Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высше
профессионального образования «Томский государственный университет
систем управления и радиоэлектроники»
(ТУСУР)

УТВЕРЖДАЮ
Заведующий кафедрой УИ
_____ А.Ф. Уваров
«____» _____ 2012 г.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ
К ПРАКТИЧЕСКИМ ЗАНЯТИЯМ
по дисциплине

«ОБЪЕКТНО-ОРИЕНТИРОВАННЫЕ МЕТОДЫ АНАЛИЗА, ПРОГРАММИРОВАНИЯ И ПРОЕКТИРОВАНИЯ»

Составлены кафедрой «Управление инновациями»
Для студентов, обучающихся по направлению подготовки 220600 «Инноватика»
Магистерская программа «Мультимедийные многопроцессорные системы на кристалле»

Форма обучения очная

Разработчик:
профессор, д.т.н. _____ М.Ю. Катаев
«____» _____ 2012 г.

Томск 2012 г.

Содержание

Введение

Изучение дисциплины «**Объектно-ориентированное программирование**» имеет существенное значение в специальной подготовке студентов по направлению «Инноватика».

Цель данного пособия состоит в выработке навыков в разработке программных средств и интерпретации программ на языке программирования C++.

Для полноценного понимания и усвоения материала необходимо предварительно изучить дисциплины "Информатика" и "Основы программирования".

Для углубленного изучения и освоения материала целесообразно применение различных форм самопроверки знаний студентов: тесты, задачи, упражнения. Они могут быть использованы при проведении практических занятий в университете, выполнении курсовых, контрольных и аудиторных работ, а также при самостоятельном изучении данных дисциплин.

Одним из наиболее интенсивных способов изучения дисциплины является самостоятельная реализация различных алгоритмов, в виде программного кода. При этом вырабатывается опыт и навыки, необходимые при разработке сложных программных средств.

Предлагаемые задания позволят глубже освоить теоретические и практические вопросы объектно-ориентированной парадигмы программирования, понять принципы написания объектно-ориентированных (основанных на классах) программ, научиться применять эти теоретические знания при разработке программных средств различной сложности.

Практическое занятие №1.

Тема: *Классы. Открытые и закрытые уровни доступа. Конструкторы. Инициализация данных объекта. Определение методов. Создание объекта в памяти. Стандартные потоки ввода-вывода.*

1. В среде программирования на C++ создайте консольный проект с именем LAB1.
2. В проекте создайте файлы *main.h* (заголовочный файл) и *main.cpp* (файл исходного кода).
3. В файле *main.h* определите с помощью ключевого слова ***class*** объект *Person.*

   Данные объекта (название переменных придумайте сами):

   Номер человека (целый тип)

   ФИО (символьный массив)

   Пол (логический тип: 0-муж., 1-жен.)

   Возраст (вещественный тип)

   Пусть данные имеют закрытый уровень доступа (***private***).
4. Опишите конструктор объекта, аргументы которого будут инициализировать все данные объекта.
5. Опишите конструктор объекта по умолчанию (без аргументов) проинициализировав все данные.
6. Опишите в объекте функцию ***void*** *Print()* с открытым уровнем доступа (***public***), которая будет выводить данные на экран. Откройте файл *main.cpp*. С помощью директивы **#include** включите в файл *main.cpp* заголовочные файлы <*stdlib.h*>, <*string.h*>, <*iostream.h*>, а также ваш заголовочный файл "*Main.h*". Зачем мы включаем эти файлы? Разберитесь самостоятельно.
7. Ниже определите конструктор объекта инициализирующий все данные объекта значениями аргументов. В теле конструктора используйте функцию *strcpy(cmp1, cmp2)* для копирования строки имени человека (ФИО).
8. Затем определите функцию ***void*** *Person::Print()*. В теле функции для вывода данных используйте стандартный поток вывода *cout << значение1 << значение2 << ... << endl;* , где *значениеN* - переменная стандартного типа (***int, char, double*** и т.д.)
9. Ниже напишите главную функцию программы ***int*** *main()*. Внутри ее создайте объект *Person*, указав все значения данных объекта. Выведите данные объекта на экран, вызвав функцию *Print*.

10. Затем создайте динамический объект *Person* с помощью обычного конструктора и оператора ***new***. Выведите данные объекта на экран. Удалите динамический объект из памяти, с помощью оператора ***delete***.
11. Напишите функцию ввода данных в объект с клавиатуры ***void*** *Person::Input()*. В теле функции для ввода данных используйте стандартный поток ввода *cin >> значение1 >> значение2 >> … ;*.
12. Затем в теле функции *main* создайте объект *Person* с помощью конструктора по умолчанию и введите данные в объект с клавиатуры, вызвав функцию *Input*. Выведите данные объекта на экран.

Практическое занятие №2

Тема: *Создание динамического массива объектов. Деструктор объекта.*

1. В среде программирования на C++ создайте консольный проект с именем LAB2_3 в каталоге LAB2_3.
2. Переименуйте файл *main.h* из предыдущей лабораторной в *person.h*. Создайте файл *person.cpp* и включите в проект эти два файла. Переместите конструктор и функции объекта *Person* из *main.cpp* в файл *person.cpp*. Таким образом файл *person.h* содержит описание объекта *Person*, а файл *person.cpp* - реализацию объекта *Person*.
3. Включите в проект файл *main.cpp* и очистите тело функции *main()*.
4. Определим объект *Group*, который будет содержать динамический массив объектов *Person*. Создайте два файла *group.h* и *group.cpp* и включите их в проект.
5. В файле *group.h* определите с помощью ключевого слова ***class*** объект *Group*.
   Данные объекта: *размер массива (целый тип)*
   указатель на массив (тип Person)*
   Пусть данные имеют закрытый уровень доступа (***private***).
6. Опишите конструктор объекта с одним аргументом - размер массива (целый тип) и деструктор объекта.
7. Откройте файл *group.cpp*. С помощью директивы ***#include*** включите необходимые заголовочные файлы.
8. Определите конструктор объекта *Group*. В теле конструктора проинициализируйте данные объекта, т.е. проинициализируйте размер массива значением аргумента конструктора и выделите динамическую память под массив с помощью строки кода: *указатель на массив* = ***new*** *Person[размер массива]*.
9. В деструкторе объекта освободите память, занимаемую массивом, с помощью строки кода ***delete*** *[]указатель на массив*. Таким образом мы создали объект G*roup*, который содержит массив объектов *Person.*
10. Определим открытые (***public***) методы для объекта *Group*. Напишите функцию ***void*** *Group::Print()*, которая выводит в цикле ***for*** все записи массива на экран. В теле цикла примените ранее написанную функцию *Print()* для объекта *Person*.
11. Напишите функцию ***int*** *Group::Size()*, которая возвращает размер массива.
12. Напишите функции *void PutPerson(int i, Person& man);* и *Person& GetPerson(int i);*, первая функция заносит объект *man* типа *Person* в i-й

элемент массива, вторая функция возвращает объект типа *Person* из i-го элемента массива

13. Заполните массив данными и, затем, выведите их на экран. Для этого в теле функции ***int*** *main()* сначала определите массив имен, которые будут заносится в поле ФИО объекта *Person*, например, ***char*** *names[5][25] = {"Peter", "Susan", "Michael", "Bob", "Jenny"};*. Затем создайте объект с именем group типа *Group* размером пять записей, т.е. *Group group(5);*.
14. Ниже с помощью цикла ***for*** заполните массив данными. Для этого в теле цикла создайте объект *Person*, проинициализировав все его данные, и с помощью функции *PutPerson* занесите объект в массив.
15. Выведите массив на экран с помощью строки кода: *group.Print();*. Получилось? Если да, то вы научились создавать динамический массив объектов, определять функции работы с таким массивом и выводить его на экран.

Практическое занятие №3

Тема: *Два типа полиморфизма: принудительное приведение типа, перегрузка функций и перегрузка операторов (унарных и бинарных).*

1. В этой части лабораторной работы изучим первые два типа полиморфизма - это: а) принудительное приведение типа; б) перегрузка функций и операторов. Работу будем вести в том же проекте LAB2.
2. Напишите функцию приведения типа. Для этого с помощью ключевого слова ***operator*** напишите функцию объекта *Person*, которая преобразует тип *Person* в ***double***. Пусть функция возвращает возраст человека, например, *Person::operator double() { **return this**->Age; }*. Что означает ключевое слово ***this***? Разберитесь сами.
3. Проверьте функцию преобразования типа. В функции ***int*** *main()* далее определите переменную ***double*** и присвойте ей объект *Person,* например, ***double*** *age = group.GetPerson(2);*. Т.е. совершается неявное преобразование из типа *Person* в тип ***double*** при обращении к объекту. Выведите значение переменной на экран.
4. Перегруженные функции имеют одинаковое название, но разный возвращаемый тип или/и разный список аргументов. Определим в объекте *Group* две функции с одинаковым именем, например, ***double*** *Age();* и ***double*** *Age(**int** limit);*. Первая функция пусть возвращает средний возраст группы людей, а вторая функция пусть возвращает средний возраст людей в группе, возраст которых не больше некоторого граничного значения *limit*. Функции отличаются списком аргументов.
5. Проверьте работу перегруженных функций, отобразив на экране подсчитанные два значения среднего возраста.
6. Перегрузите оператор индексирования. Если раньше, чтобы обратиться к элементу массива, нам необходимо было вызывать функцию *GetPerson,* то, определив оператор индексирования, мы будем использовать только квадратные скобки. Сравните две строки кода:

   Person man = group.GetPerson(2);

   Person man = group[2];

   В объекте *Group* с помощью ключевого слова ***operator*** определите оператор индексирования, например: *Person& Group::operator[](int i)*. В теле оператора напишите код, возвращающий i-тый элемент массива, т.е. объект *Person*
7. Выведите на экран с помощью оператора индексирования любой один элемент массива *group*, например третий.

8. Перегрузим бинарный оператор, например, оператор сложения (+) для объекта *Person*. Пусть оператор сложения будет возвращать суммарный возраст двух человек. Опишем в объекте *Person* данный оператор как дружественную функцию с помощью ключевого слова ***friend***, например:

friend double operator*+(Person& p1, Person& p2);*

Эта строка кода означает, что оператор сложения не принадлежит объекту, но ему доступны все закрытые данные и методы объекта.

В файле *person.cpp* определите оператор сложения, например:

double operator *+ (Person& p1, Person& p2){* ***return*** *(p1.Age + p2.Age); }*

Здесь мы напрямую обращаемся к закрытому полю *Age* объекта *Person.*

9. Проверим работу оператора с помощью следующих строк кода:

double *sum = group[1] + group[3];*

cout << sum << endl;

Определите другие операторы и методы для объектов *Person* и *Group* по Вашему усмотрению. Покажите результаты вашей работы преподавателю

Практическое занятие №4

Тема: *Наследование. Защищенный уровень доступа. Виртуальные функции, как один из видов чистого полиморфизма.*

1. В среде программирования на C++ создайте консольный проект с именем LAB4 в каталоге LAB4. Скопируйте все файлы с расширением *.cpp и *.h из каталога LAB2_3 в каталог LAB4 и включите их в текущий проект.
2. Наследование - это механизм получения нового объекта из базового, при этом новый объект вбирает в себя все свойства базового объекта и дополняется новыми данными и методами. Например в C++ чтобы описать объект-наследник с открытым уровнем доступа используют запись вида:

   class *имя_класса :* ***public*** *имя базового_класса { объявление данных и методов };*

   Создадим новый объект *Student*, который будет наследником от *Person*.

   Для этого создайте файлы *student.cpp* и *student.h* и включите их в текущий проект.
3. В файле *srudent.h* опишите новый объект *Student*, который будет открытым наследником от *Person*. Укажите закрытые (***private***) данные объекта *Student*:

   год обучения (целый тип), коэф. успеваемости (вещественный тип).
4. Опишите конструктор объекта со всеми необходимыми аргументами
5. Откройте файл *student.cpp*. С помощью директивы ***#include*** включите необходимые заголовочные файлы.
6. Определите конструктор объекта *Student* с вызовом конструктора базового класса и инициализацией всех данных объекта.
7. Напишите функцию ***void*** *Student::Print()*, которая выводит данные объекта на экран с помощью стандартного потока вывода *cout*. В теле функции вызовите функцию базового класса *Person::Print()*.
8. В файле *main.cpp* очистите тело функции *main()*. Затем создайте объект *Student* и выведите данные объекта на экран. Получилось? Если да, то Вы научились создавать объект-наследник.
9. В случае использования защищенного уровня доступа (***protected***) к данным объекта может обращаться или только сам объект или его наследники. Поэкспериментируйте с функцией *Print()*. Закомментируйте вызов функции базового класса и напишите код вывода данных базового класса на экран. Должны появится ошибки:"*Cannot access private member declared in class 'Person'*".

10. Теперь измените уровень доступа к данными базового класса с ***private*** на ***protected***. Теперь Вы получили доступ к данным базового класса из класса-наследника. Выведите данные на экран.
11. Виртуальные функции являются первым видом чистого полиморфизма. Типичный случай - это когда базовый класс содержит виртуальную функцию, а классы-наследники имеют свои версии этой функции. Выбор необходимой функции происходит на этапе выполнения на основе информации о типе объекта. Укажите в файле *person.h* перед описанием функции *Print* ключевое слово ***virtual***.
12. Теперь далее в функции *main()* определите указатель на объект *Person* и проинициализируйте его значением созданного ранее объекта *Student* и вызовите функцию *Print()* например:

    ```
    Person* st2 = &st1;          st2->Print();
    ```

13. На экране должны появиться все данные объекта *Student*, хотя указатель st2 имеет тип *Person*. Таким образом на этапе выполнения происходит неявный выбор вызываемой виртуальной функции на основе типа объекта.
14. Теперь удалите ключевое слово ***virtual*** и запустите программу на выполнение. На экране должны появиться только данные объекта *Person*. Т.е. механизм неявного выбора функции на основе типа объекта не работает (чистый полиморфизм отсутствует).

Практическое занятие №5

Тема: *Шаблоны функций.*

Задание. Итерационный алгоритм вычисления с машинной точностью математического выражения $\sqrt{x}\cdot\cos(x)$ оформить в виде **шаблона функций.** Функция должна возвращать значение выражения и передавать через параметры точность его вычисления и количество итераций. Параметром шаблона будет являться тип представления вещественных данных функции.

Написать второй шаблон для функций печати таблицы значений выражения, представленного в виде указанного выше первого шаблона функций (для итерационного алгоритма).

В основной программе продемонстрировать работу с шаблонами, выведя на экран таблицы математического выражения (значение, точность, число итераций) для всех типов вещественных данных (**float, double, long double).**

Описание. Шаблон функций с именем *fun* вычисляет сначала косинус аргумента методом разложения косинуса в ряд Тейлора, а затем – корень по итерационной формуле, после чего возвращает произведение полученных результатов. Оба процесса (итерационный и суммирование ряда) происходят с точностью до машинного нуля, т.е. до тех пор, пока на следующем шаге итераций выражение не меняет своего значения.

Шаблон функций *printtab* выводит таблицу для заданного в параметре шаблона типа данных, вызывая соответствующую функцию *fun* для всех *x* от указанного в параметре функции *printtab* минимального значения до максимального с указанным там же шагом.

Главная функция *main* производит запрос пользователя на минимальное значение аргумента, максимальное, а также на шаг вычислений. Затем она производит вызов функций *printtab* для каждого из трех необходимых типов данных, преобразовывая к нему введенные пользователем значения. Это типы – *float, double* и *long double.*

Результат работы программы. В показанном ниже результате работы (как и во всех результатах работы, представленных далее в отчете) данные, выводимые программой отражены курсивом, а вводимые пользователем – нормальным шрифтом.

Enter xmin: 1

Enter xmax: 2
Enter delimeter: 0.1

Table for the type FLOAT

x	*fun*	*n*	*zero*
1	*0.5403023*	*8*	*2.087676e-09*
1.1	*0.4757356*	*11*	*7.152557e-07*
1.2	*0.396943*	*12*	*9.417534e-06*
1.3	*0.3049956*	*12*	*4.196167e-05*
1.4	*0.2011077*	*12*	*0.0001174212*
1.5	*0.08663481*	*12*	*0.0002551079*
1.6	*-0.03693496*	*13*	*0.0004734993*
1.7	*-0.1679929*	*14*	*2.384186e-07*
1.8	*-0.3048239*	*14*	*5.960464e-07*
1.9	*-0.4456242*	*14*	*1.072884e-06*
2	*-0.5885208*	*14*	*2.145767e-06*

Листинг программы. Далее представлен листинг самой программы, написанной на СИ++.

```
#include <iostream.h>
#include <conio.h>
#include <typeinfo>

template <class type>
type fun(type x,type &zero,int &count)
 {
 type cos=1,v=1.0,oldcos,sqrt=1,oldsqrt=1,v2;
 int n=2;
 do
  {
  oldcos=cos;
  v*=x*x/(n*(n-1));
  if(n%4==0) cos+=v;
  else cos-=v;
  n+=2;
  }
 while (cos!=oldcos);
 n=n/2;
```

```
do
  {
  v2=oldsqrt-sqrt;
  if (v2<0) v2=-v2;
  oldsqrt=sqrt;
  sqrt=0.5*(oldsqrt+x/oldsqrt);
  n++;
  }
while (sqrt!=oldsqrt && n<=10000);

count=n;
zero=v>v2?v:v2;
return sqrt*cos;
}
...
```

Практическое занятие №6

Тема: *Шаблоны классов.*

Задание. Итерационный алгоритм вычисления математического выражения из работы №5 оформить в виде **шаблонов классов**, объекты которых сохраняют значения: аргумента, соответствующего аргументу значения, выражения, точности его вычисления, количества итераций.

Написать **шаблон функций** (с параметром шаблона, определяющим тип класса из шаблона классов) для печати с пояснительным текстом данных конкретного объекта.

Продемонстрировать работу с шаблонами для всех допустимых вариантов параметров шаблонов.

Описание. Данная программа содержит шаблон классов *Fun*, которые, фактически, инкапсулируют вызов функции *fun*, вычисляющей математическое выражение. Сама функция осталась фактически такой же, что и в лабораторной работе №5, но теперь она является методом класса и не возвращает значение выражения, а лишь изменяет соответствующий ему компонент класса. Вызов этой функции происходит неявно всякий раз, когда вызывается метод изменения аргумента. Непосредственное обращение к данным класса невозможно – разрешено только получать их значения с помощью соответствующих методов, а менять их может только функция *fun* (за исключением значения аргумента *x*).

Шаблон функций печати *printtab* для простоты остался почти что тем же, что и в первой работе, но теперь в нем происходит постоянное обращение к объекту созданного там класса.

Функция *main* осталась абсолютно без изменений.

Результат работы программы. Результат работы программы абсолютно идентичен результату работы программы из лабораторной работы №5.

Листинг программы. Далее представлен листинг самой программы, написанной на СИ++.

```
#include <iostream.h>
#include <conio.h>
#include <typeinfo>
```

```
template <class type>
class Fun
 {
 type expr,zero,x;
 unsigned n;
 void fun();
 public:
   Fun()
    { expr=0; x=0; zero=0; n=1; }
   type GetExpr() { return expr; }
   type GetZero() { return zero; }
   type GetX() { return x; }
   void SetX(type newx) { x=newx; fun(); }
   int GetN() { return n; }
 };

template <class type>
void Fun <type>::fun()
 {
 type cos=1,v=1.0,oldcos,sqrt=1,oldsqrt=1,v2;
 n=2;
 do
  {
  oldcos=cos;
  v*=x*x/(n*(n-1));
  if(n%4==0) cos+=v;
  else cos-=v;
  n+=2;
  }
 while (cos!=oldcos);
 n=n/2;
 do
  {
  v2=sqrt-oldsqrt;
  if (v2<0) v2=-v2;
  oldsqrt=sqrt;
  sqrt=0.5*(oldsqrt+x/oldsqrt);
  n++;
```

```
  }
while (sqrt!=oldsqrt);

zero=v>v2?v:v2;
expr=sqrt*cos;
}
```

Практическое занятие №7

Тема*:* *Шаблоны классов с контейнерами в качестве компонентов.*

Задание. Написать **шаблон класса**, включающего в качестве компонента вектор (**контейнер)** со значениями приближений, получаемых при итерационном вычислении математического выражения из задания №5.

В основной программе создать объекты разных типов и распечатать (вывести на экран) информацию об объектах (размерность вектора, аргумент, значение функции, достигнутая точность, элементы вектора).

Описание. Данная программа содержит тот же шаблон классов *Fun*, что и в лабораторной работе №6, но с добавленными в него новыми компонентами – вектора и итератора, а также набором методов для извлечения из контейнера данных (без возможности их изменения – этим занимается только функция *fun*). Функция *fun* почти такая же, как и во второй лабораторной, но теперь она еще заносит получаемые приближения в вектор *Pribl.*

Шаблон функций *printtab* почти идентичен соответствующему шаблону из лабораторной работы №6, но в нем добавлена процедура вывода содержимого вектора (в цикле).

Функция *main* осталась абсолютно без изменений.

Результат работы программы. Ниже представлен результат работы программы.

Enter xmin: 1
Enter xmax: 1
Enter delimeter: 1

Table for the type FLOAT

x	*fun*	*n*	*zero*
1	*0.5403023*	*8*	*2.087676e-09*

Iterations Vector Elements:

1 0.5 0.5416667 0.5402778 0.5403026 0.5403023 0.5403022 1

Table for the type DOUBLE

x	*fun*	*n*	*zero*

1 0.54030230586814 12 4.11031762331216e-19

Iterations Vector Elements:

1 0.5 0.541666666666667 0.540277777777778 0.540302579365079
0.540302303791887 0.540302305879563 0.540302305868092
0.54030230586814 0.54030230586813 0.54030230586812 1

Table for the type LONG DOUBLE

x	*fun*	*n*	*zero*
1	*0.5403023058681397*	*13*	*8.896791392450573e-22*

Iterations Vector Elements:

1 0.5 0.5416666666666667 0.5402777777777778
0.5403025793650794 0.5403023037918871 0.5403023058795628
0.5403023058680921 0.5403023058681399 0.5403023058681397
0.5403023058681396 0.5403023058681395 1

Листинг программы.

```
#include <iostream.h>
#include <conio.h>
#include <typeinfo>
#include <vector>

template <class type>
class Fun
 {
 type expr,zero,x;
 unsigned n;
 void fun();
 vector <type> Pribl;
 vector <type>::iterator ip;
 public:
  Fun()
   { expr=0; x=0; zero=0; n=1; Pribl.insert(Pribl.end(),0); }
  type GetExpr() { return expr; }
```

```
  type GetZero() { return zero; }
  type GetX() { return x; }
  type GetPribl(unsigned index)
   {
   ip=Pribl.begin();
   if (index+1>Pribl.size()) return 0;
   return ip[index];
   }
  int GetPriblSize() { return Pribl.size();}
  void SetX(type newx) { x=newx; fun(); }
  int GetN() { return n; }
 };

. . .
}
```

Практическое занятие №8

Тема: *Шаблоны классов и последовательные контейнеры.*

Задание. Компонентами класса являются: значения аргумента *x,* значение функции *y,* точность *zero* (точность вычисления *y* по значению *x*). В основной программе формируется и заполняется контейнер для заданных: минимального x_{min}, максимального x_{max} значений аргумента и шага Δx его изменения. Тип контейнера, заданный преподавателем – вектор.

Упорядочить объекты в контейнере по значениям функции или по точности *zero* с помощью функций стандартной библиотеки.

Напечатать (вывести на экран) содержимое контейнера до упорядочивания и после.

Описание. В данной программе используется шаблон классов *Fun*, полностью идентичный соответствующему шаблону из лабораторной работы №6.

Шаблон функций *printtab* из той же работы модифицирован следующим образом: в нем используется объявление контейнера-вектора, в который заносятся все классы для всех необходимых значений аргумента (при этом тут же неявно вычисляется внутри класса и значение функции). Затем с помощью аналогичной таблицы производится вывод всех объектов класса из контейнера с помощью итератора. Далее производится сортировка объектов в контейнере с помощью стандартной функции *sort*, в которую в качестве параметра задается и соответствующая нужному типу функция *FunCompare*, которая осуществляет сравнение двух объектов соответствующего класса *Fun* (т.е. осуществляет операцию <). Для этого в ней просто сравниваются соответствующие значения функции, чтобы сортировка производилась именно по ним.

Функция *main* осталась точно такой же, как и во всех предыдущих работах.

Результат работы программы. Ниже представлен результат работы программы.

Enter xmin: 1
Enter xmax: 2
Enter delimeter: 0.2

Table for the type FLOAT

x	*fun*	*n*	*zero*
Before Sorting:			
1	*0.5403023*	*8*	*2.087676e-09*
1.2	*0.396943*	*12*	*9.417534e-06*
1.4	*0.2011077*	*12*	*0.0001174212*
1.6	*-0.03693496*	*13*	*0.0004734993*
1.8	*-0.3048239*	*14*	*5.960464e-07*
2	*-0.5885208*	*14*	*2.145767e-06*
After Sorting:			
2	*-0.5885208*	*14*	*2.145767e-06*
1.8	*-0.3048239*	*14*	*5.960464e-07*
1.6	*-0.03693496*	*13*	*0.0004734993*
1.4	*0.2011077*	*12*	*0.0001174212*
1.2	*0.396943*	*12*	*9.417534e-06*
1	*0.5403023*	*8*	*2.087676e-09*

Table for the type DOUBLE

x	*fun*	*n*	*zero*
Before Sorting:			
1	*0.54030230586814*	*12*	*4.11031762331216e-19*
1.2	*0.396943032027585*	*16*	*4.0591974226345e-11*
1.4	*0.201107835580664*	*17*	*5.82139136895421e-09*
1.6	*-0.0369347988243814*	*19*	*3.10862446895044e-15*
1.8	*-0.304823597018452*	*19*	*1.13020703906841e-13*
2	*-0.588520500183628*	*19*	*1.5949463971765e-12*
After Sorting:			
2	*-0.588520500183628*	*19*	*1.5949463971765e-12*
1.8	*-0.304823597018452*	*19*	*1.13020703906841e-13*
1.6	*-0.0369347988243814*	*19*	*3.10862446895044e-15*
1.4	*0.201107835580664*	*17*	*5.82139136895421e-09*
1.2	*0.396943032027585*	*16*	*4.0591974226345e-11*
1	*0.54030230586814*	*12*	*4.11031762331216e-19*

Table for the type LONG DOUBLE

x	*fun*	*n*	*zero*
Before Sorting:			
1	*0.5403023058681397*	*13*	*8.896791392450573e-22*
1.2	*0.3969430320275854*	*18*	*4.059195265072174e-11*
1.4	*0.2011078355806638*	*19*	*1.431146867680866e-17*
1.6	*-0.03693479882438149*	*20*	*3.103637138957005e-15*
1.8	*-0.3048235970184521*	*21*	*1.130986580430426e-13*
2	*-0.5885205001836283*	*21*	*1.594861829407046e-12*
After Sorting:			
2	*-0.5885205001836283*	*21*	*1.594861829407046e-12*
1.8	*-0.3048235970184521*	*21*	*1.130986580430426e-13*
1.6	*-0.03693479882438149*	*20*	*3.103637138957005e-15*
1.4	*0.2011078355806638*	*19*	*1.431146867680866e-17*
1.2	*0.3969430320275854*	*18*	*4.059195265072174e-11*
1	*0.5403023058681397*	*13*	*8.896791392450573e-22*

Листинг программы. Далее представлен листинг самой программы, написанной на СИ++.

```
#include <iostream.h>
#include <conio.h>
#include <typeinfo>
#include <vector>
#include <algorithm>

template <class type>
class Fun
 {
 type expr,zero,x;
 unsigned n;
 void fun();
 public:
  Fun()
   { expr=0; x=0; zero=0; n=1; }
  type GetExpr() { return expr; }
  type GetZero() { return zero; }
```

```
    type GetX() { return x; }
    void SetX(type newx) { x=newx; fun(); }
    int GetN() { return n; }
  };

template <class type>
bool FunCompare(Fun <type> a,Fun <type> b)
 {
 if (a.GetExpr()<b.GetExpr()) return true;
 else return false;
 }


. . .
 }
```

Библиографический список

1. Катаев М.Ю. Конспект лекций по дисциплине «Объектно Ориентированное Программирование».
2. Подбельский В.В. Язык C++: Учебное пособие. – М.: Финансы и статистика, 1995. – 560 с.
3. Страуструп Б. Программирование. Принципы и практика использования C++. – М.: Вильямс, 2011. – 1246 с.