{                                     мерного массива, размерностью 5
  for (int j = 0; j < 3; j++)         строк на 3 столбца, с помощью
    {                                 цикла for
      mas4[i][j] = 10*(i+j) ;
      cout << mas4[i][j] << "\t";
    }
  cout << endl;
}
```

Задание для самостоятельного выполнения:

Дано: В файле записано 2 двумерных массива размерностью 4 на 3.

1. Считать данные из двумерных массивов, вывести все члены массива на экран и в выходной файл.

2. Выполнить почленное сложение, вычитание, умножение и деление 2-х массивов, вывести результаты на экран и в файл.

3. Выполнить почленное сравнение элементов массивов и записать больший элемент в третий массив, также вывести результат на экран и в файл.

4. Произвести транспонирование получившегося в п.3 массива, вывести в четвертый массив размерностью 3 на 4. Вывести результат на экран и в файл.

ГЛАВА 4. СОРТИРОВКИ

4.1 СОРТИРОВКА МЕТОДОМ ВЫБОРА

Суть сортировки заключается в том, что в массиве производится поиск и *выбор* минимального элемента и перемещение его в начало массива.

Сортировка выполняется с помощью *двойного цикла* и оператора ветвления *if*. *Первый* цикл производит перебор всех членов массива, *второй* – сравнение этого члена с каждым последующим. *Оператор ветвления* производит перестановку сравниваемых эле-

ментов, если найден минимальный, если такой элемент не найден, то все члены массива остаются на своих местах.

Пример:

```
const int n=5;                          //ввод необходимых данных для
int a[n] = { 1, 25, 9, 36, 49 };        операции с массивом, где temp –
int temp = 0;                           переменная для временного хране-
int i, j;                               ния данных
for (i = 0; i < n; i++)                 //реализация сортировки массива
{                                       по возрастанию методом выбора
    int min = i;
    for (j = i + 1; j < n; j++)
    {
        if (a[j]<a[min])
        {
        min=j;
        temp = a[i];
        a[i] = a[min];
        a[min] = temp;
        }
    }
}
for (int i = 0; i < n; i++)             //вывод отсортированного масси-
{                                       ва
    cout << a[i] << '\t';
}
```

4.2 СОРТИРОВКА ВСТАВКАМИ

Заключается в том, что берется один из элементов массива и находится позиция для его *вставки*.

Он реализуется так же при помощи *двойного цикла*. С помощью *первого* цикла аналогично производится перебор всех элементов массива и запись их во временную переменную *temp*, *второй* цикл сравнивает эту переменную с остальными членами массива. Внутри первого цикла после одной итерации второго последнему замененному члену массива присваивается значение переменной *temp*. Дополнительные условия для цикла можно записать через знак *“&&”*(логическое «И»).

Пример:

```
for (i = 1; i < n; i++)                    //реализация сортировки
{                                          вставками для одномер-
   temp = a[i];                            ного массива размерно-
   for (j = i - 1; j >= 0 && a[j] > temp; j--)   стью n
        {
                a[j + 1] = a[j];
        }
   a[j + 1] = temp;
}
```

4.3 СОРТИРОВКА ПУЗЫРЬКОМ

Суть метода заключается в том, что берется элемент массива и сравнивается со всеми предшествующими элементами. Если обнаруживается, что взятый элемент меньше того, с которым сравнивается, они *меняются местами*. Такой проверке подвергается каждый элемент массива.

Все выполняется также с помощью *двойного цикла*: *первый* цикл производит перебор всех членов массива, *второй* осуществляет сравнение каждого члена со всеми остальными.

Если необходимо найти какой-либо элемент в массиве, можно воспользоваться «*линейным*» поиском: сравнением всех элементов массива с необходимым значением(ключом). Этот процесс реализуется с помощью цикла, который выполняет перебор всех членов массива и сравнивает их с ключом. Данный метод абсолютно непригоден для больших массивов, однако, имеет преимущество: не требуется предварительная сортировка массива.

Пример:

```
for (i = 0; i < n; i++)                    //реализация сортировки мето-
{                                          дом пузырька для одномерного
   for (j=i; j>0 && a[j-1]>a[j]; j--)      массива размерностью n
        {
                temp=a[j];
                a[j]=a[j-1];
                a[j-1]=temp;
        }
}
```

```
int key=5;                          //поиск необходимого значения в
for (i=0; i<n;)                     массиве и вывод его на консоль
{
        in>>a[i];
        if (a[i]==key)
        {
                cout<<a[i];
                i++;
        }
        else {i++;}
}
```

Задание для самостоятельного выполнения:

Дано: В файле записано 25 чисел, из которых будет формироваться один одномерный динамический массив размерностью *до* 25 членов.

1. Необходимо предусмотреть возможность задания из консоли размерности массива: n≤25.

2. Необходимо считать данные из одномерного массива размерности n, вывести все его члены на экран и в выходной файл.

3. Необходимо провести сортировку массива по возрастанию и вывести результат на экран и в файл.

4. Необходимо найти, какое место в несортированном массиве занимает определенное число: Вводится переменная key (ключ), значение которой задается с клавиатуры – то самое определенное число, которое нужно найти в массиве. Если требуемого числа нет в массиве, необходимо вывести на экран сообщение: «Число в массиве не найдено». Вывод на экран и в файл.

ГЛАВА 5. ФУНКЦИИ. ТИПЫ ФУНКЦИЙ.

5.1. ФУНКЦИИ

Функции – блоки кода, выполняющие определенные операции. Они могут использоваться неограниченное число раз для выполнения одних и тех же действий в разных местах программы. Таким образом, программа становится наиболее читаемой.

Объявление функций начинается *после блока переменных*. *Нельзя* объявлять функции *в теле другой функции* (т.е. сделать так,

чтобы функция была вложенной). Сначала определяется тип функции, её имя, а затем, в круглых скобках, записываются параметры, от которых зависит функция. После чего ставятся фигурные скобки, и внутри них уже пишется необходимая последовательность действий, которая должна выполняться при вызове этой функции (т.е. её тело).

Для выполнения функции её необходимо вызвать: записать её имя и параметры, поставить точку с запятой.

Примеры:

```
void name1()                    //объявление функции name1 без
{                               параметров, от которых она
        cout << "1" << endl;    зависит, типа void, которая вы-
}                               водит на консоль число 1.
int main()                      // вызов функции name1 в теле
{                               главной функции
        name1 ();
        system("pause");
        return 0;
}
```

5.2. ТИПЫ ФУНКЦИЙ

Функции делятся на два типа: функции *без возврата* и *возвращающие значение*.

Функция *без возврата* должна иметь тип *void* (такие функции иногда называют процедурами). Функции, *возвращающие значение*, должны иметь *такой же тип*, что и *возвращаемое значение*. В теле такой функции необходимо наличие оператора *return*, чтобы указать возвращаемое значение или выражение.

С помощью функций также можно обеспечить выполнение такого процесса, как *рекурсия* – обращение функции к самой себе в ходе выполнения этой функции (например, нахождение факториала, или числа последовательности Фибоначчи).

Примеры:

```
void sum1(int a, int b)         //объявление невозвратной функ-
{ cout << a + b; }              ции sum1
```

```
int main()                          //вызов функции sum2 со значениями a=5 и b=2
{
        sum1(5, 2);
        return 0;
}
int sum2(int a, int b)              //объявление возвратной функции sum2, в которую возвращается значение выражения a+b
{
        return a + b;
}
int main()                          //вывод значения функции sum2 на консоль, где переменные a=5 и b=2
{
        int a = 5;
        int b = 2;
        cout << sum2(a, b);
        return 0;
}
unsigned long int fact (int f)      //пример объявления функции, использующей рекурсию для вычисления факториала числа f. Если f=0 или f=1, значение функции становится равным 1. || - логическое «ИЛИ»
{
        if (f == 0 || f == 1)
             return 1;
        result = f * fact(f - 1);
        return result;
}
int main()                          //ввод числа f и вывод значения функции fact, зависящей от f, в консоль
{
        cin >> f;
        cout << fact(f) << endl;
        system("pause");
        return 0;
}
```

Задание для самостоятельного выполнения:

1. Функция - рекурсия.

Реализовать нахождение n-ного члена последовательности Фибоначчи, используя рекурсию, а также сумму всех членов, включая n-ный. Номер члена n вводится с клавиатуры. Результат вывести на экран и записать в файл.

Вики: Чи́сла Фибона́ччи - элементы числовой последовательности, в которой первые два числа равны либо 1 и 1, либо 0 и 1, а каждое последующее число равно сумме двух предыдущих чисел.

2. Функция - множественный выбор.

Используя оператор множественного выбора switch, реализовать функцию математических операций (+,-,/,*) по выбору с двумя двумерными массивами 2*5, которые считываются из файла.

Дополнительно добавить опцию поиска максимального или минимального элемента одного из массивов по выбору пользователя (max1, min1, max2, min2).

Повторить вызов оператора множественного значения в основной функции программы 3 раза.

Массивы и результаты операций над ними вывести на экран и в файл.

ГЛАВА 6. СТРУКТУРЫ

Структуры – это совокупность данных, объединенных под одним именем. Стоит отметить, что данные, объединенные одной структурой, могут иметь различный тип. Таким образом, пользователь может создать свой тип данных, который будет объединять несколько переменных различных типов вместе.

Определение структуры всегда начинается с записи ключевого слова *struct*, затем записывается имя структуры (имя нового типа данных), далее идут фигурные скобки, в которые записываются её члены. После фигурных скобок *обязательно* ставится точка с запятой. Для записи одного члена структуры нужно сначала указать его тип, затем имя и точку с запятой.

Можно объявлять переменные нового типа данных, а также работать с членами структуры или объявлять массивы структур.

Массивы структур бывают как *фиксированного размера*, так и *динамические*. Массивы структур фиксированного размера можно объявлять после определения структуры, между фигурной скобкой и точкой с запятой. Объявление динамических массивов структур происходит так же, как и объявление динамических массивов: с помощью *указателя* и оператора *new*.

Примеры:

```
struct student                          //объявление новой структуры
{                                       student, состоящей из членов
        string name;                    различных типов – int, string с
        int year;                       именами name, year и mark.
        int mark;                       Стоит помнить, что для рабо-
};                                      ты с типом string, необходимо
                                        подключение заголовочного фай-
                                        ла string.h
student ivanov;                         //объявление новой переменной
ivanov.name = “Ivan”;                   ivanov типа student и присвоение
ivanov.year = 2000;                     значений её элементам
ivanov.mark = 5;
student ivanov = {“Ivan”, 2000, 5};     //альтернативная запись объяв-
                                        ления новой переменной ivanov с
                                        присвоением значений её эле-
                                        ментам
student ivanov={“Ivan”, 2000};          //если в списке инициализаторов
                                        не будет одного или нескольких
                                        элементов, то им присвоятся
                                        значения по умолчанию. В дан-
                                        ном случае ivanov.mark будет
                                        равен 0
struct student                          //объявление нового массива r
{                                       структур student размерностью
        string name;                    5
        int year;
        int mark;
}r[5];
student *s = new student[l];            //объявление динамического мас-
                                        сива структур s размерностью l
for(i = 0; i < l; i++)                          //работа с массивом
{                                               структур: запись l
cin>>s[i].name>>s[i].year>>s[i].mark>>endl;      значений её членов,
        cout << s[i].year;                      вывод каждого значе-
}                                               ния year после записи
```

Задание для самостоятельного выполнения:

Дано: файл, содержащий базу данных 40-ти студентов.

База данных содержит следующие поля:

1. Номер студента (от 1 до 1333);
2. Фамилия студента;
3. Год рождения (от 1998 до 2002);
4. Год обучения (от 1 до 5);
5. Средний балл (от 5 до 10).

Реализовать (с помощью функции, структур, оператора множественного значения switch):

1. Поиск по номеру студента:

1.1 Вывод студента по конкретному номеру N (с клавиатуры);

1.2 Поиск всех студентов с номером свыше N (с клавиатуры);

1.3 Поиск всех студентов с номером ниже N (с клавиатуры);

1.4 Сортировка базы данных по возрастанию номера;

1.5 Сортировка базы данных по убыванию номера.

2. Поиск по фамилии студента:

2.1 Вывод студента по фамилии (с клавиатуры);

3. Поиск по году рождения студента:

3.1 Вывод всех студентов с одним годом рождения N (с клавиатуры);

3.2 Вывод всех студентов с годом рождения свыше N (с клавиатуры);

3.3 Вывод всех студентов с годом рождения ниже N (с клавиатуры).

4. Поиск по году обучения студента:

4.1 Вывод всех студентов с одним годом обучения N (с клавиатуры);

4.2 Вывод всех студентов с годом обучения свыше N (с клавиатуры);

4.3 Вывод всех студентов с годом обучения ниже N (с клавиатуры).

5. Поиск по среднему баллу:

5.1 Вывод всех студентов со средним баллом свыше N (с клавиатуры);

5.2 Вывод всех студентов со средним баллом ниже N (с клавиатуры).

6. Предусмотреть возможность добавления строк в базу данных без перезаписи исходного файла (воспользоваться динамическим массивом структур).

fopen(fileName, "a"); - открытие файла с возможностью дозаписи.

7. Обеспечить полный функционал по пп.1-5 для расширенной базы данных из п.6.

Вывести результаты работы программы на экран и в файл в полном объеме.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Березин Б.И., Березин С.Б.* Начальный курс С и С++/ Начальный курс С и С++ - М.: ДИАЛОГ-МИФИ, 2001. - 288 с. ISBN 5-86404-075-4

2.Уроки программирования на С++ с нуля/ [Электронный ресурс]. URL: https://code-live.ru/tag/cpp-manual/ (дата обращения: 07.10.2020).

3. Уроки программирования на языке С++/ [Электронный ресурс]. URL:https://ravesli.com/uroki-cpp/ (дата обращения: 07.10.2020).

4. Справочник по языку С++/[Электронный ресурс]. URL:https://docs.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view=vs-2019 (дата обращения: 07.10.2020).

5. Основы языка программирования С++/[Электронный ресурс]. URL:http://cppstudio.com/cat/274/275/ (дата обращения: 07.10.2020).

СОДЕРЖАНИЕ

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

Методические указания к лабораторным работам для студентов бакалавриата направления 11.03.04

Сост.: *А.А. Белицкий, А.В.Полукеева*

Печатается с оригинал-макета, подготовленного кафедрой электронных систем

Ответственный за выпуск *А.А. Белицкий*

Лицензия ИД № 06517 от 09.01.2002

Подписано к печати 25.01.2021. Формат 60×84/16.
Усл. печ. л. 1,3. Усл.кр.-отт. 1,3. Уч.-изд.л. 1,0. Тираж 75 экз. Заказ 40.

Санкт-Петербургский горный университет
РИЦ Санкт-Петербургского горного университета
Адрес университета и РИЦ: 199106 Санкт-Петербург, 21-я линия, 2