Лист утверждения к методическим указаниям

Форма
Ф СО ПГУ
7.18.1/05

УТВЕРЖДАЮ
Декан факультета ФМиИТ
_____ С. К. Аяжкенов
« » 2009

Составитель: к.т.н., доцент Титов М.В.

Кафедра Вычислительная техника и программирование

Методические указания

к лабораторным занятиям

по дисциплине «Объектно-ориентированное программирование»

для студентов специальности 050704 " Вычислительная техника и программное обеспечение"

Рекомендовано на заседании кафедры

«1» 09 2009 г., протокол № 1

Заведующий кафедрой _____ Потапенко О.Г.

Одобрено МС факультета ФМиИТ

«1» 09 2009г., протокол № 1

Председатель МС _____ А. Т. Кишубаева

Введение

Объектно-ориентированное программирование (ООП) основная методология программирования 90-х годов. Она является продуктом 25 летней практики и включает ряд языков: Simula 67, Smalltalk, Lisp, Clu, Actor, Eiffel, Objective C, C++. Это стиль программирования, который фиксирует поведение реального мира таким способом, при котором детали его реализации скрыты.

Почему ООП так популярно:

- надежда, что ООП может просто и быстро привести к росту продуктивности и улучшению надежности программ, помогая тем самым разрешить кризис в программном обеспечении;
- желание перейти от существующих языков программирования к новой технологии;
- вдохновляющее сходство с идеями, родившимися в других областях.

ООП является лишь последним звеном в длиной цепи решений, которые были предложены для разрешения "кризиса программного обеспечения". Кризис программного обеспечения означает, что те задачи, которые мы хотим решить, опережают наши возможности.

Несмотря на то, что ООП действительно помогает в проектировании сложных и больших программ, ООП не панацея. Чтобы стать профессионалом в программировании, необходим талант, способность к творчеству, интеллект, знания, логика, умение строить и использовать абстракции и опыт, даже, если используются лучшие средства разработки. ООП является новым понимание того, что называется вычислениями. Чтобы стать профессионалом в ООП, недостаточно просто добавить новые знания, необходимо полная переоценка привычных методов разработки программ.

Цель лабораторных работ:

- закрепление полученных теоретических знаний;
- освоение навыков создания программного продукта с использованием объектно-ориентированного программирования.

Лабораторные работы предназначены для студентов очного и заочного отделения по дисциплине 370140 «Вычислительная техника и программное обеспечение» на базе общего среднего, среднего профессионального, высшего профессионального образования.

Студенты представляют описание выполненной работы и дискету с выполненным заданием.

На специальной наклейке гибкого диска (дискеты) необходимо указать свою фамилию и номер варианта, во избежание потери и путаницы при проверке.

Задание должно быть выполнено аккуратно и написано разборчиво. По всем вопросам обращаться за письменной и устной консультацией на кафедру «ВТиП» ПГУ им С. Торайгырова. Результатами лабораторных работ является пояснительная записка, оформленная в соответствии с требованиями ГОСТ, стандартов Университета и задания для лабораторных работ.

Организация выполнения лабораторных работ

Для выполнения лабораторных работ необходимо:

1. Для выполнения контрольных работ необходимо: «Информатики», «Языки программирования», «Технологии программирования».
2. Знание языков программирования;
3. Знать основы проектирования программных средств;

Пояснительная записка к лабораторным работам должна содержать следующие разделы:

1. Введение;
2. Документ описания требований к программе;
3. программа или задача;
4. Заключение;
5. Используемая литература.

Содержание практических занятий

Лабораторная работа №1 –Потоки. Использование стандартного ввода-вывода.

Тема 2. Принципы объектно-ориентированного представления программных систем

Цель работы: Изучения основных навыков объектно-ориентированного программирования.

Задание к лабораторной работе

1. Написать программу вычисления ближайшего сверху числа степени 2; Программа должна использовать цикл while. Входные данные поступают с клавиатуры. Результат выводится на экран. Предусмотреть обработку ошибок.
2. Написать программу, печатающую цифры. Использовать цикл while. В строку выводится цифра, в десятичной, шестнадцатеричной и восьмеричной системе

Лабораторная работа №2 –Файлы-Потоки. Использование стандартного ввода-вывода.

Тема 2. Принципы объектно-ориентированного представления программных систем.

Цель работы: Изучения основных навыков объектно-ориентированного программирования.

Задание к лабораторной работе

1. Написать программу копирования файлов. Чтение происходит блоками. Обработать ошибки.
2. Написать программу вычисления количества символов пробела в файле. Обработать ошибки.

Лабораторная работа №3 –Протокол класса. Конструкторы и деструкторы.

Тема4 Классы.

Цель работы: Изучения основных навыков объектно-ориентированного программирования.

1 Теоретические сведения

Класс является основным элементом языка СИ, обеспечивающим ООП. Он представляет собой расширение структур языка СИ. В классах помимо определение данных допускаются определения выполняемых над ними функций. Кроме того, в них имеются средства управления доступа к данным, позволяющие организовать взаимодействие различных классов друг с другом и классов с обычными функциями. Функции, входящие в класс, часто называются методами.

Определение классов иллюстрирует следующий пример:

```
// iarray.h
 class iarray
{ int maxitems          // максимальное число элементов
  int citems            // число элементов в массиве
  int *items            // указатель на массив
public:
                        // конструктор
array(int nitems);
                        // деструктор
 _array();
                        // занесение элементов в массив
int putitem(int item);
                        // получение элемента из массива
int getitem(int ind; int &item);
                        //получение фактического числа элементов в массиве
int count({return citems;});
};
```

В определении даны объявления внутренних переменных класса (переменные состояния), а также прототипы методов, обеспечивающих работу с данными.

Приведенный класс предназначен для создания целых массивов заданной длины и для размещения в них целых значений. Назначение переменных пояснено соответствующими комментариями.

Класс iarray содержит в себе пять методов. Прототипы четырех из них даны в файле iarray.h, а для пятого метода дана полная реализация.

Сами исходные тексты методов класса iarray приведены ниже:

```
// iarray.cop
# include "iarray.h"
                              //конструктор
iarray::iarray(int nitems)
{ items=new int[nitems];
maxitems=nitems;
citems=0;
}
                              //деструктор
iarray::_iarray()
{delete items;
}
                              //занесение элемента в массив
int iarray::putitem(int item)
{ if (citems<maxitems){
 items[citems]=item;
citems++;
return 0;
}
else return –1;
}
                                    //получение элемента из массива
int iarray::getitem(int ind, int &item)
{ if (ind>=0 && ind<citems){
 item-items[ind];
return 0;
}
else return –1
}
```

Среди методов особое место занимают такие два метода, как конструктор и деструктор. С помощью конструктора, в данном случае, создается массив целых значений заданной длины. В нем фиксируется длина массива и устанавливается счетчик занятых элементов, а также производится распределение памяти под массив. Последняя операция выполняется введенным в C++ оператором new. Он проще, чем функция malloc, так как при ее применении не требуется задавать размер выполняемой памяти в байтах и преобразовывать полученный указатель к конкретному тилу.

Второй специальный метод- деструктор. С его помощью уничтожается созданный конструктором объект. Для уничтожения динамически созданных объектов в C++ используется оператор delete, выполняющий роль функции tree в СИ.

Методы putitem и getitem обеспечивают контролируемое обращение к элементам массива. В первом методе отслеживается возможность размещения нового элемента в массиве. Во втором - правильность задания индекса получаемого из массива элемента.

Метод count позволяет получить число включений в массив элементов. Он реализован прямо в исходном тексте заголовочного файла iarray.h.

Отметим, что при определении методов за пределами текста класса используется операция" ::", например: int iarray::putitem()

Эта операция говорит о том, что областью действия метода putitem является класс irarray.

Коснемся теперь использования определенного нами класса. Можно привести следующий пример:

```
#include "iarray.h"
#include<iostream.h>
#include<stdlib.h>
#include<time.h>
int main()
{
iarray m(10);
int i,j,k;
int n;
randomize()
for (i=1;i<=s;i++)
{
m.putitem(rand());
}
n=m.count()
for(i=0;i<=n;i++){
m.getitem(i,j);
count<<j<<"\n";
}
}
```

В этом примере создается объект m, принадлежащий классу iarray. Далее в массив m вводятся пять целых чисел, полученных путем обращения к датчику случайных чисел. Эти числа вводятся в выходной поток count, которому соответствует дисплей.

В данном примере мы сталкиваемся с понятием объект. Объектом является переменная m, объявленная как принадлежащая классу iarray. Объект- это конкретный экземпляр массива целых чисел.

Действия с объектами осуществляются с помощью методов, обращение к которым задается составными именами, например : m.putitem() или m.getitem();

Отметим, что в объявлении iarray m(10); также производит вызов конструктора, в котором собственно и выполняются действия по созданию массива.

Есть другая возможность создания объекта с помощью оператора new. Например, в следующем фрагменте программы:

```
iarray *a;
iarray *b;
int n1=10
int n2=20;
iarray(n1);
b=new iarray(n2);
.....
delete a;
delete b;
```

, где динамически создаются два массива a и b, размерностью 10 и 20 соответсвенно.

2 Задание к лабораторной работе

Создать класс типа - время с полями: час (0-23), минуты (0-59), секунды (0-59). Класс имеет конструктор. Функции-члены установки времени,. функции-члены получения часа, минуты и секунды, а также две функции-члены печати: печать по шаблону: "16 часов 18 минут 3 секунды" и "4 p.m. 18 минут 3 секунды".

Функции-члены установки полей класса должны проверять корректность задаваемых параметров.

Лабораторная работа №4 –Преобразование типов. Дружественные функции.

Тема4 Классы.

Цель работы: Изучения основных навыков объектно-ориентированного программирования.

1 Теоретические сведения

Класс может предоставлять особые привилегии определенным внешним функциям или функциям-членам другого класса. Эти функции получили названия дружественных. Если функция или класс объявлены как дружественные данному классу, то такие функции или функции-члены такого класса могут осуществлять непосредственный доступ ко всем полям класса, для которого они дружественны. Дружественные функции и классы могут осуществлять прямой доступ к закрытым полям класса без использования функций-членов этого класса.

Ключевое слово **friend** - спецификатор функции, который дает функции-не члену класса доступ к скрытым членам класса. Он используется для того, чтобы выйти за строгие рамки типизации и сокрытия данных в C++.

Одна из причин их использования состоит втом, что некоторые функции нуждаются в привилегированном доступе более, чем к одному классу. Вторая причина - **friend**-функция передаст все параметры через список параметров, и значение каждого из них подчинено преобразованию, совместимому с назначением. Такие преобразования применяются к явно переданным

аргументам-классам и поэтому особенно полезны в случаях перегрузки оператора.

Объявление **friend** функции должно появляться внутри объявления класса, которому она дружественна. Имени функции предшествует ключевое слово **friend**, и ее объявление может находится как в **public** так и в **private** части класса, что не повлияет на значение. Функция-член одного класса может быть **friend**-функцией другого класса. Это происходит тогда, когда функция-член объявлена в **friend** классе с использованием оператора разрешения контекста для определения имени функции дружественного класса. Если все функции-члены одного класса являются **friend**-функциями другого класса, то это можно определить записью:

freind class *имя класса*

```
class t1 {
friend void a(); // friend-функция
int b(); // функция-член
};
class t2 {
friend int t1::b(); // функция-член класса t1 имеет доступ ко всем скрытым полям класса t2
};
class t3 {
friend class t1; // все функции-члены класса t1 имеют доступ ко всем полям класса t3
};
```

Рассмотрим класс **matrix** и класс **vector**. Функция умножения вектора на матрицу должна иметь доступ к **private-членам** обоих классов. Эта функция будет **friend** для обоих классов.

```
class matix;
class vect {
int *p;
int size;
friend vect mpy(const vect &,const matix &);
public:
};
class matrix {
int **base;
int row,column;
friend vect mpy(constr vect&,const matirx &);
};
vect mpy(const vect &v,const matrix &m) {
if(v.size!=m.row) { exit(1); }
vect ans(column);
//.....
return ans;
}
```

Второстепенное значение требует предварительного описания класса **matrix**. Оно необходимо потому, что функция **mpy** должна появляться в обоих классах, и использует каждый класс как тип аргумента.

Friend-функции можно рассматривать как часть общего интерфейса класса.

Существует ряд ситуаций, в которых они могут быть альтернативой функциям-членам. Использование **freind**-функций спорно, потому что они нарушают инкапсуляцию, окружающую **private** члены классов. Парадигма ООП утверждает, что объекты (в C++ они - переменные класса) доступны через их **public** члены. Только функции-члены должны иметь доступ к скрытой реализации АТД. Это ясный и строгий принцип проектирования. **Friend**-функция находится на самой его границе, поскольку имеет доступ к **private** членам, сама не являясь функцией-членом. С ее помощью можно организовать быстрый код для доступа к подробностям реализации класса.

2. Задание к лабораторной работе

Создать класс вещественных чисел. Класс имеет конструктор по умолчанию, конструктор - преобразующий float в объект класса.

Определить оператор преобразования объекта типа вещественных чисел в число типа float.

Создать класс целых чисел. Определить взаимное преобразование с классом вещественных чисел.

Порядок оценки результатов работы

Выполненные лабораторных работ оцениваются по следующим показателям:

- содержание пояснительной записки;
- способность исполнителя отвечать на вопросы по содержанию пояснительной записки и по сути работы.

Текст пояснительной записки должен удовлетворять требованиям ГОСТ и стандартов Университета. Содержание пояснительной записки должно удовлетворять требованиям настоящего задания на выполнение лабораторных работ.

Способность исполнителя практических работ отвечать на вопросы по содержанию пояснительной записки и по сути работы проверяется в личной беседе с преподавателем при защите лабораторных работ.

Список литературы

Основная литература

1. Фридман А.Л. Основы объектно-ориентированной разработки программных систем -М.: Финансы и статистика, 2000.

2. Катаев М.Ю. Объектно-ориентированное программирование: Учебное пособие. - Томск: Томский межвузовский центр дистанционного образования, 2000.

Дополнительная литература

3. Скляров В.А. Язык С++ и объектно-ориентированное программирование. Мн.: Выш. шк., 1997.

4. Ирэ П. Объектно-ориентированное программирование с использованием С++: Пер. с англ. – Киев: НИПФ «ДиаСофт Лтд», 1995.