Пермский государственный технический университет
Кафедра ИТАС
дисциплина «Объектно-ориентированное программирование»
специальность ПОВТ

Методические указания к курсовой работе

Цель курсовой работы

Целью курсовой работы является закрепление и углубление знаний, полученных при изучении дисциплины в ходе лекционных и практических занятий, получение практических навыков создания программ с использованием объектно-ориентированной технологии в современных средах проектирования. Выполнение курсовой работы предусматривает:

1. Создание классов, методов класса, конструкторов и деструкторов, объектов класса.
2. Создание иерархии классов путем наследования.
3. Использование виртуальных функций, абстрактных классов и полиморфных объектов.
4. Использование классов-контейнеров для хранения объектов
5. Использование стандартной библиотеки C++.
6. Использование потоковых классов стандартной библиотеки C++ для сохранения объектов в файле их загрузки из файла
7. Использование стандартной библиотеки шаблонов C++.
8. Получение практических навыков программирования алгоритмов обработки данных.

Содержание курсовой работы

Используя технологию объектно-ориентированного программирования разработать программу для создания объектов пользовательского класса, сохранения их в контейнере и в файле с возможностью последующего восстановления. Использовать алгоритмы сортировки и поиска для работы с данными, хранящимися в контейнере. Программа разрабатывается на языке C++ в среде Microsoft Visual Studio как консольное приложение.

Методические указания к выполнению курсовой работы

Курсовая работа выполняется поэтапно.

I этап. Определение классов. Программирование меню.

1. В соответствии с вариантом (смотри приложение 1) написать программный код для иерархии классов. Определить поля и методы классов. Во главе иерархии должен стоять абстрактный класс. На первом этапе для классов создается только конструктор с параметрами. При определении полей следует учитывать возможность выполнения запроса (смотри приложение 3).

Полиморфизм реализуется с помощью виртуальных функций (например, функцию show() для показа экземпляра класса).

Пример иерархии классов

Файл person.h

```
#include<string>              //для класса string
#include <iostream>          // для потокового ввода-вывода
#include <windows.h>         // для смены кодовой страницы
#include<vector>             //для контейнера
using namespace std;         //использовать пространство имен std
//абстрактный класс
class person{
protected:
    string name;
    int age;
public:
    person(string Name,int Age);
    virtual void show()=0;
};
//класс «рабочий»
class worker:public person{
protected:
    int rang; //разряд
public:
    worker(string Name,int Age,int Rang);
    void show();
};
//класс «служащий»
class employe:public person{
protected:
string post; //должность
public:
    employe(string Name,int age,string Post);
    void show();
};
```

Определение функций – членов класса помещается в файл *.cpp

Пример файла *person.cpp*

```
#include "person.h"
person::person(std::string Name, int Age)
{name=Name; age=Age;}

worker::worker(string Name,int Age,int Rang):person(Name,Age)
{rang=Rang;}
void worker::show()
{cout<<"имя:"<<name<<"  возраст:"<<age<<"  стаж:"<<rang<<endl;}

employe::employe(string Name,int Age,string Post):person(Name,Age)
```

```
{post=Post;}
void employe::show()
{cout<<"имя:"<<name<<"  возраст:"<<age<<"  должность:"<<post<<endl;}
```

2. После создания и отладки иерархии классов создается меню для работы с программой. Вызов меню помещается в функцию main(). Меню предусматривает вызов глобальных функций для выполнения различного вида работ. Эти функции размещаются там же, где и функция main(). Здесь же в глобальной области определяются две переменные - контейнер, хранящий указатели на объекты и указатель типа базового класса.

Пример файла cwork11.cpp

```
#include"person.h"
using namespace std;
int CreateWorker();          //создать объект «рабочий»
int CreateEmployer();        //создать объект «служащий»
int ViewFactory();           //просмотреть контейнер «предприятие»
int ClearFactory();          //ликвидировать «предприятие»
int Save();                  // сохранить контейнер в файле
int Load();                  // загрузить контейнер из файла
int Sort();                  //  сортировать объекты в контейнере
int Find();                  // найти объект в контейнере
int Inquiry();               // выполнить запрос

vector<person*> fact; //одномерный массив указателей типа person*
person* p;

int main()
{
SetConsoleOutputCP(1251);
int i;
do {
system ("cls");  //очистка экрана
cout<<"          <<ГЛАВНОЕ МЕНЮ>>          "<<endl;
cout<<"<Выберите пункт меню>"<<endl;
cout<<"<1>.Создать объект-рабочий"<<endl;
cout<<"<2>.Создать объект-инженер"<<endl;
cout<<"<3>.Просмотреть фабрику"<<endl;
cout<<"<4>.Очистить фабрику"<<endl;
cout<<"<5>.Сохранить в файле"<<endl;
cout<<"<6>.Загрузить из файла"<<endl;
cout<<"<7>.Отсортировать объекты в контейнере"<<endl;
cout<<"<8>.Выполнить поиск"<<endl;
cout<<"<9.Выход в WINDOWS"<<endl<<endl;
cout<<"Выбранный пункт меню:";
```

```
cin>>i;
  switch(i)
  {
  case 1:
  {
   system("cls");
    //вызов функции для создания объекта и помещения его в контейнер
  CreateWorker();
            system ("pause");
            break;                    //Возвращаемся в главное меню
              }
    case 2:
            {
            system("cls");
           CreateEmployer();
           system("pause");
           break;
              }
  case 3:
            {
           system("cls");
           ViewFactory();
           system("pause");
           break;
              }
    case 4:
            {
           system("cls");
                    ClearFactory();
                    system("pause");
                    break;
              }
    case 5:
            {
           system("cls");
           Save();
           system("pause");
           break;
              }
  case 6:
            {
           system("cls");
           Load();
           system("pause");
           break;
```

```
            }
  case 7:
            {
            system("cls");
            Sort();
            system("pause");
            break;
            }
case 8:
            {
            system("cls");
            Find();
            system("pause");
            break;
            }
default:if (i!=9 {cout<<"Вводите значения от 1 до 9<<endl;
system("pause"); break;}}
}
while (i!=9;
return 0;}
int CreateWorker()
{cout<<"Создание объекта Worker"<<endl;
return 0;
}
int CreateEmployer()
{cout<<"Создание объекта Employe"<<endl;
return 0;
}
int ViewFactory()
{cout<<"Просмотр фабрики"<<endl;
return 0;
}
int ClearFactory()
{cout<<"Удаление фабрики"<<endl;
return 0;
}
int Save()
{cout<<"Сохранение фабрики"<<endl;
  return 0;
}
int Load()
{cout<<"Загрузка фабрики"<<endl;
  return 0;
}
int Sort()
```

```
{cout<<"Сортировка"<<endl;
  return 0;
}
int Find()
{cout<<"Поиск"<<endl;
  return 0;
}
```

На этом этапе разработки программы функции реализованы в виде заглушек, и каждый их вызов сопровождается выдачей сообщения.

II этап. Создание объектов. Использование контейнеров.

1.После отладки меню реализуются функции создания объектов и помещения их в контейнер. На этом этапе объект инициализируется без ввода данных с клавиатуры непосредственно в теле функции. В качестве контейнера используется контейнер vector<>. В дальнейшем (на этапе IV) мы должны заменить контейнер vector контейнером, заданным в соответствии с вариантом (смотри приложение 2).

Пример реализации функций для контейнера **vector –** одномерный динамический массив**.**

```
int CreateWorker()
{cout<<"Создание объекта Worker"<<endl;
string ss="Иванов Иван";
int ii=23;
int rr=5;
p= new worker(ss,ii,rr);
fact.push_back(p); //объект добавляется в конец массива
return 0;
}
```

Следует убедиться, что объекты создаются и помещаются в контейнер.

2. Дальше следует реализовать просмотр и очистку контейнера.

Пример функций

```
int ViewFactory()
{cout<<"Просмотр фабрики"<<endl;
int k=fact.size(); //получаем количество элементов массива
if(k==0){cout<<"Фабрика пуста"<<endl;return 1;}
for(int i=0;i<k;i++) fact[i]->show();
return 0;
}

int ClearFactory()
{cout<<"Удаление фабрики"<<endl;
int k=fact.size();
```

```
for(int i=0;i<k;i++) delete fact[i];//удаляем все объекты
fact.clear();//очищаем фабрику
return 0;
}
```

III этап. Сохранение объектов в файле.

К третьему этапу переходим после того, как убедимся, что объекты помещаются в контейнер, контейнер можно просмотреть и очистить.

На этом этапе надо, во-первых, запрограммировать инициализацию объектов вводом данных с клавиатуры и, во-вторых, разработать модули для сохранения объектов в файле и загрузки их из файла. Основная сложность заключается в том, что, поскольку в контейнере хранятся полиморфные объекты разных классов, то и в файл должны записываться объекты разных классов. Следовательно, файл должен быть полиморфным. Основная трудность при работе с таким файлом связана с тем, что объекты разных классов имеют разный размер. Мы должны иметь механизм для записи в файл и чтении из файла количества байт в зависимости от типа записываемого(читаемого) объекта. Одним из способов решения этой задачи является запись перед полями объекта идентификатора, определяющего тип объекта. Например, это может быть целое число.

Пример.

В глобальной области определим объект потокового класса

```
fstream f;
```

Здесь же определим переменные - структуры, содержащие те же поля, что и объекты. Они понадобятся нам для записи/чтении объектов.

```
struct zap_worker{
string name;
int age;
int rang;
};
struct zap_employe{
string name;
int age;
string post;
};
```

Напишем также функции для записи объектов в файл и для чтения из файла.

```
void WritePersons(vector<person*>fact,fstream& f);
void ReadPersons(vector<person*>fact,fstream& f);
```

Примерная их реализация следующая.

```
void WritePersons(vector<person*>fact ,fstream& f)
{
//Открываем файлы для записи
f.open("my.dat",ios::binary|ios::out);
f.seekp(0);
```

```
//Запись объектов в файл
int n=fact.size();
for(int i=0;i<n;i++)
{
if(typeid(fact[i])==typeid(worker)) WriteWorker(fact[i],f);
if(typeid(fact[i])==typeid(employe))WriteEmploye(fact[i],f);
}
f.close();
}
```

Здесь WriteWorker() и WriteEmploye вспомогательные функции, имеющие примеро следующий вид

```
void WriteWorker(person& ob,fstream& f)
{
int k=1; //идентификатор рабочего
worker *w;
zap_worker zap;
w=dynamic_cast<worker*>(&ob);
zap.name=w->GetName();
zap.age=w->GetAge();
zap.rang=w->GetRang();
f.write((char*)&k,sizeof(int)); //записываем в файл идентификатор
f.write((char*)&zap,sizeof(zap)); //записываем в файл поля
};
```

Функция для чтения объектов из файла**:**

```
void ReadPersons(vector<person*>fact,fstream& f)
{
int k; //идентификатор класса объекта
zap_worker zap1;
worker* w;
zap_employe zap2;
employer* p;
f.open("my.dat",ios::binary|ios::in|ios::out); // открываем файл для чтения
f.clear();
f.seekp(0);
f.read((char*)&k,sizeof(int)); //читаем идентификатор
if(k==1) f.read((char*)&zap1,sizeof(zap1));
if(k==2) f.read((char*)&zap2,sizeof(zap2));
while(!f.eof())
{
if(k==1)
{
w=new worker(zap1.name,zap1.age,zap1.rank); fact.push_back(w);
}
if(k==2)
```

```
{
p=new engineer(zap2.name,zap2.age,zap2.category); fact.push_back(p);
}
f.read((char*)&k,sizeof(int));
if(k==1) f.read((char*)&zap1,sizeof(zap1));
if(k==2) f.read((char*)&zap2,sizeof(zap2));
}
f.close();
}
```

При вводе данных с клавиатуры необходимо учесть ввод строк, содержащих пробелы. Использовать для ввода таких слов перегруженную операцию “>>” нельзя, так как она рассматривает пробел, как конец ввода символов. Поэтому, для ввода строк следует использовать метод класса istream::getline(). Числа также надо вводить как строки с последующим преобразованием.

Пример корректного ввода строк и чисел

```
#include<iostream>
#include<string>
#include<stdlib.h>//для функции atoi()
#include<windows.h>
using namespace std;
struct person
{
string name;
string street;
int house;
string mail;
int age;
};
void input(person&p)
{
char ss[10];
cout<<"Имя->";          getline(cin,p.name,'\n');
cout<<"Улица->";        getline(cin,p.street,'\n');
```

Числовые данные вводим в строку char*, затем строку преобразуем в число

```
cout<<"Дом->";          cin.getline(ss,10); p.house=atoi(ss);
cout<<"e-mail->";       getline(cin,p.mail,'\n');
cout<<"Возраст->";      cin.getline(ss,10); p.age=atoi(ss);
}
```

IV этап. Сортировка контейнера. Поиск.

1. Первым делом необходимо заменить контейнер vector контейнером, заданным в соответствии с вариантом (смотри приложение 2) и отладить реализованную к этому этапу программу для нового контейнера.

Затем реализуем функцию Sort(). При выборе этой функции выводится подменю с двумя пунктами: «Сортировка STL» и «Сортировка программная».

«Сортировка STL» предполагает использование стандартного алгоритма STL. Функция сортировки зависит от типа контейнера. Так для контейнеров **vector** и **deque** - это функция sort(). В случае контейнера **list** используется метод контейнера **list::sort().** Ассоциативные контейнеры **map** и **multimap** сортируют свои элементы автоматически, т.е. в них элементы всегда располагаются отсортированными по ключу.

Чтобы использовать алгоритмы стандартной библиотеки C++, необходимо включить в программу заголовочный файл <algorithm>.

«Сортировка программная» предполагает написание своей функции сортировки. Метод сортировки выбирается в соответствии с вариантом. Эта функция должна принимать контейнер(или итераторы начала и конца сортируемой последовательности) и функцию сравнения. В качестве функции сравнения следует использовать объект-функцию. В случае ассоциативного контейнера функция принимает этот контейнер, а возвращает контейнер **vector**, элементы которого отсортированы по полю, не совпадающему с ключом ассоциативного контейнера. В этом случае следует дополнительно предусмотреть просмотр отсортированного контейнера **vector.**

Чтобы использовать объекты-функции и функциональные адаптеры, необходимо включить в программу заголовочный файл <functional>.

2. Реализуем функцию Find().При выборе этой функции выводится подменю с двумя пунктами: «Поиск STL» и «Поиск программный». Значение поля объекта для поиска следует вводить в диалоге.

«Поиск STL» предполагает использование для поиска подходящего алгоритма стандартной библиотеки C++. Для ассоциативных контейнеров поиск осуществляется прямым обращением по ключу.

«Поиск программный» предполагает написание своей функции поиска. Следует использовать бинарный поиск в отсортированной последовательности. Эта функция должна принимать контейнер(или итераторы начала и конца отсортированной последовательности) и значение поля поиска. Возвращать функция должна указатель(итератор) на найденный элемент. В случае ассоциативного контейнера поиск ведется в отсортированном вспомогательном контейнере **vector.**

V этап. Программирование запросов

Для выполнения запроса следует использовать алгоритмы STL, предикаты и объекты-функции.

VI этап. Оформление отчета

Отчет по курсовой работе состоит из:

- пояснительной записки;
- приложения;

- диска CD.

Содержание пояснительной записки

1. Титульный лист с указанием варианта.
2. Аннотация на 0.5 страницы.
3. Оглавление.
4. Постановка задачи. Постановка задачи должна быть конкретная с указанием классов, контейнера, алгоритмов, запроса и функций, выполняемых программой.
5. Структура программы. Описывается логическая и физическая структура программы.

Логическая структура включает определения классов, глобальных переменных и функций. Приводится диаграмма классов в виде графа.

Физическая структура представляет распределение классов, переменных и функций по файлам. Указываются используемые стандартные библиотеки(подключаемые файлы).

6. Описание программы. Здесь описываются классы: название класса, обязанности класса, компонентные данные и их назначение, компонентные функции и их назначение. Описываются также глобальные (не члены классов) функции. Описывается логика работы программы: диалог с пользователем, создание и сохранение объектов, использование контейнеров, алгоритмов, потоков.
7. Описание основных алгоритмов: сортировки, поиска, запись и чтение объектов, выполнение запроса. Для алгоритмов сортировки и поиска приводятся блок-схемы.
8. Результаты тестирования. Порядок тестирования, тестовые данные, скриншоты.
9. Список использованной литературы.

Приложение

В приложение помещается листинг программы.

Состав CD

- машинный вариант пояснительной записки;
- исходные файлы программы. Только те файлы, которые необходимы для компиляции и компоновки программы в среде Microsoft Visual Studio;
- выполняемый exe-файл;
- файл с данными, содержащий не менее 10 объектов.

CD должен быть помещен в подписанный конверт.

VII этап

Защита курсовой работы

График выполнения курсовой работы

недели	Выполняемая работа
4	Получение задания. Осмысление задачи.
4-5	I этап. Программирование классов. Создание меню с заглушками.
6-8	II этап. Использование контейнеров.
9-10	III этап. Программирование сохранения объектов в файле.
11-12	IV этап. Замена контейнера. Программирование сортировки и поиска
13-14	V этап. Программирование выполнения запросов.
15	VI этап. Оформление отчета.
16-17	VII этап. Защита курсовой работы.

Приложение 1. Перечень классов:

1. студент, преподаватель, **персона;**
2. **персона**, рабочий, инженер;
3. рабочий, **кадры**, администрация;
4. деталь, механизм**, изделие;**
5. **организация**, судостроительный завод, промышленное предприятие;
6. книга, **печатное издание**, учебник;
7. тест, выпускной экзамен, **испытание;**
8. **место**, город, мегаполис;
9. игрушка, продукт, **товар;**
10. квитанция, накладная, **документ;**
11. автомобиль, поезд, **транспортное средство;**
12. **двигатель**, двигатель внутреннего сгорания, турбореактивный двигатель;
13. республика, монархия, **государство;**
14. млекопитающие, парнокопытные, **животное;**
15. **корабль**, пароход, парусник;
16. **идентификатор**, адрес, электронный адрес;
17. **производство**, цех, мебельный цех;
18. **организация**, учреждение культуры, театр;
19. преподаватель, **персона**, заведующий кафедрой;
20. служащий, **персона**, инженер;
21. рабочий, **кадры**, инженер;
22. деталь, **изделие**, узел;
23. **организация**, страховая компания, промышленное предприятие;
24. журнал, книга, **печатное издание;**
25. экзамен, выпускной экзамен, **испытание;**
26. **место**, область, город;

27. продукт, **товар**, молочный продукт;
28. квитанция, **документ**, чек;
29.автомобиль, поезд, **транспортное средство**, экспресс;
30.**двигатель**, двигатель внутреннего сгорания, дизель
31. диктатура, королевство, **государство;**
32.млекопитающие, птицы, **животное;**
33. **корабль**, парусник, корвет;
34.**идентификатор**, адрес, имя;
35.**цех**, мебельный цех, пищеблок
36.**учреждение культуры**, библиотека, театр

Здесь **выделены** абстрактные классы.

Приложение 2. Варианты контейнеров и методов сортировки

Номер варианта	Тип контейнера	Метод сортировки
1	**deque**	**Хоора**
2	**list**	**простого обмена**
3	**deque**	**простого выбора**
4	**map**	**простого включения**
5	**deque**	**древовидная**
6	**list**	**пирамидальная**
7	**deque**	**слиянием**
8	**map**	**Хоора**
9	**deque**	**Хоора**
10	**list**	**древовидная**
11	**deque**	**простого обмена**
12	**map**	**Хоора**
13	**deque**	**простого обмена**
14	**list**	**простого обмена**
15	**deque**	**древовидная**
16	**map**	**простого выбора**
17	**deque**	**простого выбора**
18	**list**	**простого выбора**
19	**deque**	**простого выбора**
20	**map**	**древовидная**
21	**deque**	**простого включения**
22	**list**	**простого включения**
23	**deque**	**простого включения**
24	**map**	**простого включения**
25	**deque**	**древовидная**
26	**list**	**древовидная**
27	**deque**	**простого обмена**
28	**map**	**пирамидальная**
29	**deque**	**слиянием**
30	**list**	**древовидная**
31	**deque**	**Хоора**
32	**map**	**слиянием**

33	**deque**	**пирамидальная**
34	**list**	**слиянием**
35	**deque**	**древовидная**
36	**map**	**простого включения**

Приложение 3. Варианты запросов

Номер варианта	запросы
1	Вывести имена всех неуспевающих студентов
2	Вывести имена всех рабочих(инженеров)
3	Вывести имена всех рабочих заданного разряда
4	Вывести названия всех деталей с весом не больше заданного
5	Подсчитать общую численность рабочих на заданном заводе
6	Подсчитать количество учебников по заданной теме
7	Вывести фамилии всех отличников
8	Подсчитать общую численность населения всех городов
9	Вычислить среднюю стоимость игрушек
10	Вывести номера накладных, выписанных после заданной даты
11	Вывести номер поезда с максимальным количеством вагонов
12	Вычислить суммарную мощность всех двигателей заданного типа
13	Вывести площадь заданной республики
14	Вывести наименование парнокопытного с максимальным средним весом
15	Подсчитать среднее водоизмещение всех пароходов
16	Вывести электронный адрес персоны, проживающей по заданному адресу
17	Подсчитать суммарную мощность всех станков во всех цехах
18	Найти театр, в котором идет заданная пьеса
19	Вывести фамилии всех преподавателей, ведущих занятия на заданном курсе
20	Вывести фамилии всех инженеров заданного подразделения
21	Вывести фамилии всех рабочих, заданного цеха
22	Вывести наименования всех деталей, входящих в заданный узел
23	Вывести наименования страховых компаний с уставным капиталом выше заданного
24	Вывести наименования журналов, выходящих не реже 6 раз в год
25	Вывести наименования экзаменов в заданном семестре
26	Вывести наименования городов заданной области
27	Вывести наименования продуктов жирностью не выше заданной
28	Найти все чеки, выписанные за заданный период
29	Найти экспресс в заданном направлении
30	Найти самый дешевый дизель
31	Вывести государства с диктатурой
32	Найти самую большую птицу
33	Определить среднюю скорость всех кораблей, заданного типа
34	Найти адрес, по которому проживает самый старый человек
35	Найти цех, производящий заданную мебель
36	Вывести адреса всех районных библиотек