**Лабораторная работа №5**

**ЗАДАНИЕ 1**

*#include <iostream>*

*class BASE {*

*public:*

*int i;*

*protected:*

*long l;*

*private:*

*double d;*

*public:*

*BASE(int \_i, long \_l, double \_d) : i(\_i), l(\_l), d(\_d) {}*

*void PrintValues() {*

*std::cout << "Значения полей: i = " << i << ", l = " << l << ", d = " << d << std::endl;*

*}*

*};*

*int main() {*

*BASE baseObject(10, 1000, 3.14);*

*baseObject.PrintValues();*

*// Попытка доступа к закрытому полю d вызовет ошибку компиляции*

*// std::cout << "Попытка доступа к полю d: " << baseObject.d << std::endl;*

*return 0;*

*}*

***Сходства:***

И **private**, и **protected** члены класса доступны только изнутри самого класса. Вне класса они недоступны напрямую.

***Различия:***

Область видимости:

**private**: Поле или метод, объявленное как **private**, видно только внутри самого класса. Оно недоступно в производных классах.

**protected**: Поле или метод, объявленное как **protected**, видно внутри самого класса, а также в производных классах.

Уровень доступа:

**private**: Члены с уровнем доступа **private** скрываются от внешнего мира, и к ним можно обратиться только через публичные методы класса.

**protected**: Члены с уровнем доступа **protected** доступны внутри класса и в производных классах, что позволяет наследникам использовать их.

Использование:

**private** используется, чтобы скрыть детали реализации класса от внешнего мира и обеспечить инкапсуляцию.

**protected** используется, когда необходимо предоставить доступ производным классам к некоторым членам базового класса, но оставить их скрытыми от внешнего мира.

В данном примере поле d объявлено как **private**, что делает его недоступным за пределами класса BASE. Поле l объявлено как **protected**, поэтому оно доступно в производных классах, но также скрыто от внешнего мира.

**ЗАДАНИЕ 2**

*#include <iostream>*

*class BASE {*

*public:*

*int i;*

*protected:*

*long l;*

*private:*

*double d;*

*public:*

*BASE(int \_i, long \_l, double \_d) : i(\_i), l(\_l), d(\_d) {}*

*void PrintValues() {*

*std::cout << "Значения полей: i = " << i << ", l = " << l << ", d = " << d << std::endl;*

*}*

*};*

*class DERIVED : public BASE {*

*private:*

*float f;*

*public:*

*// Конструктор без параметров*

*DERIVED() : BASE(0, 0, 0), f(0.0f) {}*

*// Конструктор с 4 параметрами*

*DERIVED(int \_i, long \_l, double \_d, float \_f) : BASE(\_i, \_l, \_d), f(\_f) {}*

*};*

*int main() {*

*BASE baseObject(10, 1000, 3.14);*

*baseObject.PrintValues();*

*// Попытка доступа к закрытому полю d вызовет ошибку компиляции*

*// std::cout << "Попытка доступа к полю d: " << baseObject.d << std::endl;*

*return 0;*

*}*

В этом коде:

1. Мы объявляем класс **DERIVED**, который является производным от класса **BASE**, используя наследование с модификатором доступа **public**. Это означает, что все открытые (**public**) и защищенные (**protected**) члены класса BASE остаются таковыми в классе **DERIVED**.
2. В классе **DERIVED** мы добавляем свое собственное приватное поле **float f**, как было указано в задании.
3. Мы определяем два конструктора для класса **DERIVED**:

* Конструктор без параметров инициализирует все поля базового класса **BASE** значением по умолчанию (0) и поле f значением по умолчанию (0.0f).
* Конструктор с 4 параметрами позволяет инициализировать все поля базового класса **BASE** и поле f с заданными значениями.

Теперь у вас есть класс **DERIVED**, который наследует функциональность класса **BASE** и имеет дополнительное поле f. Вы можете создавать объекты класса **DERIVED** с использованием этих конструкторов и работать с ними как с объектами базового класса.

**ЗАДАНИЕ 3**

*#include <iostream>*

*class BASE {*

*public:*

*int i;*

*protected:*

*long l;*

*private:*

*double d;*

*public:*

*BASE(int \_i, long \_l, double \_d) : i(\_i), l(\_l), d(\_d) {}*

*void PrintValues() {*

*std::cout << "Значения полей: i = " << i << ", l = " << l << ", d = " << d << std::endl;*

*}*

*};*

*class DERIVED : public BASE {*

*private:*

*float f;*

*public:*

*// Конструктор без параметров*

*DERIVED() : BASE(0, 0, 0), f(0.0f) {}*

*// Конструктор с 4 параметрами*

*DERIVED(int \_i, long \_l, double \_d, float \_f) : BASE(\_i, \_l, \_d), f(\_f) {}*

*};*

*int main() {*

*BASE baseObject(10, 1000, 3.14);*

*baseObject.PrintValues();*

*// Создаем объект класса DERIVED*

*DERIVED derivedObject(20, 2000, 6.28, 1.0);*

*std::cout << "Размер класса BASE: " << sizeof(BASE) << " байт" << std::endl;*

*std::cout << "Размер класса DERIVED: " << sizeof(DERIVED) << " байт" << std::endl;*

*return 0;*

*}*

В результате выполнения этой программы вы получите размеры классов BASE и DERIVED. Размер класса DERIVED будет больше или равен размеру класса BASE, так как DERIVED наследует все поля BASE и добавляет свое поле **float f**.

**ЗАДАНИЕ 4**

Объект класса **DERIVED** был инициализирован следующим образом:

*DERIVED derivedObject(20, 2000, 6.28, 1.0);*

Это означает, что сначала инициализируется базовый класс **BASE** через его конструктор, который принимает три аргумента (int, long, double), а затем инициализируется свое собственное поле **f**.

В конструкторе класса **DERIVED** мы можем непосредственно инициализировать только те поля, которые собственные для класса **DERIVED** или те поля класса **BASE**, которые унаследованы с публичным или защищенным модификатором доступа. Непосредственная инициализация приватных полей базового класса **BASE** в производном классе **DERIVED** недопустима, поскольку они недоступны из производного класса.

Смотрите на конструктор класса **DERIVED**:

*// Конструктор с 4 параметрами*

*DERIVED(int \_i, long \_l, double \_d, float \_f) : BASE(\_i, \_l, \_d), f(\_f) {}*

Здесь мы инциализируем поля **i,** l, и d базового класса **BASE**, а также поле **f** класса **DERIVED**. Инициализация полей i и d базового класса выполняется с помощью конструктора класса **BASE**, так как они объявлены как приватное поле **d** и защищенное поле **l.** Поле **i** можно было бы инициализировать и непосредственно в конструкторе класса **DERIVED**, так как оно объявлено как публичное.

В общем, приватные и защищенные поля базового класса обязательно нужно инициализировать с помощью конструктора класса **BASE**, в то время как публичные поля могут быть инициализированы как в конструкторе класса **BASE**, так и непосредственно в конструкторе производного класса.

**ЗАДАНИЕ 5**

В начале следует отметить, что поля l и d в базовом классе BASE являются приватными и защищенными соответственно, поэтому производный класс DERIVED не имеет к ним прямого доступа. Для получения доступа к этим полям можно реализовать методы get в классе BASE.

Также, чтобы перегрузить оператор вывода << для объектов класса DERIVED, можно добавить соответствующий оператор внутрь класса или сделать его дружественной функцией.

*#include <iostream>*

*class BASE {*

*public:*

*int i;*

*protected:*

*long l;*

*private:*

*double d;*

*public:*

*BASE(int \_i, long \_l, double \_d) : i(\_i), l(\_l), d(\_d) {}*

*long getL() const {*

*return l;*

*}*

*double getD() const {*

*return d;*

*}*

*void PrintValues() {*

*std::cout << "Значения полей: i = " << i << ", l = " << l << ", d = " << d << "\n";*

*}*

*};*

*class DERIVED : public BASE {*

*private:*

*float f;*

*public:*

*DERIVED(int \_i, long \_l, double \_d, float \_f) : BASE(\_i, \_l, \_d), f(\_f) {}*

*friend std::ostream& operator<<(std::ostream& os, const DERIVED& obj) {*

*os << "i = " << obj.i << " " << &obj.i << "\n"*

*<< "l = " << obj.getL() << " " << &obj.l << "\n"*

*<< "d = " << obj.getD() << " Unable to get address of private member.\n"*

*<< "f = " << obj.f << " " << &obj.f << "\n";*

*return os;*

*}*

*};*

*int main() {*

*DERIVED derivedObject(20, 2000, 6.28, 1.0);*

*std::cout << derivedObject << "\n";*

*std::cout << "Размер класса BASE: " << sizeof(BASE) << " байт\n";*

*std::cout << "Размер класса DERIVED: " << sizeof(DERIVED) << " байт\n";*

*return 0;*

*}*

Обратите внимание, что у нас нет доступа к адресу приватного члена d, поэтому мы не можем его напечатать.

**ЗАДАНИЕ 6**

Сначала опишем класс Derived\_1, производный от класса Derived. Этот класс будет иметь конструктор, который принимает четыре аргумента, так как он унаследует четыре поля: i, l, d и f. Метод foo будет увеличивать значения i и l:

*class Derived\_1 : public DERIVED {*

*public:*

*Derived\_1(int \_i, long \_l, double \_d, float \_f) : DERIVED(\_i, \_l, \_d, \_f) {}*

*void foo() {*

*i++;*

*l+=1;*

*}*

*};*

Теперь, если мы поменяем тип наследования класса Derived от Base на private, то поля i и l, а также метод getL станут недоступны для классов, производных от класса Derived. Это вызовет ошибки компиляции в классе Derived\_1.

Один из способов решить эту проблему - восстановить уровень доступа к переменным на public в классе Derived.

*#include <iostream>*

*class BASE {*

*public:*

*int i;*

*protected:*

*long l;*

*private:*

*double d;*

*public:*

*BASE(int \_i, long \_l, double \_d) : i(\_i), l(\_l), d(\_d) {}*

*long getL() const {*

*return l;*

*}*

*double getD() const {*

*return d;*

*}*

*void PrintValues() {*

*std::cout << "Значения полей: i = " << i << ", l = " << l << ", d = " << d << "\n";*

*}*

*};*

*class DERIVED : private BASE {*

*private:*

*float f;*

*public:*

*DERIVED(int \_i, long \_l, double \_d, float \_f) : BASE(\_i, \_l, \_d), f(\_f) {}*

*using BASE::i;*

*using BASE::getL;*

*void incrementL() {*

*l += 1;*

*}*

*friend std::ostream& operator<<(std::ostream& os, const DERIVED& obj) {*

*os << "i = " << obj.i << " " << &obj.i << "\n"*

*<< "l = " << obj.getL() << " " << &obj.l << "\n"*

*<< "d = " << obj.getD() << " Unable to get address of private member.\n"*

*<< "f = " << obj.f << " " << &obj.f << "\n";*

*return os;*

*}*

*};*

*class Derived\_1 : public DERIVED {*

*public:*

*Derived\_1(int \_i, long \_l, double \_d, float \_f) : DERIVED(\_i, \_l, \_d, \_f) {}*

*void foo() {*

*i++;*

*}*

*};*

*int main() {*

*Derived\_1 derivedObject1(20, 2000, 6.28, 1.0);*

*derivedObject1.foo();*

*std::cout << "Размер класса BASE: " << sizeof(BASE) << " байт\n";*

*std::cout << "Размер класса DERIVED: " << sizeof(DERIVED) << " байт\n";*

*return 0;*

*}*

**ЗАДАНИЕ 7**

Ваши функции **ff** должны быть объявлены в **public** разделах классов **BASE** и **DERIVED**, чтобы быть доступными в классе **Derived\_1**. Также глобальную функцию **ff** можно вызвать из любого места в коде, поскольку она по определению доступна глобально

*#include <iostream>*

*void ff() {*

*std::cout << "Это глобальная функция ff.\n";*

*}*

*class BASE {*

*public:*

*int i;*

*protected:*

*long l;*

*private:*

*double d;*

*public:*

*BASE(int \_i, long \_l, double \_d) : i(\_i), l(\_l), d(\_d) {}*

*long getL() const { return l; }*

*double getD() const { return d; }*

*void PrintValues() {*

*std::cout << "Значения полей: i = " << i << ", l = " << l << ", d = " << d << "\n";*

*}*

*virtual void ff() {*

*std::cout << "Это функция ff класса BASE.\n";*

*}*

*};*

*class DERIVED : public BASE {*

*private:*

*float f;*

*public:*

*DERIVED(int \_i, long \_l, double \_d, float \_f) : BASE(\_i, \_l, \_d), f(\_f) {}*

*using BASE::i;*

*using BASE::getL;*

*void incrementL() { l += 1; }*

*friend std::ostream& operator<<(std::ostream& os, const DERIVED& obj) {*

*os << "i = " << obj.i << " " << &obj.i << "\n"*

*<< "l = " << obj.getL() << " " << &obj.l << "\n"*

*<< "d = " << obj.getD() << " Unable to get address of private member.\n"*

*<< "f = " << obj.f << " " << &obj.f << "\n";*

*return os;*

*}*

*void ff() override {*

*std::cout << "Это функция ff класса DERIVED.\n";*

*}*

*};*

*class Derived\_1 : public DERIVED {*

*public:*

*Derived\_1(int \_i, long \_l, double \_d, float \_f) : DERIVED(\_i, \_l, \_d, \_f) {}*

*void foo() {*

*i++;*

*::ff(); // вызывает глобальную функцию*

*BASE::ff(); // вызывает функцию из BASE*

*DERIVED::ff(); // вызывает функцию из DERIVED*

*}*

*};*

*int main() {*

*Derived\_1 derivedObject1(20, 2000, 6.28, 1.0);*

*derivedObject1.foo();*

*std::cout << "Размер класса BASE: " << sizeof(BASE) << " байт\n";*

*std::cout << "Размер класса DERIVED: " << sizeof(DERIVED) << " байт\n";*

*return 0;*

*}*

**ЗАДАНИЕ 8**

1. Если функция ff будет определена на глобальном уровне, в классах DERIVED, BASE и Derived\_1, то вызов ff() в функции foo класса Derived\_1 обращается к функции ff в Derived\_1.
2. Если функцию убрать из класса DERIVED и функция ff определена на глобальном уровне и в классах BASE и Derived\_1, то вызов ff() в функции foo класса Derived\_1 обращается к функции ff в классе BASE, так как она публична и виртуальна.
3. Если функцию ff убрать и из класса BASE, и класса DERIVED (так что функция ff определена только на глобальном уровне и в Derived\_1), то вызов ff() в функции Derived\_1:foo обращается к глобальной функции ff(). Это связано с тем, что в противном случае искомое имя функции уже не находится в классе Derived\_1 или его родителях, и поиск продолжается на глобальном уровне.