МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Методические указания
к лабораторным работам по дисциплине
для студентов дневной и заочной форм обучения направлений
09.03.02 – «Информационные системы и технологии»
09.03.03 – «Прикладная информатика»

В двух частях

Часть 1

Севастополь
СевГУ
2024

УДК:004.434:519.682.2(076.5)
ББК 32.973.2я73
О-294

Р е ц е н з е н т ы:
В. Ю. Карлусов - канд. техн. наук,
доцент кафедры «Информационные системы»
И. В. Кудрявченко - канд. техн. наук, доцент кафедры
«Радиоэлектронные системы и технологии»

Ответственный за выпуск:
И. П. Шумейко – канд. физ.-мат. наук, доцент, заведующий
кафедрой информационных систем

Составитель: Т. И. Сметанина

О-294 **Объектно-ориентированное программирование :** методические указания к выполнению лабораторных работ по дисциплине для студентов дневной и заочной форм обучения направлений подготовки 09.03.02 – «Информационные системы и технологии» и 09.03.03 – «Прикладная информатика». В 2-х частях. Ч. 1 / Севастопольский государственный университет ; сост. Т. И. Сметанина. – Севастополь : СевГУ, 2024. – 38 с. – Текст : электронный.

Целью методических указаний является обучение студентов основным принципам объектно-ориентированного программирования, навыкам разработки программ на языке С++.

УДК:004.434:519.682.2(076.5)
ББК 32.973.2я73

Рассмотрены и утверждены на заседании кафедры информационных систем, протокол № 08 от 16 апреля 2024 г.

Рассмотрены и рекомендованы к изданию на заседании Учёного совета Института информационных технологий, протокол № 05 от 18 апреля 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цели и задачи лабораторных работ

Основная цель выполнения лабораторных работ – приобретение практических навыков написания программ на объектно-ориентированном языке программирования – С++. В результате выполнения лабораторных работ студенты углубляют знания основных теоретических положений дисциплины «Объектно-ориентированное программирование», решая практические задачи на ЭВМ.

По окончанию курса студенты должны овладеть объектно-ориентированным подходом решения практических задач, овладеть инструментами, сопровождающими разработку программных систем с использованием этого подхода.

Выбор вариантов и график выполнения лабораторных работ

В лабораторных работах студент решает заданную индивидуальным вариантом задачу. Варианты заданий приведены в каждой лабораторной работе и уточняются преподавателем.

Лабораторная работа выполняется в два этапа. На первом этапе, выполняемом самостоятельно дома, студенту необходимо выполнить следующее:

- проработать по конспекту и рекомендованной литературе, приведенной в конце настоящих методических указаний, основные теоретические положения лабораторной работы и ответить на контрольные вопросы;
- разработать алгоритм решения задачи;
- оформить результаты первого этапа в виде заготовки отчета по выполнению лабораторной работы (студенты-заочники первый этап выполнения лабораторных работ оформляют в виде контрольной работы, которую сдают на кафедру до начала зачетно-экзаменационной сессии).

На втором этапе, выполняемом в лабораториях кафедры, студенту следует:

- написать программу на языке Си++, реализующую разработанный алгоритм;
- отладить программу;
- разработать и выполнить тестовые примеры (не менее двух);
- подготовить и защитить отчёт по лабораторной работе.

Студенты должны выполнять и защищать работы строго по графику.

График уточняется преподавателем, ведущим лабораторные занятия.

Требования к оформлению отчета

Отчёты по лабораторной работе оформляются каждым студентом индивидуально. В общем случае отчёт должен включать: название и номер лабораторной работы; цель работы; вариант задания и постановку задачи; разработку и описание алгоритма решения задачи; разработку и описание программы; выводы по работе; приложения. Содержание отчета приведено в методических указаниях к каждой лабораторной работе.

Постановка задачи представляет изложение содержания задачи, определение списка подзадач для её решения. На этом этапе должны быть полностью определены исходные данные, перечислены задачи по преобразованию и обработке входных данных, описаны выходные данные, дана характеристика результатов.

Разработка и описание программы включает описание вычислительного процесса на языке C++. Кроме текста программы, в отчёте предоставляется её пооператорное описание (комментарии в тексте программы), даётся характеристика конструкций языка, обеспечивающих решение задачи конкретной лабораторной работы.

В приложениях приводятся текст программы, результат выполнения тестовых примеров.

Лабораторная работа № 1
ИССЛЕДОВАНИЕ ОТЛИЧИЙ МЕЖДУ СТРУКТУРНЫМ И ОБЪЕКТНЫМ ПОДХОДОМ

1.1 Цель работы

Исследование основных средств определения класса, создания объектов класса. Исследование отличий структурного и объектного подходов.

1.2 Краткие теоретические сведения

1.2.1 Описание класса

Класс является абстрактным типом данных, определяемым пользователем, и представляет собой модель реального объекта в виде данных и функций для работы с ними. Данные, определенные в классе, называются полями (по аналогии с полями структуры), а функции класса – методами. Поля и методы называются элементами класса. Описание класса в первом приближении выглядит так:

```
class <имя> {
 [private: <описание скрытых элементов>]
 [protected: <описание защищенных элементов>]
 [public: <описание общедоступных элементов>]
};
```

Описание класса заканчивается точкой с запятой.

Спецификаторы доступа `public`, `protected` и `private` управляют видимостью элементов класса. Элементы, описанные после служебного слова private, видимы только внутри класса. Этот вид доступа принят в классе по умолчанию. Спецификатор `protected` говорит о том, что элементы класса доступны методам данного класса и классам, производным от него. Элементы класса, расположенные после спецификатора `public`, являются общедоступными и представляют собой интерфейс класса. Действие любого спецификатора распространяется до следующего спецификатора или до конца класса. Можно задавать несколько секций спецификаторов доступа, порядок их следования значения не имеет.

Поля класса:

– могут иметь любой тип, кроме типа этого же класса (но могут быть указателями или ссылками на этот класс);

– могут быть описаны с модификатором `const`, при этом они инициализируются только один раз (с помощью конструктора) и не могут изменяться;

– могут быть описаны с модификатором `static`, но не как `auto`, `extern` или `register`.

В качестве примера создадим класс `my_time`:

```
#include <iostream>
using namespace std;
class my_time {
    private:                    // поля
        int hour, min, sec;
                                // описание методов
    public:
        my_time() {hour=10; min=10; sec=10;}
        my_time(int x) { hour=min=sec=x;}
        my_time(int a,int b,int c) {
            hour=a; min=b; sec=c;
        }                       // объявление методов
        void out_my_time();
        bool dinner_time();
        ~my_time();
};
                                // описание методов
void my_time::out_my_time() {
    cout<<hour<<" "<<min<<' '<<sec;
}
bool my_time::dinner_time() {
    if (hour==12) return true; return false;
}
my_time::~my_time() {
    cout<<"деструктор my_time"<<endl;
}
                            // описание функции
void print( my_time &x) {
    cout<<"время = ";
    x.out_my_time();
    cout<<" - это время обеда? - ";
    if (x.dinner_time()==true) cout<<"да"<<endl;
    else cout<<"нет"<<endl;
}

int main() {                    // создание объектов
    my_time ob1, ob2(5), ob3(12,13,14);
                                // вызов методов
    ob1.out_my_time();   cout<<endl;
    ob2.out_my_time();   cout<<endl;
    ob3.out_my_time();   cout<<endl;
    print(ob1);
    print(ob3);
                            // создание указателей
    my_time *ob4 = new my_time();
```

```
    my_time *ob5 = new my_time(7);
    ob5->out_my_time();  cout<<endl;
                        // создание ссылки
    my_time &ob6 = ob3;
    ob6.out_my_time();   cout<<endl;
    delete ob4;
    delete ob5;
}
```

Для этого требуется задать его свойства (поля: часы, минуты, секунды) и поведение (методы).

В этом классе описаны три скрытых целочисленных поля – `hour, min, sec`.

Все методы класса имеют непосредственный доступ к его скрытым полям, иными словами, тела функций класса входят в область видимости `private` элементов класса.

В приведенном классе содержится три определения методов и три объявления. Если тело метода определено внутри описания класса, метод является встроенным (`inline`). Как правило, встроенными делают короткие методы. Если внутри описания класса записано только объявление (заголовок) метода, то сам метод должен быть определен в другом месте программы с помощью **операции расширения области видимости** (**::**), например:

```
void my_time::out_my_time() { /* тело метода*/}
```

В каждом классе есть хотя бы один метод, имя которого совпадает с именем класса, и у этого метода нет типа. Он называется конструктором и вызывается автоматически при создании объекта класса. Конструктор предназначен для инициализации объекта. Автоматический вызов конструктора позволяет избежать ошибок, связанных с использованием неинициализированных переменных.

1.2.2 Описание объектов

Конкретные переменные типа «класс» называются экземплярами класса, или **объектами**. Время жизни и видимость объектов зависит от вида и места их описания. Примеры:

```
my_time ob1;   // Объект класса my_time с параметрами по умолчанию
my_time ob2(5),  ob3(12,13,14); // Объекты класса my_time
 // с параметрами (с явной инициализацией)
my_time *ob4 = new my_time(); // Динамический объект
my_time &ob6 = ob3; // Ссылка на объект
```

При создании каждого объекта выделяется память, достаточная для хранения всех его полей, и автоматически вызывается конструктор, выполняющий их инициализацию. Методы класса не тиражируются. При выходе объекта из области видимости он уничтожается, то есть выделенная для полей память освобождается и автоматически вызывается деструктор.

Доступ к элементам объекта аналогичен доступу к полям структуры. Для этого используются операция «.» (точка) при обращении к элементу через имя объекта и операция «->» при обращении через указатель, например:

```
my_time ob1; ob1.out_my_time();
my_time *ob5 = new my_time(7);  ob5->out_my_time();
```

Обратиться таким образом можно только к элементам со спецификатором `public`. Получить или изменить значения элементов со спецификатором `private` можно только через обращение к соответствующим методам, либо невозможно вовсе, если таких методов в классе нет.

1.2.3 Конструкторы

Конструктор предназначен для инициализации объекта и вызывается автоматически при его создании. Ниже перечислены основные свойства конструкторов:

1. Конструктор не возвращает значение, даже типа `void`. Нельзя получить указатель на конструктор.

2. Конструкторы нельзя описывать с модификаторами `const`, `virtual` и `static`.

3. Конструктор, вызываемый без параметров, называется конструктором по умолчанию.

4. Класс может иметь несколько конструкторов с разными параметрами для разных видов инициализации (при этом используется механизм перегрузки). Параметры конструктора могут иметь любой тип, кроме этого же класса.

5. Можно задавать значения параметров по умолчанию. Их может содержать только один из конструкторов.

6. Если программист не указал ни одного конструктора, компилятор создает его автоматически. Такой конструктор вызывает конструкторы по умолчанию для полей класса и конструкторы по умолчанию базовых классов. В случае, когда класс содержит константы или ссылки, при попытке создания объекта класса будет выдана ошибка, поскольку их необходимо инициализировать конкретными значениями, а конструктор по умолчанию этого делать не умеет.

7. Конструкторы глобальных объектов вызываются до вызова функции `main`. Локальные объекты создаются, как только становится активной область их действия. Конструктор запускается и при создании временного объекта (например, при передаче объекта из функции).

8. Конструктор вызывается, если в программе встретилась какая-либо из синтаксических конструкций:

```
имя_класса имя_объекта [(список параметров)];
```

```
// Список параметров не должен быть пустым
имя_класса (список параметров);
// Создается объект без имени (список может быть пустым)
имя_класса имя_объекта = выражение;
// Создается объект без имени и копируется
```

Примеры:

```
my_time ob1, ob2(5);
my_time(15); // создается безымянный объект
my_time ob7 = my_time(15); // выдел.память под объект ob7
                           // в который копируется безымянный объект.
```

Объекты класса `my_time` можно инициализировать различными способами, требуемый конструктор будет вызван в зависимости от списка значений в скобках. При задании нескольких конструкторов следует соблюдать те же правила, что и при написании перегруженных функций – у компилятора должна быть возможность распознать нужный вариант.

Существует еще один способ инициализации полей в конструкторе. Например, конструктор:

```
my_time(int a,int b,int c) {  hour=a; min=b; sec=c; }
```

можно записать и так:

```
my_time(int a,int b,int c):hour(a),min(b),sec(c) {}
```

Поля перечисляются через запятую. Для каждого поля в скобках указывается инициализирующее значение, которое может быть выражением. Без этого способа не обойтись при инициализации полей-констант, полей-ссылок и полей-объектов. В последнем случае будет вызван конструктор, соответствующий указанным в скобках параметрам.

1.2.4 Конструктор копирования

Помимо стандартных конструкторов, часто используется конструктор копирования, который вызывается всякий раз, когда необходимо создать копию объекта.

При передаче и возвращении объекта в функцию в виде значения всегда создается его временная копия. Если объект является пользовательским, то вызывается конструктор копирования его класса.

Все конструкторы копирования имеют только один параметр – ссылку на объект своего класса. Разумно сделать эту ссылку постоянной, поскольку конструктор не должен изменять передаваемый ему объект. Например:

```
my_time (const my_time & theCopy);
```

В данном случае конструктор `my_time` принимает постоянную ссылку на объект класса `my_time`, ведь задачей конструктора является создание копии `theCopy`.

Стандартный конструктор копирования (создаваемый компилятором по умолчанию) просто копирует каждое поле переданного ему объекта в поля нового объекта. Такое копирование называется **поверхностным** (`shallow copy`). Хоть оно и подходит для большинства случаев, могут возникнуть серьезные проблемы, если в классе поля являются указателями на объекты в динамической памяти.

В результате поверхностного копирования создаются точные копии значений всех полей одного объекта в другом. Следовательно, указатели в объектах-копиях будут указывать на ту же область динамической памяти, что и оригинал.

Катастрофа случится тогда, когда любой из объектов прекратит свое существование. Как уже говорилось, задача деструктора состоит в освобождении памяти, занимаемой объектом. Если деструктор первого объекта освободит динамическую память, а указатель во втором объекте будет указывать на нее, то возникнет паразитный указатель, а работа программы не будет стабильной и предсказуемой.

Во избежание подобных проблем, необходимо вместо стандартного конструктора копирования использовать собственный, который будет осуществлять **глубокое копирование** с перемещением значений полей, находящихся в динамической памяти. Глубокое копирование переносит значения, находящиеся в динамической памяти, во вновь созданные участки.

1.2.5 Деструкторы

Деструктор – это особый вид метода, применяющийся чаще всего для освобождения памяти, занимаемой объектом. Деструктор вызывается автоматически, когда объект выходит из области видимости:

- для локальных объектов – при выходе из блока, в котором они объявлены;
- для глобальных – как часть процедуры выхода из `main`;
- для объектов, заданных через указатели, деструктор вызывается неявно при использовании операции `delete`.

Деструктор, например:

```
my_time::~my_time() {
     cout<<"деструктор my_time"<<endl;
}
```

Имя деструктора начинается с тильды (~), непосредственно за которой следует имя класса. Если деструктор явным образом не определен, компилятор автоматически создает пустой деструктор. Описывать в классе деструктор явным образом требуется в случае, когда объект содержит указатели на выделяемую динамически память – иначе при уничтожении объекта память, на которую ссыла-

лись его поля-указатели, не будет помечена как свободная. Указатель на деструктор определить нельзя.

Деструктор:

- не имеет аргументов и возвращаемого значения;
- не может быть объявлен как `const` или `static`;
- не наследуется;
- может быть виртуальным.

1.3 Порядок выполнения

1.3.1 В ходе самостоятельной подготовки изучить основы работы со структурами в языке C++. Также изучить основные средства языка C++ для определения класса, создания объектов класса, работы с такими объектами и их уничтожения после использования.

1.3.2 Разработать две программы на языке C++ согласно своему варианту. Работа программ должна быть реализована в виде меню со следующими пунктами:

- чтение данных с клавиатуры;
- вывод данных на дисплей;
- обработка данных;
- выход.

1.3.3 Разработать тестовые примеры.

1.3.4 Выполнить отладку программ.

1.3.5 Сформулировать выводы, проанализировав созданные программы (сравнить структурный и объектный подходы).

1.3.6 Оформить отчет по проделанной работе.

1.4 Варианты заданий

В варианте задания дано описание типа данных и способа обработки этих данных. Требуется написать две программы.

В первой программе описать **структуру** (`struct`) данных на языке C++ и оформить следующие действия как функции для работы с описанной структурой:

- ввод с клавиатуры данных в переменную структуры;
- вывод на дисплей введенных данных;
- обработку всех введенных данных по заданному вариантом способу и вывод на дисплей результата обработки.

Для демонстрации работы функций создать **две переменные** заданного типа и осуществить вызов функций.

Во второй программе, написанной на базе первой, вместо структуры использовать **класс**, поля которого совпадают с полями структуры, а действия, выполняемые функциями в первой программе, выполняют методы класса.

В результате обе программы должны работать одинаково.

Интерфейс пользователя программы оформить в виде меню.

Вариант 1

Описать тип данных с именем `clerk`, содержащий следующие поля: порядковый номер, фамилия, имя, должность, стаж, подразделение, оклад.

Способ обработки – вывод на дисплей фамилии служащего, если он работает в должности, название которой задает пользователь, иначе вывести соответствующее сообщение.

Вариант 2

Описать тип данных с именем `product`, содержащий следующие поля: порядковый номер, название продукта, дата изготовления, срок годности (количество суток), производитель, цена.

Способ обработки – вывод на дисплей названия продукта, если он изготовлен производителем, название которого задает пользователь, иначе вывести соответствующее сообщение.

Вариант 3

Описать тип данных с именем `student`, содержащий следующие поля: порядковый номер, фамилия, имя, пол, рост, группа.

Способ обработки – вывод на дисплей фамилии и имени студента, если он учится в группе, название которой задает пользователь, иначе вывести соответствующее сообщение.

Вариант 4

Описать тип данных с именем `man`, содержащий следующие поля: порядковый номер, фамилия, имя, пол, рост, вес, возраст.

Способ обработки – вывод на дисплей фамилии и имени человека, если его вес выше значения, заданного пользователем, иначе вывести соответствующее сообщение.

Вариант 5

Описать тип данных с именем `books`, содержащий следующие поля: автор книги, название книги, издательство, жанр, дата поступления, стоимость книги.

Способ обработки – вывод на дисплей названия книги, если она напечатана в издательстве, название которого задает пользователь, иначе вывести соответствующее сообщение.

Вариант 6

Описать тип данных с именем `hostel`, содержащий следующие поля: номер зачетной книжки, ФИО студента, направление, группа, № общежития, № комнаты, долг за общежитие.

Способ обработки – вывод на дисплей ФИО студента, если его долг за общежитие превышает значение, заданное пользователем, иначе вывести соответствующее сообщение.

Вариант 7

Описать тип данных с именем `route`, содержащий следующие поля: номер маршрута, начальный пункт, конечный пункт, стоимость проезда, время в пути.

Способ обработки – вывод на дисплей номера маршрута, если название начального и конечного пунктов совпадают, иначе вывести соответствующее сообщение.

Вариант 8

Описать тип данных с именем `bank_client`, содержащий следующие поля: порядковый номер, фамилия, имя, отчество, город, улица, дом, квартира.

Способ обработки – вывод на дисплей фамилии и имени клиента, если номер дома и номер квартиры совпадают, иначе вывести соответствующее сообщение.

Вариант 9

Описать тип данных с именем `stipends`, содержащий следующие поля: номер зачетной книжки, фамилия, имя, отчество, номер группы, курс, средний балл за последнюю сессию, размер стипендии.

Способ обработки – вывод на дисплей фамилии и имени студента, если размер его стипендии выше значения, заданного пользователем, иначе вывести соответствующее сообщение.

Вариант 10

Описать тип данных с именем `clerk`, содержащий следующие поля: порядковый номер, фамилия, имя, должность, стаж, подразделение, оклад.

Способ обработки – вывод на дисплей фамилии служащего, если его оклад меньше значения, заданного пользователем, иначе вывести соответствующее сообщение.

Вариант 11

Описать тип данных с именем `books`, содержащий следующие поля: автор книги, название книги, издательство, жанр, дата поступления, стоимость книги.

Способ обработки – вывод на дисплей название книги, если она издана в жанре, название которого задает пользователь, иначе вывести соответствующее сообщение.

Вариант 12

Описать тип данных с именем `student`, содержащий следующие поля: порядковый номер, фамилия, имя, пол, рост, группа.

Способ обработки – вывод на дисплей фамилии и имени студентки, если ее рост ниже значения, заданного пользователем, иначе вывести соответствующее сообщение.

Вариант 13

Описать тип данных с именем `product`, содержащий следующие поля: порядковый номер, название продукта, дата изготовления, срок годности (количество суток), производитель, цена.

Способ обработки – вывод на дисплей названия продукта, при условии, что его цена не дороже значения, заданного пользователем, иначе вывести соответствующее сообщение.

Вариант 14

Описать тип данных с именем `clerk`, содержащий следующие поля: порядковый номер, фамилия, имя, должность, стаж, подразделение, оклад.

Способ обработки – вывод на дисплей фамилии служащего, если он работает в подразделении, название которого задает пользователь, иначе вывести соответствующее сообщение.

Вариант 15

Описать тип данных с именем `bank_client`, содержащий следующие поля: порядковый номер, фамилия, имя, отчество, город, улица, дом, квартира.

Способ обработки – вывод на дисплей фамилии и имени клиента, если номер дома больше значения, заданного пользователем, иначе вывести соответствующее сообщение.

Вариант 16

Описать тип данных с именем `hostel`, содержащий следующие поля: номер зачетной книжки, ФИО студента, направление, группа, № общежития, № комнаты, долг за общежитие.

Способ обработки – вывод на дисплей ФИО студента, если он учится по направлению, название которого вводит пользователь, иначе вывести соответствующее сообщение.

Вариант 17

Описать тип данных с именем `route`, содержащий следующие поля: номер маршрута, начальный пункт, конечный пункт, стоимость проезда, время в пути.

Способ обработки – вывод на дисплей номера маршрута, если стоимость проезда выше значения, заданного пользователем, иначе вывести соответствующее сообщение.

Вариант 18

Описать тип данных с именем `stipends`, содержащий следующие поля: номер зачетной книжки, фамилия, имя, отчество, номер группы, курс, средний балл за последнюю сессию, размер стипендии.

Способ обработки – вывод на дисплей фамилии и имени студента, если средний балл за последнюю сессию выше 3.0, но ниже 4.5, иначе вывести соответствующее сообщение.

Вариант 19

Описать тип данных с именем `clerk`, содержащий следующие поля: порядковый номер, фамилия, имя, должность, стаж, подразделение, оклад.

Способ обработки – вывод на дисплей фамилии служащего, если стаж работы ниже значения, заданного пользователем, иначе вывести соответствующее сообщение.

Вариант 20

Описать тип данных с именем `books`, содержащий следующие поля: автор книги, название книги, издательство, жанр, дата поступления, стоимость книги.

Способ обработки – вывод на дисплей названия книги, если ее написал автор, чью фамилию вводит пользователь, иначе вывести соответствующее сообщение.

Вариант 21

Описать тип данных с именем `route`, содержащий следующие поля: номер маршрута, начальный пункт, конечный пункт, стоимость проезда, время в пути.

Способ обработки – вывод на дисплей номера маршрута, если время в пути меньше значения, заданного пользователем, иначе вывести соответствующее сообщение.

Вариант 22

Описать тип данных с именем `bank_client`, содержащий следующие поля: порядковый номер, фамилия, имя, отчество, город, улица, дом, квартира.

Способ обработки – вывод на дисплей фамилии и имени клиента, если он проживает в городе, название которого задает пользователь, иначе вывести соответствующее сообщение.

Вариант 23

Описать тип данных с именем `books`, содержащий следующие поля: автор книги, название книги, издательство, жанр, дата поступления, стоимость книги.

Способ обработки – вывод на дисплей названия книги, если её стоимость выше значения, заданного пользователем, иначе вывести соответствующее сообщение.

Вариант 24

Описать тип данных с именем `route`, содержащий следующие поля: номер маршрута, начальный пункт, конечный пункт, стоимость проезда, время в пути.

Способ обработки – вывод на дисплей номера маршрута, если он прибывает в конечный пункт, название которого вводит пользователь, иначе вывести соответствующее сообщение.

Вариант 25

Описать тип данных с именем `hostel`, содержащий следующие поля: номер зачетной книжки, ФИО студента, направление, группа, № общежития, № комнаты, долг за общежитие.

Способ обработки – вывод на дисплей ФИО студента, если № комнаты и № общежития совпадают, иначе вывести соответствующее сообщение.

1.5 Содержание отчета

Титульный лист, наименование работы, цель работы, задание, тексты двух программ с комментариями, описание тестовых примеров для обеих программ и выводы по проделанной работе.

1.6 Контрольные вопросы

1.6.1 Объясните понятие «класс».

1.6.2 Объясните понятия «поле класса», «метод класса» и «элементы класса».

1.6.3 Объясните применение операции расширения области видимости.

1.6.4 Объясните понятие «инкапсуляция».

1.6.5 Поясните значение директив `public, protected и private`.

1.6.6 Опишите как осуществляется доступ к элементам класса.

1.6.7 Объясните понятие «конструктор класса» и его применение.

1.6.8 Объясните понятие «конструктор по умолчанию».

1.6.9 Перечислите свойства конструкторов.

1.6.10 Объясните понятие «конструктор копирования» и его применение.

1.6.11 Объясните понятие «деструктор класса».

1.6.12 Перечислите свойства деструкторов.

1.6.13 Объясните понятие «объект класса». Приведите пример.

1.6.14 Как создать указатель на объект класса, ссылку?

1.6.15 Как осуществляется допуск к полям и методам класса, если в программе создан объект, указатель на объект, ссылка на объект.

Лабораторная работа № 2
ИССЛЕДОВАНИЕ МЕХАНИЗМА ПРОСТОГО НАСЛЕДОВАНИЯ

2.1 Цель работы

Исследование основных средств создания базового и производного классов. Исследование особенностей вызова методов производного класса при простом наследовании.

2.2 Краткие теоретические сведения

2.2.1 Простое наследование

Язык C++ позволяет классу наследовать поля и методы одного или нескольких других классов. При этом новый класс называют **производным** классом (или классом-потомком, или подклассом). Класс, элементы которого наследуются, называется **базовым** классом (родительским классом или классом-предком, или суперклассом). Наследование дает возможность некоторые общие черты поведения классов абстрагировать (объединить) в одном базовом классе. Производные классы, наследуя это общее поведение, могут его несколько видоизменять, переопределяя некоторые методы базового класса, или дополнять, вводя новые элементы класса. Таким образом, определение производного класса значительно сокращается, поскольку нужно определить только отличающие его от других производных классов черты поведения.

Если у производного класса имеется всего один базовый класс, то говорят о **простом** (или одиночном) **наследовании**. Синтаксис объявления производного класса имеет следующий вид:

```
  class <имя_класса> : [<спецификатор_доступа>]
<имя_базового
    класса> {
    <тело класса> };
```

Спецификатор доступа – это одно из ключевых слов `public`, `protected` или `private`. Он не является обязательным и по умолчанию принимает значение `private` для классов и `public` для структур. Рассмотрим пример:

```
#include <iostream>
using namespace std;
class base {      // Базовый класс
    private:  int x;
    public:
        base();
        ~base();
        void f();
```

```
};
class derived: public base { // Производный класс
};
int main() {
    derived ob1;      // Создание объекта класса-наследника
    ob1.f();          // Обращение к методу базового класса
      derived *ob2=new derived(); // Создание указателя и объекта
      ob2->f();                   // Обращение к методу базового класса
      delete ob2;
}
    // Описание методов базового класса
base::base() { x=11; cout<<"constr_base\n";}
base::~base() { cout<<"destr_base\n";}
void base::f(){ cout<<"x="<<x<<endl;}
```

Спецификатор доступа при наследовании определяет уровень доступа к элементам базового класса, который получают элементы производного класса. В приведенной ниже таблице описаны возможные варианты наследуемого доступа.

Таблица 2.1 – Распространение спецификаторов доступа в рамках иерархии наследования

Доступ в базовом классе	Спецификатор наследуемого доступа	Доступ в производном классе
public protected private	public	public protected нет доступа
public protected private	protected	protected protected нет доступа
public protected private	private	private private нет доступа

Можно сделать вывод, что спецификатор наследуемого доступа устанавливает тот уровень доступа, до которого понижается уровень доступа к элементам, установленный в базовом классе.

Следует отметить, что не все элементы класса будут наследоваться. Не подлежат наследованию следующие элементы класса:

- конструкторы;
- конструкторы копирования;
- деструкторы;
- операторы присваивания, определенные программистом;
- друзья класса.

2.2.2 Переопределение функций

В производном классе обычно добавляются новые поля и методы, не существующие в базовом классе. Однако существует также возможность переопределения методов базового класса. Чтобы переопределить метод базового класса в производном классе, достаточно включить его прототип в объявление этого класса и затем дать ему определение. Конечно, прототипы переопределяемой функции в базовом и производном классах должны совпадать. Рассмотрим пример:

```
#include<iostream>
using namespace std;
class base {                        // Базовый класс
    private:     int x;
    public:      base();
                 base(int _x);
                 ~base();
                 void f();
};
class derived: public base { // Производный класс
    private:     int y;
    public:      derived();
                 derived(int x, int y);
                 ~derived();
                 void f();
};
int main() {
   derived ob1, ob2(4,5);// Создаем объект производного класса
   ob1.f();              // Обращение к методу производного класса
   ob1.base::f();        // Обращение к методу базового класса
   ob2.f();              // Обращение к методу производного класса
   ob2.base::f();        // Обращение к методу базового класса
}
     // Описание методов базового класса
base::base() { x=1; cout<<"constr_base\n";}
base::base(int _x) { x=_x; cout<<"constr_base\n";}
base::~base()  { cout<<"destr_base\n";}
void base::f() { cout<<"x="<<x<<endl;}
     // Описание методов производного класса
derived::derived():base() {
         y=2; cout<<"constr_derived\n";}
derived::derived(int _x, int _y):base(_x) {
             y=_y;  cout<<"constr_derived\n";}
```

```
derived::~derived() { cout<<"destr_derived\n";}
void derived::f() { cout<<"y="<<y<<endl;}
```

В этом случае метод `f()` переопределен в производном классе. Для вызова переопределенного метода используют операцию **расширения области видимости**:

<имя_класса> :: <имя_функции>

В примере:

`ob1.f();` – происходит обращение к методу класса `derived`

`ob1.base::f();` – происходит обращение к методу класса `base`

2.2.3 Конструкторы и деструкторы

Конструкторы не наследуются, поэтому производный класс либо должен объявить свой конструктор, либо предоставить возможность компилятору генерировать конструктор по умолчанию. Поскольку производный класс должен унаследовать все элементы родительского, при построении объекта своего класса он должен обеспечить инициализацию унаследованных полей, причем она должна быть выполнена до инициализации полей производного класса, так как последние могут использовать значения первых. В связи с этим в производном классе применяется следующая конструкция:

<имя класса>([<список аргументов>]) **:** <имя базового класса>([<список аргументов>])

То есть используется список инициализации элементов, в котором указывается конструктор базового класса. Часть параметров, переданных конструктору производного класса, обычно используется в качестве аргументов конструктора базового класса. Затем в теле конструктора производного класса выполняется инициализация полей, принадлежащих собственно этому классу.

В приведенном выше примере эта конструкция использована дважды. Описание первого конструктора:

```
derived::derived():base(){
                        y=2;
                        cout<<"constr_derived\n";}
```

Создание объекта: `derived ob1;`

При создании объекта класса `derived` вызывается конструктор класса `derived`, он в свою очередь вызывает конструктор базового класса `base`. В конструкторе базового класса происходит инициализация приватного поля `x`, далее управление передается в конструктор производного класса и происходит инициализация приватного поля `y`. Оба конструктора являются конструкторами по умолчанию и в результате поле `x` будет равно 1, поле `y` будет равно 2.

Описание второго конструктора:

```
derived::derived(int _x, int _y):base(_x) {    y=_y;
                          cout<<"constr_derived\n";}
```

Создание объекта: `derived ob2(4,5);`

При создании объекта класса `derived` вызывается конструктор класса `derived`, он в свою очередь вызывает конструктор базового класса `base`. В конструкторе базового класса происходит инициализация приватного поля `x`, далее управление передается в конструктор производного класса и происходит инициализация приватного поля `y`. Оба конструктора с параметрами, поле `x` будет равно 4, поле `y` будет равно 5.

В отношении деструкторов производных классов также действуют определенные правила. Деструктор производного класса должен выполняться раньше деструктора базового класса (иначе деструктор базового класса мог бы разрушить поля, которые используются и в производном классе). Когда деструктор производного класса выполнит свою часть работы по уничтожению объекта, вызывается деструктор базового класса. Всю работу по организации очередности соответствующих вызовов выполняет компилятор, программист не должен заботиться об этом.

2.3 Порядок выполнения

2.3.1 В ходе самостоятельной подготовки изучить средства описания классов при простом наследовании на языке C++.

2.3.2 Разработать программу на языке C++, в которой необходимо описать иерархию, состоящую из трех классов, согласно рисунку 2.1. Программа должна продемонстрировать корректную работу механизма простого наследования.

2.3.3 Разработать тестовые примеры.

2.3.4 Выполнить отладку программы.

2.3.5 Сформулировать выводы.

2.3.6 Оформить отчет по проделанной работе.

2.4 Варианты заданий

Описать иерархию классов, состоящую из трёх уровней (рис. 2.1).

Каждый уровень реализовать как один C++ класс, используя средства языка для организации иерархии.

Первый уровень – базовый класс, минимум с одним информационным полем. Второй и третий уровень – производные классы, минимум с двумя информационными полями.

Для каждого класса необходимо определить по два конструктора (с параметрами и без), а также методы ввода и вывода данных и метод, переопределяемый в производных классах (с одинаковым названием и разной реализацией).

Рисунок 2.1 – Вид иерархии

В основной программе необходимо создать 2 объекта производного класса третьего уровня (вызов конструктора по умолчанию и вызов конструктора с параметрами), осуществить обращение к доступным элементам иерархии (обязательно осуществить **вызов переопределенного метода и вызов этого же метода у классов первого и второго уровней)**.

Результаты работы вывести на экран и внести в отчет.

Вариант 1

Класс первого уровня – Транспорт; второго – Автомобиль; третьего – Легковой.

Вариант 2

Класс первого уровня – Млекопитающие; второго – Сумчатые; третьего – Кенгуру.

Вариант 3

Класс первого уровня – Теплокровные позвоночные животные; второго – Семейство кошачьих; третьего – Пантера.

Вариант 4

Класс первого уровня – Устройство; второго – Часы; третьего – Песочные.

Вариант 5

Класс первого уровня – Здание; второго – Жилое; третьего – Частный дом.

Вариант 6

Класс первого уровня – Геометрическая фигура; второго – Четырехугольник; третьего – Трапеция.

Вариант 7

Класс первого уровня – Транспортное средство; второго – Воздушное; третьего – Самолет.

Вариант 8

Класс первого уровня – Часы; второго – Механические; третьего – Секундомер.

Вариант 9

Класс первого уровня – Теплокровные позвоночные животные; второго – Птицы; третьего – Голубь.

Вариант 10

Класс первого уровня – Школа; второго – Средняя школа; третьего – 7 класс.

Вариант 11

Класс первого уровня – Животные; второго – Приматы; третьего – Шимпанзе.

Вариант 12

Класс первого уровня – Геометрическая фигура; второго – Многоугольник; третьего – Тетраэдр.

Вариант 13

Класс первого уровня – Млекопитающие; второго – Парнокопытные; третьего – Жираф.

Вариант 14

Класс первого уровня – Устройство; второго – Часы; третьего – Кварцевые.

Вариант 15

Класс первого уровня – Школа; второго – Начальная школа; третьего – 2 класс.

Вариант 16

Класс первого уровня – Млекопитающие; второго – Водоплавающие; третьего – Кит.

Вариант 17

Класс первого уровня – Нежилое здание; второго – Торговля; третьего – Маркет.

Вариант 18

Класс первого уровня – Животные Австралии; второго – Млекопитающие; третьего – Яйцекладущие.

Вариант 19

Класс первого уровня – Продукты питания; второго – Молокосодержащие; третьего – Творог.

Вариант 20

Класс первого уровня – Институт; второго – Форма обучения; третьего – Группа.

Вариант 21

Класс первого уровня – Транспорт; второго – Железнодорожный; третьего – Электричка.

Вариант 22

Класс первого уровня – Водоплавающие; второго – Костные рыбы; третьего – Осетровые.

Вариант 23

Класс первого уровня – Институт; второго – Кафедра; третьего – Специальность.

Вариант 24

Класс первого уровня – Тара; второго – Ящик; третьего – Ящик для бутылок.

Вариант 25

Класс первого уровня – Рыба; второго – Хрящевая рыба; третьего – Акула.

2.5 Содержание отчета

Титульный лист, наименование работы, цель работы, задание, текст программы с комментариями, описание тестовых примеров и выводы по проделанной работе.

2.6 Контрольные вопросы

2.6.1 Объясните понятие «простое наследование» и его применение.

2.6.2 Объясните понятия «базовый класс» и «производный класс».

2.6.3 Объясните понятие «косвенный базовый класс». Приведите пример.

2.6.4 Объясните понятие «спецификатор доступа» и его роль при наследовании.

2.6.5 Приведите пример переопределения функции и объясните суть механизма.

2.6.6 Опишите, как обратиться к методу базового класса, если он был переопределен в наследнике.

2.6.7 Перечислите элементы класса, которые можно наследовать и которые нельзя.

2.6.8 Перечислите особенности описания и поведения конструкторов при простом наследовании.

2.6.9 Перечислите особенности вызова деструкторов при простом наследовании.

Лабораторная работа № 3
ИССЛЕДОВАНИЕ МЕХАНИЗМА ВИРТУАЛЬНЫХ ФУНКЦИЙ

3.1 Цель работы

Исследование основных средств описания виртуальных функций и использование их при написании объектно-ориентированных программ.

3.2 Краткие теоретические сведения

Абстрактные классы используются в качестве обобщенных концепций, на основе которых можно создавать более конкретные производные классы. Невозможно создать объект абстрактного класса. Однако можно использовать указатели и ссылки на абстрактные типы классов. Работа с объектами чаще всего производится через указатели. Указателю на базовый класс можно присвоить значение адреса объекта любого производного класса. Такой указатель связывается со своим объектом только во время выполнения программы, то есть динамически.

Класс называется абстрактным, если в нем объявлена по крайней мере один чисто виртуальный метод. Виртуальный метод объявляется с помощью синтаксиса чистого описателя (= 0). Класс, производный от абстрактного класса, должен реализовать чисто виртуальные функции (описать тело метода), иначе производный класс останется абстрактным и создать объект такого класса будет невозможно.

3.2.1 Абстрактные классы

Класс, содержащий хотя бы один чисто виртуальный метод, называется **абстрактным**.

Чисто виртуальный метод — это метод класса, не имеющий описания. Чисто виртуальный метод содержит признак «=0» вместо тела, например:

```
virtual void func (int) = 0;
```

Чисто виртуальный метод должен переопределяться в производном классе (возможно, опять как чисто виртуальный).

Абстрактные классы предназначены для представления общих понятий, которые предполагается конкретизировать в производных классах. Абстрактный класс может использоваться **только в качестве базового** для других классов — объекты абстрактного класса создавать нельзя, поскольку прямой или косвенный вызов чисто виртуального метода приводит к ошибке при выполнении.

При определении абстрактного класса необходимо иметь в виду следующее:

1) абстрактный класс не может использоваться в качестве типа аргумента, передаваемого функции;

2) абстрактный класс не может использоваться в качестве типа возвращаемого значения функции;

3) нельзя осуществлять явное преобразование типа объекта к типу абстрактного класса;

4) нельзя объявить представителя абстрактного класса (создать объект);

5) можно объявить указатель или ссылку на абстрактный класс.

Таким образом, можно создать функцию, параметром которой является указатель на абстрактный класс. На место этого параметра при выполнении программы может передаваться указатель на объект любого производного класса. Это позволяет создавать **полиморфные функции**, работающие с объектом любого типа в пределах одной иерархии.

3.2.2 Виртуальные методы

Для определения **виртуального метода** используется спецификатор `virtual`.

Например, опишем иерархию: фигура → многоугольник → треугольник.

```
class figure {
    public:
        figure() {cout<<"constr figure"<<endl;}
        virtual        ~figure()        {cout<<"destr
figure"<<endl;}
        virtual void show()=0;
};
    // Класс figure – абстрактный, создать объект этого класса нельзя.
class polygonal: public figure {
        int n;       // количество углов
    public:
        polygonal() {
            n=4;
            cout<<"constr polygonal()"<<endl;
            }
        polygonal(int x):figure() {
            n=x;
            cout<<"constr polygonal(int)"<<endl;
            }
        ~polygonal() {cout<<"destr polygonal"<<endl;}
        void show() {cout<<"n="<<n<<endl;}
};
class triangle: public polygonal {
        float s;         // площадь
    public:
        triangle() {
            s=11.1;
```

```
            cout<<"constr triangle()"<<endl;  }
        triangle(int x, float y):polygonal(x) {
            s=y;
            cout<<"constr triangle(int,float)"<<endl;
            }
        ~triangle() {cout<<"destr triangle"<<endl;}
        void show() {cout<<"s="<<s<<endl;}
};
int main() {
    figure *ob;
    ob=new polygonal(5);
    ob->show();
    delete ob;
cout<<"----------"<<endl;
        ob=new triangle(3,12.5);
        ob->show();
        delete ob;
}
```

Работа с объектами чаще всего производится через указатели. Указателю на базовый класс можно присвоить значение адреса объекта любого производного класса, например:

`figure *ob;` // Описывается указатель на базовый класс

`ob=new polygonal(5);` // Указатель ссылается на объект производного класса `polygonal`.

Вызов методов объекта происходит в соответствии с типом объекта, на который указатель ссылается, а не в соответствии с типом указателя, поэтому при выполнении оператора `ob->show();` будет вызван метод класса `polygonal`, а не класса `figure`.

Для вызова метода класса `triangle` через тот же указатель `ob` сначала нужно освободить выделенную под `polygonal` память, а затем выделить её под `triangle`:

```
delete ob;
ob=new triangle(3,12.5);
ob->show(); // вызов метода класса triangle
```

В C++ реализован механизм **позднего связывания**, когда разрешение ссылок на метод происходит на этапе выполнения программы в зависимости от конкретного типа объекта, вызвавшего метод. Этот механизм реализован с помощью виртуальных методов.

Методы, у которых известен интерфейс вызова (то есть прототип), но реализация не может быть задана в общем случае, а может быть определена только для конкретных случаев, называются **виртуальными** (термин, означающий, что метод может быть переопределен в производном классе).

Механизм виртуальных функций гарантирует, что для объекта будет вызнана правильная функция независимо от того, какое выражение используется для осуществления вызова.

Правила описания и использования виртуальных методов:

1. Если в базовом классе метод определен как виртуальный, то метод, определенный в производном классе с тем же именем и набором параметров, автоматически становится виртуальным, а с отличающимся набором параметров – обычным.
2. Виртуальные методы наследуются, то есть, переопределять их в производном классе требуется только при необходимости задать отличающиеся действия. Права доступа при переопределении изменить нельзя.
3. Если виртуальный метод переопределен в производном классе, объекты этого класса могут получить доступ к методу базового класса с помощью операции расширения области видимости.
4. Виртуальный метод не может объявляться с модификатором static, но может быть объявлен как дружественный.
5. Если в классе вводится описание виртуального метода, он должен быть определен хотя бы как чисто виртуальный.

Итак, **виртуальным называется метод, ссылка на который разрешается на этапе выполнения программы** (перевод английского слова virtual – в данном значении всего-навсего «фактический», то есть ссылка разрешается по факту вызова).

3.2.3 Механизм позднего связывания

Для каждого класса (не объекта!), содержащего хотя бы один виртуальный метод, компилятор создает **таблицу виртуальных методов** (vtbl), в которой для каждого виртуального метода записан его адрес в памяти. Адреса методов содержатся в таблице в порядке их описания в классах. Адрес любого виртуального метода имеет в vtbl одно и то же смещение для каждого класса в пределах иерархии.

Каждый объект содержит скрытое дополнительное **поле ссылки** на vtbl, называемое vptr. Оно заполняется конструктором при создании объекта (для этого компилятор добавляет в начало тела конструктора соответствующие инструкции).

На этапе компиляции ссылки на виртуальные методы заменяются на обращения к vtbl через vptr объекта, а на этапе выполнения в момент обращения к методу его адрес выбирается из таблицы. Таким образом, вызов виртуального метода, в отличие от обычных методов и функций, выполняется через дополнительный этап получения адреса метода из таблицы. Это несколько замедляет выполнение программы.

Поскольку объекты с виртуальными функциями должны поддерживать и таблицу виртуальных функций, то их использование всегда ведет к некоторому повышению затрат памяти и снижению быстродействия программы. Если вы работаете с небольшим классом, который не собираетесь делать базовым для

других классов, то в этом случае нет никакого смысла в использовании виртуальных функций.

3.2.4 Виртуальный деструктор

Рекомендуется делать виртуальными деструкторы для того, чтобы гарантировать правильное освобождение памяти из-под динамического объекта, поскольку в этом случае в любой момент времени будет выбран деструктор, соответствующий фактическому типу объекта. Деструктор передает операции delete размер объекта, имеющий тип size_t. Если удаляемый объект является производным и в нем не определен виртуальный деструктор, передаваемый размер объекта может оказаться неправильным.

При удалении объекта производного класса, на который ссылается указатель базового класса, если деструктор объявлен как виртуальный, то будет вызван деструктор соответствующего производного класса. Затем деструктор производного класса вызовет деструктор базового класса, и объект будет правильно удален (удален целиком). В противном случае будет вызван только деструктор базового класса и произойдет утечка памяти. Рассмотрим пример:

```
class Base {
  int x;
  public:  Base(_x){x = _x;};
          ~Base() { /*Освобождение памяти */ }
};

class Derived: public Base {
 public:
    ~Derived() { /*Освобождение памяти */ }
};

int main () {
  Base* pBase = new Derived;
    //...
 delete pBase;
 return 0;
}
```

Так как объявленный в классе `Derived` деструктор не является виртуальным, инструкция «`delete pBase;`» вызовет удаление только памяти, принадлежащей классу `Base`, поскольку она будет квалифицирована как вызов `Base::~Base()`. Это может привести к утечке памяти. Если же сделать деструктор базового класса виртуальным, то и деструктор производного класса будет виртуальным. Значит использование предыдущей инструкции будет квалифицировано как вызов «`Derived :: ~ Derived();`» и удаление объекта произойдет правильно.

В связи с этим можно сформулировать следующие правила:

- используйте виртуальный деструктор, если в базовом классе имеются виртуальные функции;

- используйте виртуальные функции только в том случае, если программа содержит и базовый, и производный классы;

- не пытайтесь создать виртуальный конструктор.

Виртуальный механизм работает только при использовании указателей или ссылок на объекты. Объект, определенный через указатель или ссылку и содержащий виртуальные методы, называется **полиморфным.** В данном случае полиморфизм состоит в том, что с помощью одного и того же обращения к методу выполняются различные действия в зависимости от типа, на который ссылается указатель в каждый момент времени.

3.3 Порядок выполнения лабораторной работы

3.3.1 В ходе самостоятельной подготовки изучить основы работы с виртуальными функциями.

3.3.2 Разработать программу на языке C++, в которой необходимо продемонстрировать корректную работу механизма виртуальных функций. Работа программы должна быть реализована в виде меню.

3.3.3 Разработать тестовые примеры и выполнить отладку программы.

3.3.4 Получить результаты работы программы и исследовать её свойства для различных режимов работы, сформулировать выводы.

3.3.5 Оформить отчет по проделанной работе.

3.4 Варианты заданий

Для варианта задания из лабораторной работы № 2 необходимо выполнить следующее:

- Переопределить базовый класс как абстрактный – для этого один из методов определить чисто виртуальным, переопределить этот метод в производных классах.

- Создать указатель на базовый класс. Под указатель поместить объект производного класса. Проиллюстрировать работу программы с объектом через указатель: ввод/вывод данных, вызов методов класса.

- Проиллюстрировать корректную работу механизма виртуальных функций – вызвать виртуальную функцию и убедиться, что тип выполняемой функции соответствует типу объекта под указателем. Поменять объект под указателем на объект второго производного класса. Проверить корректную работу механизма.

- Описать деструкторы классов, содержащие тестовые фразы для вывода на экран. Отследить во время выполнения программы работу деструкторов по тестовым фразам – место вызова, порядок выполнения. Определить работу деструкторов с использованием виртуальных методов. Отследить изменения в работе программы.

Все выполненные действия отобразить в отчете с пояснениями.

3.5 Содержание отчета о выполнении лабораторной работы

Титульный лист, наименование работы, цель работы, задание, текст программы с комментариями, описание тестовых примеров и выводы по проделанной работе.

3.6 Контрольные вопросы

3.6.1 Объясните работу механизмов раннего и позднего связывания.

3.6.2 Объясните понятие «виртуальный метод» и его применение, перечислите основные правила по работе с виртуальными методами.

3.6.3 Опишите содержимое и принцип работы таблицы виртуальных методов.

3.6.4 Объясните необходимость создания в программе виртуальных деструкторов. Приведите пример.

3.6.5 Объясните понятие «полиморфизм». Перечислите существующие виды полиморфизма.

3.6.6 Объясните понятие «чисто виртуальная функция», приведите пример

3.6.7 Объясните понятие «абстрактный класс», приведите пример.

3.6.8 Перечислите правила работы с абстрактным классом.

Лабораторная работа № 4
ИССЛЕДОВАНИЕ МЕХАНИЗМА МНОЖЕСТВЕННОГО НАСЛЕДОВАНИЯ

4.1 Цель работы

Исследование основных средств описания производного класса, наследующего свойства нескольких базовых классов.

4.2 Краткие теоретические сведения

4.2.1 Множественное наследование

Если у производного класса имеется несколько базовых классов, то говорят о множественном наследовании. Множественное наследование позволяет сочетать в одном производном классе свойства и поведение нескольких классов.

```
#include <iostream>
using namespace std;
class base1 {
    private: int a1;
    protected:   int a2;
    public:
        base1() { a1=1; a2=2;
            cout<<"constr_base1"<<endl;}
        ~base1() { cout<<"destr_base1"<<endl;}
        void f1(){ cout<<"метод base1"<<endl;}
};
class base2 {
    private: int b1;
    protected:   int b2;
    public:
        base2() { b1=3;  b2=4;
            cout<<"constr_base2"<<endl;}
        ~base2() {
            cout<<"destr_base2"<<endl;}
        void f1(){cout<<"метод base2"<<endl;}
};
class derived: public base1, public base2 {
    public:
        int f() {return a2+b2;}
        derived () {cout<<"constr_derived"<<endl;}
        ~derived() {cout<<"destr_derived"<<endl;}
};
int main() {
    derived ob1; cout<<ob1.f()<<endl;
```

```
/*  derived *ob2=new derived();
    cout<<ob2->f()<<endl;
    delete ob2;*/
}
```

В этом примере класс `base1` и `base2` базовые классы, между собой никак не связанные, класс `derived` – производный класс, он наследует свойства двух классов: поле `a2` от класса `base1` и поле `b2` от класса `base2`.

В основной программе создается объект производного класса `ob1` и производится вызов метода `f()`, который возвращает сумму полей базовых классов `a2` и `b2`. Запустите программу и посмотрите, как происходит вызов конструкторов и деструкторов. Закомментируйте первую строку `main()`, а многострочный комментарий уберите и посмотрите результат выполнения программы, сравните с предыдущей версией.

При множественном наследовании, как и при простом, конструкторы базовых классов вызываются компилятором до вызова конструктора производного класса. Причем порядок объявления базовых классов при наследовании определяет и порядок вызова их конструкторов, которому должен соответствовать порядок следования конструкторов базовых классов в списке инициализации. Деструкторы вызываются в обратном порядке.

4.2.2 Устранение неоднозначностей

Если в двух базовых классах будут объявлены одинаковые по сигнатуре методы (или поля с одинаковым именем), то, когда потребуется вызвать данный метод в классе-наследнике, возникнет вопрос: какой из унаследованных методов при этом будет использован? Ведь методы, объявленные в базовых классах, имеют одинаковые имена и сигнатуры. В результате при компилировании программы возникнет неоднозначность, которую необходимо устранить, иначе компилятор вернет сообщение об ошибке.

Неоднозначность можно устранить явным обращением к необходимой функции:

```
ob1.base1::f1();  // обращение к методу класса base1
ob1.base2::f1();  // обращение к методу класса base2
ob2->base1::f1();  // обращение к методу класса base1
ob2->base2::f1();  // обращение к методу класса base2
```

В любом случае при возникновении подобной ситуации, когда необходимо сделать выбор между одноименными методами или полями различных базовых классов, следует явно указывать имя необходимого базового класса перед именем метода или переменной.

4.2.3 Проблемы множественного наследования

Хотя множественное наследование имеет ряд преимуществ по сравнению с одиночным, программисты неохотно используют его. Основная проблема состоит в том, что некоторые компиляторы C++ не поддерживают множественное наследование; это затрудняет отладку программы, тем более что все возможности, реализуемые этим методом, можно получить и без него. Также основным недостатком множественного наследования является ромбовидное наследование.

Рекомендуется использовать множественное наследование, когда новый класс нуждается в функциях и возможностях из более чем одного базового класса.

4.3 Порядок выполнения лабораторной работы

4.3.1 В ходе самостоятельной подготовки изучить средства описания классов при множественном наследовании на языке C++.

4.3.2 Разработать программу на языке C++, в которой необходимо продемонстрировать корректную работу механизма множественного наследования. Работа программы должна быть реализована в виде меню.

4.3.3 Разработать тестовые примеры.

4.3.4 Выполнить отладку программы.

4.3.5 Получить результаты работы программы и исследовать её свойства для различных режимов работы, сформулировать выводы.

4.3.6 Оформить отчет по проделанной работе.

4.4 Варианты заданий

Для производного класса, заданного вариантом, придумать два или более базовых класса. В базовых классах иерархии должны присутствовать минимум одно уникальное поле и один уникальный метод, а также метод с одинаковым названием, но различной реализацией. В производном классе должны присутствовать минимум два уникальных поля и один уникальный метод, а также выполнено переопределение метода базовых классов, имеющего одинаковое название.

Для иллюстрации корректной работы механизма множественного наследования необходимо в `main()` создать минимум два объекта производного класса (для вызова разных конструкторов); осуществить вызов уникальных методов всех классов иерархии; осуществить вызов методов с одинаковым названием, но различной реализацией.

Вариант 1

Производный класс: «Сфинкс».

Вариант 2

Производный класс: «Вампир».

Вариант 3

Производный класс: «Ихтиандр».

Вариант 4

Производный класс: «Энты».

Вариант 5

Производный класс: «Мантикора».

Вариант 6

Производный класс: «Дед Мороз».

Вариант 7

Производный класс: «Чеширский кот».

Вариант 8

Производный класс: «Снегурочка».

Вариант 9

Производный класс: «Кикимора».

Вариант 10

Производный класс: «Дарт Вейдер».

Вариант 11

Производный класс: «Химера».

Вариант 12

Производный класс: «Минотавр».

Вариант 13

Производный класс: «Леший».

Вариант 14

Производный класс: «Чебурашка».

Вариант 15

Производный класс: «Сирена».

Вариант 16

Производный класс: «Дриада».

Вариант 17

Производный класс: «Человек Паук».

Вариант 18

Производный класс: «Сатир».

Вариант 19

Производный класс: «Аликорн».

Вариант 20

Производный класс: «Горгона».

Вариант 21

Производный класс: «Русалка».

Вариант 22

Производный класс: «Змей Горыныч».

Вариант 23

Производный класс: «Бастет».

Вариант 24

Производный класс: «Джин».

Вариант 25

Производный класс: «Гарпии».

4.5 Содержание отчета о выполнении лабораторной работы

Титульный лист, наименование работы, цель работы, задание, текст программы с комментариями, описание тестовых примеров и выводы по проделанной работе.

4.6 Контрольные вопросы

4.6.1 Объясните понятие «множественное наследование» и его применение.

4.6.2 Перечислите особенности работы конструкторов при множественном наследовании. Приведите пример.

4.6.3 Перечислите особенности работы деструкторов при множественном наследовании. Приведите пример.

4.6.4 От чего зависит порядок вызова конструкторов базовых классов?

4.6.5 Поясните механизм возникновения неоднозначностей в программах с использованием множественного наследования, а также опишите пути решения данной проблемы. Приведите пример.

4.6.5 Перечислите достоинства и недостатки множественного наследования.

4.6.6 Объясните понятие «ромбовидное наследование», а также опишите пути решения данной проблемы. Приведите пример.

4.6.7 Объясните понятие «косвенный базовый класс». Приведите пример.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Зайцев, М. Г. Объектно-ориентированный анализ и программирование : учебное пособие / М. Г. Зайцев. — Новосибирск : НГТУ, 2017. — 84 с. — URL: https://e.lanbook.com/book/118271 (дата обращения: 17.04.2024). — Режим доступа: для авториз. пользователей. — ISBN 978-5-7782-3308-9. — Текст : электронный.

2. Мурлин, А. Г. Объектно-ориентированное программирование : учебное пособие / А. Г. Мурлин, В. А. Мурлина, М. В. Янаева. — Краснодар : КубГТУ, 2021. — 151 с. — URL: https://e.lanbook.com/book/231569 (дата обращения: 17.04.2024). — Режим доступа: для авториз. пользователей. — ISBN 978-5-8333-1059-5. — Текст : электронный.

3. Скворцова, Л. А. Объектно-ориентированное программирование на языке C++: Практикум : учебное пособие / Л. А. Скворцова, А. А. Бирюкова, К. В. Гусев. — Москва : РТУ МИРЭА, 2021. — 146 с. — URL: https://e.lanbook.com/book/176540 (дата обращения: 17.04.2024). — Режим доступа: для авториз. пользователей. — Текст : электронный // Лань : электронно-библиотечная система.

4. Скворцова, Л. А. Объектно-ориентированное программирование на языке C++ : учебное пособие / Л. А. Скворцова. — Москва : РТУ МИРЭА, 2020. — 246 с. — URL: https://e.lanbook.com/book/163862 (дата обращения: 17.04.2024). — Режим доступа: для авториз. пользователей. — Текст : электронный.

5. Теплоухов, С. В. Основы объектно-ориентированного программирования на языке C++ : учебное пособие / С. В. Теплоухов. — Майкоп : АГУ, 2021. — 92 с. — URL: https://e.lanbook.com/book/231416 (дата обращения: 17.04.2024). — Режим доступа: для авториз. пользователей. — Текст : электронный.

6. Юрина, Т. А. Объектно-ориентированное программирование : учебно-методическое пособие / Т. А. Юрина. — Омск : СибАДИ, 2023. — 72 с. — URL: https://e.lanbook.com/book/338576 (дата обращения: 17.04.2024). — Режим доступа: для авториз. пользователей. — Текст : электронный.

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Методические указания
к лабораторным работам

Часть 1

Составитель: **Сметанина** Татьяна Ивановна

В авторской редакции

Изд. № 60/2024. Объем 2,5 п.л. Усл. печ. л. 2,32. Уч.-изд. л. 2,45.
Издательский центр ФГАОУ ВО «Севастопольский государственный университет»