Программирование

Объектно-ориентированное программирование на языке C++

Лабораторная работа № 3
ИЕРАРХИЯ ОБЪЕКТОВ И ГРУППА. ИТЕРАТОРЫ

Цель. Получить практические навыки создания объектов-групп и использования методов-итераторов.

Основные теоретические сведения.

Группа.

Группа − это объект, в который включены другие объекты. Объекты, входящие в группу, называются *элементами группы*. Элементы группы, в свою очередь, могут быть группой.

Примеры групп:

1. Окно в интерактивной программе, которое владеет такими элементами, как поля ввода и редактирования данных, кнопки, списки выбора, диалоговые окна и т.д. Примерами таких окон являются объекты классов, объекты классов, порожденных от *TWinControl* в иерархии классов библиотеки **VCL** Delphi и **C++Builder** , объекты **CWnd** в библиотеке MFC.

2. Агрегат, состоящий из более мелких узлов.

3. Огород, состоящий из растений, системы полива и плана выращивания.

4. Некая организационная структура (например, ФАКУЛЬТЕТ, КАФЕДРА, СТУДЕНЧЕСКАЯ ГРУППА).

Группа более широкое понятие, чем контейнер. Контейнер используется для хранения данных одного типа. Примеры контейнеров: объекты контейнерных классов библиотеки **STL** в **C++** (массивы, списки, очереди), коллекции в библиотеке **MFC.** Группа может содержать объекты разных классов.

Мы понимаем группу как класс, который не только хранит объекты других классов, но и обладает собственными свойствами, не вытекающими из свойств его элементов.

Группа дает второй вид иерархии (первый вид − *иерархия классов*, построенная на основе наследования) − *иерархию объектов* (иерархию типа *целое/часть*), построенную на основе агрегации.

Реализовать группу можно несколькими способами:

1. Класс “группа” содержит поля данных объектного типа. Таким образом, объект “группа” в качестве данных содержит либо непосредственно свои элементы, либо указатели на них

```
class TWindowDialog: public TGroup
{
 protected:
TInputLine input1;
TEdit   edit1;
TButton button1;
/*другие члены класса*/
};
```

Такой способ реализации группы используется в **C++Builder.**

2. Группа содержит член-данное *last* типа *TObject**, который указывает на начало связанного списка объектов, включенных в группу. В этом случае объекты должны иметь поле *next* типа *TObject**, указывающее на следующий элемент в списке. Такой способ используется при реализации групп в **Turbo Vision.**

3. Создается связанный список структур типа *item:*

```
struct  item
 {
  object* item;
  item* next;
};
```

Поле *item* указывает на объект, включенный в группу. Группа *group* содержит поле *begin* типа *item**, которое указывает на начало связанного списка структур типа *item.*

Если необходим доступ элементов группы к ее полям и методам, объект типа *object* должен иметь поле *owner* типа *group**, которое указывает на собственника этого элемента. Такой способ используется в контейнерах и коллекциях C++.

Методы группы.

Имеется два метода, которые необходимы для функционирования группы:

1) *void Insert(object* p);*

Вставляет элемент в группу.

2) *void Show();*

Позволяет просмотреть группу.

Кроме этого группа может содержать следующие методы:

1) *int Empty();*

Показывает, есть ли хотя бы один элемент в группе.

2) o*bject* Delete(object* p);*

Удаляет элемент из группы, но сохраняет его в памяти.

3) *void DelDisp(object* p);*

Удаляет элемент из группы и из памяти.

Иерархия объектов.

Иерархия классов есть иерархия по принципу наследования, т.е. типа “это есть разновидность того”. Например, “рабочий есть разновидность персоны”, “автомобиль” есть разновидность “транспортного средства”. В отличие от этого **иерархия объектов** − это иерархия по принципу вхождения, т.е. типа “это есть часть того”. Например, “установка – часть завода”, “двигатель” – часть “автомобиля”. Таким образом, объекты нижнего уровня иерархии включаются в объекты более высокого уровня, которые являются для них группой.

Итератор.

Итераторы позволяют выполнять некоторые действия для каждого элемента определенного набора данных.

***Fo**r all элементов набора { действия}*

Такой цикл мог бы быть выполнен для всего набора, например, чтобы напечатать все элементы набора, или мог бы искать некоторый элемент, который удовлетворяет определенному условию, и в этом случае такой цикл может закончиться, как только будет найден требуемый элемент.

Мы будем рассматривать итераторы как специальные методы класса-группы, позволяющие выполнять некоторые действия для всех объектов, включенных в группу. Примером итератора является метод *Show*.

Нам бы хотелось иметь такой итератор, который позволял бы выполнять над всеми элементами группы действия, заданные не одним из методов объекта, а произвольной функцией пользователя. Такой итератор можно реализовать, если эту функцию передавать ему через указатель на функцию.

Определим тип указателя на функцию следующим образом:

*typedef void(*PF)(object*,< дополнительные параметры>);*

Функция имеет обязательный параметр типа *object* или *object**, через который ей передается объект, для которого необходимо выполнить определенные действия.

Метод-итератор объявляется следующим образом:

void group::ForEach(PF action,< дополнительные параметры >);

где

action − единственный обязательный параметр-указатель на функцию, которая вызывается для каждого элемента группы;

дополнительные параметры − передаваемые вызываемой функции параметры.

Затем определяется указатель на функцию и инициализируется передаваемой итератору функцией.

PF pf=myfunc;

Тогда итератор будет вызываться, например, для дополнительного параметра типа int, так:

gr.ForEach(pf,25);

Здесь *gr* − объект-группа.

Динамическая идентификация типов.

Динамическая идентификация типа характерна для языков, в которых поддерживается полиморфизм. В этих языках возможны ситуации, в которых тип объекта на этапе компиляции неизвестен.

В C++ полиморфизм поддерживается через иерархии классов, виртуальные функции и указатели базовых классов. При этом указатель базового класса может использоваться либо для указания на объект базового класса, либо для указания на объект любого класса, производного от этого базового.

Пусть группа содержит объекты различных классов и необходимо выполнить некоторые действия только для объектов определенного класса. Тогда в итераторе мы должны распознавать тип очередного объекта.

В стандарт языка C++ включены средства **RTTI** (Run-Time Type Idendification) – динамическая идентификация типов.

Информацию о типе объекта получают с помощью оператора typeid, определение которого содержит заголовочный файл <typeinfo.h>.

Имеется две формы оператора typeid:

typeid (объект)

typeid (имя_типа)

Оператор typeid возвращает ссылку на объект типа type_info.

В классе type_info перегруженные операции == и != обеспечивают сравнение типов.

Функция name () возвращает указатель на имя типа.

Имеется одно ограничение. Оператор typeid работает корректно только с объектами, у которых определены виртуальные функции. Большинство объектов имеют виртуальные функции, хотя бы потому, что обычно деструктор является виртуальным для устранения потенциальных проблем с производными классами. Когда оператор typeid применяют к не полиморфному классу (в классе нет виртуальной функции), получают указатель или ссылку базового типа.

Примеры.

1.

```
#include<iostream.h>
#include<typeinfo.h>
class Base{
virtual void f(){};
//…
};
class Derived: public Base{
//…
};
void main()
{int i;
```

```
Base ob,*p;
Derived ob1;
cout<<typeid(i).name();    //Выводится int
p=&ob1;
cout<<typeid(*p).name();  // Выводится Derived
}
```

2.

```
//начало см. выше
void WhatType(Base& ob)
{cout<< typeid(ob).name()<<endl;
}
void main()
{
Base ob;
Derived ob1;
WhatType(ob);    //Выводится Base
WhatType(ob1);  //Выводится Derived
}
```

3.

```
//начало см. выше
void main()
{
Base *p;
Derived ob;
p=&ob;
if(typeid(*p)==typeid(Derived)) cout<<“p указывает на объект типа Derived”;
…
}
```

Если при обращении typeid(*p), p=NULL, то возбуждается исключительная ситуация bad_typeid

Порядок выполнения работы.

1. Дополнить иерархию классов лабораторной работы № 2 классами “группа”.

Например, для предметной области ФАКУЛЬТЕТ можно предложить классы “*факультет*”, “*студенческая группа*”, “*кафедра*”. Рекомендуется создать абстрактный класс – “*подразделение*”, который будет предком всех групп и абстрактный класс *object*, находящийся во главе всей иерархии.

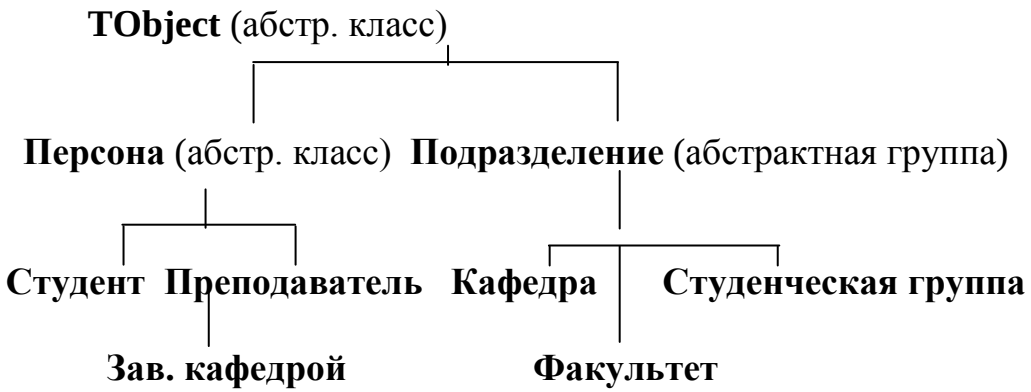
2. Написать для класса-группы метод-итератор (итераторы) для выполнения заданных запросов.

3. Написать функцию (функции), которая передается итератору для выполнения запросов.

4. Написать демонстрационную программу, в которой создаются, показываются и разрушаются объекты-группы, а также выполняются запросы к группе.

Методические указания.

1. Класс-группа должен соответствовать иерархии классов лабораторной работы № 2, т.е. объекты этих классов должны входить в группу. Например, для варианта 1 может быть предложена следующая иерархия классов:



При этом иерархия объектов будет иметь следующий вид:

2. Для включения объектов в группу следует использовать третий способ (через связанный список структур типа item).

3. Пример определения добавленных абстрактных классов:

```
class object
{
public:
virtual void Show()=0;};

class department:public object   // абстрактный класс-группа
{
```

```
protected:
string name;  // наименование
person* head; // руководитель
item* begin; // указатель на начало связанного списка структур item
public:
department(string,person*);
depardment(department&);
~ department();
string  GetName();
person* GetHead();
void SetName(string NAME);
void  SetHead(person* p);
void Insert(object* p);
virtual void Show()=0;
};
```

4. Иерархия объектов создается следующим образом (на примере ФАКУЛЬТЕТА):

а) создается пустой ФАКУЛЬТЕТ,

б) создается пустая КАФЕДРА,

в) создаются ПРЕПОДАВАТЕЛИ и включаются в КАФЕДРУ,

г) КАФЕДРА включается в ФАКУЛЬТЕТ,

д) тоже повторяется для другой кафедры,

е) создается пустая СТУДЕНЧЕСКАЯ ГРУППА,

ж) создаются СТУДЕНТЫ и включаются в СТУДЕНЧЕСКУЮ ГРУППУ,

з) СТУДЕНЧЕСКАЯ ГРУППА включается в ФАКУЛЬТЕТ,

и) тоже повторяется для другой студенческой группы.

5. Удаляется ФАКУЛЬТЕТ (при вызове деструктора) в обратном порядке.

6. Метод-итератор определяется в неабстрактных классах-группах на основе выбранных запросов.

Например, для класса *StudentGroup* может быть предложен итератор *void StudentGroup::ForEach(PF action, doublet parametr);*
где *action* – указатель на функцию, которая должна быть выполнена для всех объектов, включенных в группу (в данном случае для всех СТУДЕНТОВ), *parametr*-передаваемая процедуре дополнительная информация.

В качестве передаваемой методу функции может быть предложена, например, такая: вывести список студентов, имеющих рейтинг не ниже заданного

```
void MyProc(object* p, double rate)
{
if (((student*)p) ->GetGrade()>=rate) cout<<(((student*)p) ->GetName());
}
```

7. Передаваемые итератору функции следует определить на основе запросов, которые должны быть выполнены вызовом итератора. Варианты запросов приведены в приложении 1 и задаются вариантом (приложение 2).

8. При необходимости (для выполнения запроса) в класс могут быть добавлены новые поля (по сравнению с лабораторной работой № 2).

9.Для просмотра всех объектов вызывается метод show() группы верхнего уровня.

10.В программе должно быть минимум ввода с клавиатуры. Поля объектов задаются в тексте программы. С клавиатуры вводятся только параметры итераторов.

11.Вызов итератора следует организовать в виде бесконечного цикла-выход при вводе заранее оговоренного значения.

Содержание отчета.

1. Титульный лист.
2. Постановка задачи.
3. Иерархия классов в виде графа.
4. Иерархия объектов в виде графа.
5. Определение классов (только добавленных или измененных по сравнению с лабораторной работой № 2) с комментариями.
6. Реализация для классов-групп всех методов с пояснением.
7. Реализация итераторов с пояснением.
8. Реализация передаваемых итераторам функций с пояснением.
9. Листинг демонстрационной программы с комментариями.

Приложение 1. Запросы.

1. Имена всех лиц мужского (женского) пола.

2. Имена студентов указанного курса.

3. Имена и должность преподавателей указанной кафедры.

4. Имена служащих со стажем не менее заданного.

5. Имена служащих заданной профессии.

6. Имена рабочих заданного цеха.

7. Имена рабочих заданной профессии.

8. Имена студентов, сдавших все (заданный) экзамены на отлично (хорошо и отлично).

9. Количество инженеров на заводе.

10. Имена всех монархов на заданном континенте.

11. Наименование всех деталей (узлов), входящих в заданный узел (механизм).

12. Наименование всех книг в библиотеке (магазине), вышедших не ранее указанного года.

13. Названия всех городов заданной области.

14. Наименование всех товаров в заданном отделе магазина.

15. Количество мужчин (женщин).

16. Количество студентов на указанном курсе.

17. Количество рабочих со стажем не менее заданного.

18. Количество рабочих заданной профессии.

19. Количество инженеров в заданном подразделении.

20. Количество товара заданного наименования.

21. Количество студентов, сдавших все экзамены на отлично.

22. Количество студентов, не сдавших хотя бы один экзамен.

23. Количество деталей (узлов), входящих в заданный узел (механизм).

24. Количество указанного транспортного средства в автопарке (на автостоянке).

25. Количество пассажиров во всех вагонах экспресса.

26. Суммарная стоимость товара заданного наименования.

27. Средний балл за сессию заданного студента.

28. Количество библиотек в городе.

29. Суммарное количество учебников в библиотеке (магазине).

30. Суммарное количество жителей всех городов в области.

31. Суммарная стоимость продукции заданного наименования по всем накладным.

32. Средняя мощность всех (заданного типа) транспортных средств в организации.

33. Средняя мощность всех дизелей, обслуживаемых заданной фирмой.

34. Средний вес животных заданного вида в зоопарке.

35. Среднее водоизмещение всех парусников на верфи (в порту).

36. Суммарный вес всех деталей в заданном узле.

37.Количество рабочих в заданном цехе.

38. Наименование всех цехов на данном заводе.

39. Количество жителей данного континента.

40. Наименование птиц в зоопарке.

41. Имена пароходов, приписанных к данному порту.

42. Количество различных типов ДВС, обслуживаемых автомастерской.

43. Наименование журналов, выписываемых библиотекой.

44. Суммарная стоимость всех деталей в механизме.

45. Суммарный страховой фонд всех страховых компаний региона.

46. Количество книг во всех библиотеках города.

47.Самый мощный автомобиль в данной организации.

48. Количество чеков на сумму превышающую заданную.

49. Общая сумма по всем чекам, выписанным в организации.

50. Самая дорогая и самая дешевая игрушка в магазине(наименование и стоимость).

Приложение 2. Варианты

№ варианта лаб. работы №3	№ варианта лаб. работы №2	Варианты запро-сов
1	1	2, 21
2	2	5, 19
3	3	6, 18
4	4	11, 36
5	5	12, 28
6	6	8, 22
7	7	13, 30
8	8	14, 20
9	9	1, 31
10	10	37, 38
11	11	11, 27
12	12	25, 32
13	13	10, 39
14	14	34, 40
15	15	35, 41
16	16	33, 42
17	17	29, 43
18	1	3, 16
19	2	4, 17
20	3	7, 9
21	4	23, 44
22	5	45, 46
23	8	26, 50
24	9	48, 49
25	12	24, 47