Лабораторная работа №4

Простое наследование. Принцип подстановки.

1. Цель задания:

1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2) Создание иерархии классов с использованием простого наследования.
3) Изучение принципа подстановки.

2. Теоретические сведения

2.1. Простое открытое наследование

Наследование – это механизм получения нового класса на основе уже существующего. Существующий класс может быть дополнен или изменен для создания нового класса.

Существующие классы называются **базовыми**, а новые – **производными**. Производный класс наследует описание базового класса; затем он может быть изменен добавлением новых членов, изменением существующих функций-членов и изменением прав доступа. С помощью наследования может быть создана иерархия классов, которые совместно используют код и интерфейсы.

Наследуемые компоненты не перемещаются в производный класс, а остаются в базовых классах.

В иерархии производный объект наследует разрешенные для наследования компоненты всех базовых объектов (*public, protected*).

Допускается множественное наследование – возможность для некоторого класса наследовать компоненты нескольких никак не связанных между собой базовых классов. В иерархии классов соглашение относительно доступности компонентов класса следующее:

private – член класса может использоваться только функциями – членами данного класса и функциями – "друзьями" своего класса. В производном классе он недоступен.

protected – то же, что и ***private,*** но дополнительно член класса с данным атрибутом доступа может использоваться функциями-членами и функциями – "друзьями" классов, производных от данного.

public – член класса может использоваться любой функцией, которая является членом данного или производного класса, а также к ***public*** - членам возможен доступ извне через имя объекта.

Следует иметь в виду, что объявление friend не является атрибутом доступа и не наследуется.

Синтаксис определения производного класса:

```
class имя_класса : список_базовых_классов
{список_компонентов_класса};
```

2.2. Конструкторы и деструкторы при наследовании

Поскольку конструкторы не наследуются, при создании производного класса наследуемые им данные-члены должны инициализироваться конструктором базового класса. Конструктор базового класса вызывается автоматически и выполняется до конструктора производного класса. Параметры конструктора базового класса указываются в определении конструктора производного класса. Таким образом, происходит передача аргументов от конструктора производного класса конструктору базового класса.

```
//базовый класс
class Base
{
protected:
     int a,b;
public:
     Base(int x,int y){a=x;b=y;}
};
//производный класс
class Derive:public Base
{
protected:
     int sum;
public:
     Derive (int x,int y, int s):Base(x,y){sum=s;}
};
```

Объекты класса конструируются снизу вверх: сначала базовый, потом компоненты-объекты (если они имеются), а потом сам производный класс. Таким образом, объект производного класса содержит в качестве подобъекта объект базового класса.

Уничтожаются объекты в обратном порядке: сначала производный, потом его компоненты-объекты, а потом базовый объект.

Таким образом, порядок уничтожения объекта противоположен по отношению к порядку его конструирования.

2.3. Виртуальные функции.

К механизму виртуальных функций обращаются в тех случаях, когда в каждом производном классе требуется свой вариант некоторой компонентной функции. Классы, включающие такие функции, называются **полиморфными** и играют особую роль в ООП.

Виртуальные функции предоставляют механизм **позднего** (**отложенного**) или **динамического связывания.** Любая нестатическая функция базового класса может быть сделана виртуальной, для чего используется ключевое слово **virtual**.

```
class Base
{
public:
  virtual void print(){cout<<"\nBase";}
. . .
};

class Derive : public Base
{
public:
  void print(){cout<<"\n Derive";}
};
void main()
{
  Base B,*bp;
  Derive D,*dp;
   bp=&B;
   dp=&D;
   //указатель базового класса ставится на объект
   //производного класса
   Base *p = &D;
  bp ->print(); // вызывается метод для Base
```

```
    dp ->print(); // вызывается метод для Derive
    p ->print(); // вызывается метод для Derive
}
```

Таким образом, интерпретация каждого вызова виртуальной функции через указатель на базовый класс зависит от значения этого указателя, т.е. от типа объекта, для которого выполняется вызов.

Выбор того, какую виртуальную функцию вызвать, будет зависеть от типа объекта, на который фактически (в момент выполнения программы) направлен указатель, а не от типа указателя.

Виртуальными могут быть только нестатические функции-члены.

Виртуальность наследуется. После того как функция определена как виртуальная, ее повторное определение в производном классе (с тем же самым прототипом) создает в этом классе новую виртуальную функцию, причем спецификатор virtual может не использоваться.

Конструкторы не могут быть виртуальными, в отличие от деструкторов. Практически каждый класс, имеющий виртуальную функцию, должен иметь виртуальный деструктор.

2.4. Принцип подстановки

Открытое наследование устанавливает между классами отношение «является»: класс-наследник является частью класса-родителя. Это означает, что везде, где может быть использован объект базового класса (при присваивании, при передаче параметров и возврате результата), вместо него разрешается использовать объект производного класса. Данное положение называется принципом подстановки. Он работает и для ссылок и для указателей. Обратное неверно. Например, всякий спортсмен (производный класс) является человеком (базовый класс), но не всякий человек является спортсменом.

Закрытое наследование – это наследование реализации, в этом случае принцип подстановки не соблюдается.

```
class Base
{
public:
void f1();
void f2();
};
class Derive: private Base //закрытое наследование
{……};
```

Программа, использующая класс Derive не может использовать ни f1() ни f2(). В наследнике заново реализуются все методы:

```
class Derive: private Base
{
public:
void f1(){Base::f1();};
void f2(){Base::f2();};
};
```

3. Постановка задачи

1. Определить пользовательский класс.
2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.
3. Определить в классе деструктор.

4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).
5. Перегрузить операцию присваивания.
6. Перегрузить операции ввода и вывода объектов с помощью потоков.
7. Определить производный класс.
8. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.
9. Реализовать функции, получающие и возвращающие объект базового класса. Продемонстрировать принцип подстановки.

4. Ход работы

Задача

Базовый класс:
МАШИНА
торговая_марка - string
число_цилиндров - int
мощность - int
Создать производный класс ГРУЗОВИК, добавив в него характеристику грузоподъемности кузова типа int.

1. Создать в проекте класс Car следующим образом:
   1.1. Создать пустой проект способом, рассмотренным в лабораторной работе №2.
   1.2. Добавить в проект класс Car, для чего вызвать контекстное меню проекта, выбрать в нем пункт Add/Class , в диалоговом окне Add class выбрать категорию C++, шаблон C++ Class, нажать кнопку Add.
   1.3. В диалоговом окне Generic C++ Class Wizard ввести имя класса (Class Name) Car.
   1.4. В результате будут сформированы 2 файла: Car.h, Car.cpp. В файле Car.h будет сформирована заготовка для класса

```
class Car
{
public:
	Car(void);
public:
	~Car(void);
};
```

   1.5. Ввести следующий текст программы

```
//описание класса
#pragma once
#include <string>
#include <iostream>
using namespace std;
class Car
{
//конструктор без параметров
public:
	Car(void);
public:
//деструктор
	virtual ~Car(void);
//констрктор с параметрами
	Car(string,int,int);
//конструктор копирования
	Car(const Car&);
```

```
//селекторы
	string Get_mark(){return mark;}
	int Get_cyl(){return cyl;}
	int Get_power(){return power;}
//модификаторы
	void Set_mark(string);
	void Set_cyl(int);
	void Set_power(int);
//перегрузка операции присваивания
	Car& operator=(const Car&);
//глобальные операторы-функции ввода-вывода
	friend istream& operator>>(istream&in,Car&c);
	friend ostream& operator<<(ostream&out,const Car&c);
//атрибуты
protected:
	string mark;
	int cyl;
	int power;

};
```

1.6. В файле Car.cpp будет сформирована заготовка файла

```
#include "Car.h"
Car::Car(void)
{
}
Car::~Car(void)
{
}
```

1.7. Добавить следующий текст программы

```
#include "Car.h"
//конструктор без параметров
Car::Car(void)
{
	mark="";
	cyl=0;
	power=0;
}
//деструктор
Car::~Car(void)
{
}
//конструктор с параметрами
Car::Car(string M,int C,int P)
{
	mark=M;
	cyl=C;
	power=P;
}
//конструктор копирования
Car::Car(const Car& car)
{
	mark=car.mark;
	cyl=car.cyl;
	power=car.power;
}
//модификаторы
void Car::Set_cyl(int C)
{
	cyl=C;
}
void Car::Set_mark(string M)
{
	mark=M;
```

```
}

void Car::Set_power(int P)
{
	power=P;
}
//перегрузка операции присваивания
Car& Car::operator=(const Car&c)
{
	if(&c==this)return *this;
	mark=c.mark;
	power=c.power;
	cyl=c.cyl;
	return *this;
}
//глобальная функция для ввода
 istream& operator>>(istream&in,Car&c)
{
	cout<<"\nMark:"; in>>c.mark;
	cout<<"\nPower:";in>>c.power;
	cout<<"\nCyl:";in>>c.cyl;
	return in;
}
//глобальная функция для вывода
ostream& operator<<(ostream&out,const Car&c)
{
	out<<"\nMARK : "<<c.mark;
	out<<"\nCYL : "<<c.cyl;
	out<<"\nPOWER : "<<c.power;
	out<<"\n";
	return out;
}
```

2. Добавить в проект файл Lab4_main.cpp (см. лабораторную работу 2 и 3) и основную программу, в которой проверим работу методов класса Car.

```
#include <iostream>
#include "Car.h"
using namespace std;
void main()
{
	Car a;
	cin>>a;
	cout<<a;
	Car b("Ford",4,115);
	cout<<b;
	a=b;
	cout<<a;
	}
```

3. Выполнить компиляцию программы, используя команду Build / Build Solution или функциональную клавишу F7. Исправить имеющиеся синтаксические ошибки и снова запустить программу на компиляцию.
4. Запустить программу на выполнение, используя команду Debug/ Start Without Debugging или комбинацию функциональных клавиш Ctrl+F5.
5. Добавить в проект класс Lorry (Грузовик). Для этого:
   5.1. вызвать контекстное меню проекта, выбрать в нем пункт Add/Class , в диалоговом окне Add class выбрать категорию C++, шаблон C++ Class, нажать кнопку Add.
   5.2. В диалоговом окне Generic C++ Class Wizard в поле Class Name ввести имя класса Lorry, в поле Base Class ввести имя базового класса Car.
   5.3. В результате будут сформированы 2 файла: Lorry.h, Lorry.cpp. В файле Lorry.h будет сформирована заготовка для класса

```
#pragma once
#include "car.h"
class Lorry :
	public Car
{
public:
	Lorry(void);
public:
	~Lorry(void);
};
```

5.4. Добавить текст программы:

```
#pragma once
#include "car.h"

//класс Lorry наследуется от класса Car
class Lorry :
	public Car
{
public:
	Lorry(void);//конструктор без параметров
public:
	~Lorry(void);//деструктор
	Lorry(string,int,int,int);//конструктор с параметрами
	Lorry(const Lorry & );//конструктор копирования
	int Get_gruz(){return gruz;}//модификатор
	void Set_Gruz(int);//селектор
	Lorry& operator=(const Lorry&);//операция присваивания
	friend istream& operator>>(istream&in,Lorry&l);//операция ввода
	friend ostream& operator<<(ostream&out,const Lorry&l); //операция вывода
protected:
	int gruz;//атрибут грузоподъемность

};
```

5.5. В файле Lorry.cpp будет сформирована заготовка файла. Изменить ее следующим образом, добавив определение методов класса Lorry:

```
#include "Lorry.h"
//конструктор без параметров
Lorry::Lorry(void):Car()
{
	gruz=0;
}
//дестрктор
Lorry::~Lorry(void)
{
}
//конструктор с параметрами
Lorry::Lorry(string M, int C, int P, int G):Car(M,C,P)
{
	gruz=G;
}
//конструктор копирования
Lorry::Lorry(const Lorry &L)
{
	mark=L.mark;
	cyl=L.cyl;
	power=L.power;
	gruz=L.gruz;
}
//модификатор
void Lorry::Set_Gruz(int G)
{
	gruz=G;
}
```

```
//оперция присваивания
Lorry& Lorry::operator=(const Lorry&l)
{
	if(&l==this)return *this;
	mark=l.mark;
	power=l.power;
	cyl=l.cyl;
	gruz=l.gruz;
	return *this;
}
//операция ввода
 istream& operator>>(istream&in,Lorry&l)
{
	cout<<"\nMark:"; in>>l.mark;
	cout<<"\nPower:";in>>l.power;
	cout<<"\nCyl:";in>>l.cyl;
	cout<<"\nGruz:";in>>l.gruz;
	return in;
}
//операция вывода
	 ostream& operator<<(ostream&out,const Lorry&l)
{
	out<<"\nMARK : "<<l.mark;
	out<<"\nCYL : "<<l.cyl;
	out<<"\nPOWER : "<<l.power;
	out<<"\nGRUZ : "<<l.gruz;
	out<<"\n";
	return out;
}
```

6. Проверить работу класса Lorry. Для этого добавить в файл Lab4_main.cpp следующие операторы:

```
Lorry c;//создать объект
cin>>c;//ввести значения атрибутов
cout<<c;//вывести значения атрибутов
```

7. Выполнить компиляцию программы и запустить ее на выполнение.
8. Реализовать функции, получающие и возвращающие объект базового класса. Для этого в файл Lab4_main.cpp добавить следующие глобальные функции

```
//функция принимает объект базового класса как параметр
void f1(Car&c)
{
	c.Set_mark("Opel");
	cout<<c;
}
//функция возвращает объект базового класса как результат
Car f2()
{
	Lorry l("Kia",1,2,3);
		return l;
}
```

9. В функцию main() добавить следующие операторы:

```
f1(c);//передаем объект класса Lorry
a=f2();//создаем в функции объект класса Lorry
cout<<a;
```

В результате файл Lab4_main.cpp будет иметь следующий вид:

```
#include <iostream>
#include "Car.h"
#include "Lorry.h"
using namespace std;
// глобальные функции
void f1(Car&c)
{
	c.Set_mark("Opel");
```

```
	cout<<c;
}
Car f2()
{
	Lorry l("Kia",1,2,3);
		return l;
}
void main()
{
//работа с классом Car
	Car a;
	cin>>a;
	cout<<a;
	Car b("Ford",4,115);
	cout<<b;
	a=b;
	cout<<a;
//работа с классом Lorry
	Lorry c;
	cin>>c;
	cout<<c;
//принцип подстановки
f1(c);//передаем объект класса Lorry
a=f2();//создаем в функции объект класса Lorry
cout<<a;
}
```

10. Выполнить компиляцию программы и запустить ее на выполнение.
11. Изучить полученные результаты, сделать выводы.

5. Варианты

№	Задание
1	Базовый класс: ПАРА_ЧИСЕЛ (PAIR) Первое_число (first) - int Второе_число (second) – int Определить методы изменения полей и сравнения пар (пара p1 больше пары p2, если (p1.first>p2.first) || (p1.first==p2.first &&p1.second>p2.second). Создать производный класс ДРОБЬ (FRACTION), с полями Целая_часть_числа и Дробная_часть_числа. Определить полный набор методов сравнения.
2	Базовый класс: ПАРА_ЧИСЕЛ (PAIR) Первое_число (first) - int Второе_число (second) – int Определить методы изменения полей и вычисления произведения чисел. Создать производный класс ПРЯМОУГОЛЬНИК (RECTANGLE), с полями-сторонами. Определить методы для вычисления площади и периметра прямоугольника.
3	Базовый класс: ПАРА_ЧИСЕЛ (PAIR) Первое_число (first) - int Второе_число (second) – int Определить методы изменения полей и вычисления произведения чисел. Создать производный класс ПРЯМОУГОЛЬНЫЙ_ТРЕУГОЛЬНИК (RIGHTANGLED), с полями-катетами. Определить метод вычисления гипотенузы.

4	Базовый класс: ПАРА_ЧИСЕЛ (PAIR) Первое_число (first) - int Второе_число (second) – int Определить методы изменения полей и операцию сложения пар (a,b)+(c,d)=(a+b,c+d) Создать производный класс КОМПЛЕКСНОЕ_ЧИСЛО(COMPLEX), с полями Действительная_часть_числа и Мнимая_часть_числа. Определить операции умножения (a,b)*(c,d)= (a*c-b*d, a*d+b*c) и вычитания (a,b)-(c,d)= (a-b, c-d)
5	Базовый класс: ПАРА_ЧИСЕЛ (PAIR) Первое_число (first) - int Второе_число (second) – int Определить методы изменения полей и операцию сложения пар (a,b)+(c,d)=(a+b,c+d) Создать производный класс ДЕНЕЖНАЯ_СУММА(MONEY), с полями Рубли и Копейки. Переопределить операцию сложения и определить операции вычитания и деления денежных сумм.
6	Базовый класс: ПАРА_ЧИСЕЛ (PAIR) Первое_число (first) - int Второе_число (second) – int Определить методы изменения полей и операцию сложения пар (a,b)+(c,d)=(a+b,c+d) Создать производный класс ДЛИННОЕ_ЧИСЛО(LONG), с полями Старшая_часть_числа и Младшая_часть_числа. Переопределить операцию сложения и определить операции вычитания и умножения.
7	Базовый класс: ПАРА_ЧИСЕЛ (PAIR) Первое_число (first) - int Второе_число (second) – int Определить методы проверки на равенство и операцию перемножения полей. Реализовать операцию вычитания пар по формуле (a,b)-(c,d)=(a-b,c-d) Создать производный класс ПРОСТАЯ_ДРОБЬ(RATIONAL), с полями Числитель и Знаменатель. Переопределить операцию вычитания и определить операции сложения и умножения простых дробей.
8	Базовый класс: ТРОЙКА_ЧИСЕЛ (TRIAD) Первое_число (first) - int Второе_число (second) – int Третье_число (third) - int Определить методы изменения полей и сравнения триады. Создать производный класс DATE с полями год, месяц и число. Определить полный набор операций сравнения дат.
9	Базовый класс: ТРОЙКА_ЧИСЕЛ (TRIAD) Первое_число (first) - int Второе_число (second) – int Третье_число (third) - int Определить методы изменения полей и сравнения триады.

	Создать производный класс TIME с полями часы, минуты и секунды. Определить полный набор операций сравнения временных промежутков.
10	Базовый класс: ТРОЙКА_ЧИСЕЛ (TRIAD) Первое_число (first) - int Второе_число (second) – int Третье_число (third) - int Определить методы изменения полей и увеличения полей на 1. Создать производный класс DATE с полями год, месяц и число. Переопределить методы увеличения полей на 1 и определить метод увеличения даты на n дней.
11	Базовый класс: ТРОЙКА_ЧИСЕЛ (TRIAD) Первое_число (first) - int Второе_число (second) – int Третье_число (third) - int Определить методы изменения полей и увеличения полей на 1. Создать производный класс TIME с полями часы, минуты и секунды. Переопределить методы увеличения полей на 1 и определить методы увеличения на n секунд и минут.
12	Базовый класс: ЧЕЛОВЕК (PERSON) Имя (name) – string Возраст (age) – int Определить методы изменения полей. Создать производный класс STUDENT, имеющий поле год обучения. Определить методы изменения и увеличения года обучения.
13	Базовый класс: ЧЕЛОВЕК (PERSON) Имя (name) – string Возраст (age) – int Определить методы изменения полей. Создать производный класс EMPLOYEE, имеющий поля Должность – string и Оклад – double. Определить методы изменения полей и вычисления зарплаты сотрудника по формуле Оклад+Премия(% от оклада).
14	Базовый класс: ЧЕЛОВЕК (PERSON) Имя (name) – string Возраст (age) – int Определить методы изменения полей. Создать производный класс TEACHER, имеющий поля Предмет – string и Количество часов – int. Определить методы изменения полей, а также увеличения и уменьшения часов.
15	Базовый класс: ЧЕЛОВЕК (PERSON) Имя (name) – string Возраст (age) – int Определить методы изменения полей. Создать производный класс STUDENT, имеющий поля Предмет – string и Оценка – int. Определить методы изменения полей и метод, выдающий сообщение о неудовлетворительной оценке.

6. Контрольные вопросы

1. Для чего используется механизм наследования?
2. Каким образом наследуются компоненты класса, описанные со спецификатором public?
3. Каким образом наследуются компоненты класса, описанные со спецификатором private?
4. Каким образом наследуются компоненты класса, описанные со спецификатором protected?
5. Каким образом описывается производный класс?
6. Наследуются ли конструкторы?
7. Наследуются ли деструкторы?
8. В каком порядке конструируются объекты производных классов?
9. В каком порядке уничтожаются объекты производных классов?
10. Что представляют собой виртуальные функции и механизм позднего связывания?
11. Могут ли быть виртуальными конструкторы? Деструкторы?
12. Наследуется ли спецификатор virtual?
13. Какое отношение устанавливает между классами открытое наследование?
14. Какое отношение устанавливает между классами закрытое наследование?
15. В чем заключается принцип подстановки?
16. Имеется иерархия классов:

```
class Student
{
        int age;
public:
        string name;
        ...
};
class Employee : public Student
{
protected:
        string post;
        ...
};
class Teacher : public Employee
{
protected: int stage;
...
};
```

Teacher x;
Какие компонентные данные будет иметь объект x?

17. Для классов Student, Employee и Teacher написать конструкторы без параметров.
18. Для классов Student, Employee и Teacher написать конструкторы с параметрами.
19. Для классов Student, Employee и Teacher написать конструкторы копирования.
20. Для классов Student, Employee и Teacher определить операцию присваивания.

7. Содержание отчета

1) Постановка задачи (общая и конкретного варианта).
2) Описание класса.
3) Определение компонентных функций.
4) Определение глобальных функций.

5) Функция main().
6) Объяснение результатов работы программы.
7) Ответы на контрольные вопросы.