МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ УНИВЕРСИТЕТ
имени академика С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Классы и наследование

Электронные методические указания
к лабораторной работе № 3

Самара
2011

Составители: МЯСНИКОВ Евгений Валерьевич
ПОПОВ Артем Борисович

В лабораторной работе № 3 по дисциплине "Языки и методы программирования" изучаются синтаксис и особенностей реализации принципов объектно-ориентированного программирования на в C++. Приводятся краткие теоретические сведения, необходимые для выполнения лабораторных работ. Дан пример выполнения лабораторной работы.

Методические указания предназначены для студентов факультета информатики, направление 010400 – Прикладная математика и информатика, бакалавриат (010400.62)/магистратура (010400.68, магистерская программа – Технологии параллельного программирования и суперкомпьютинг).

Содержание

СОДЕРЖАНИЕ **3**

1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ЛАБОРАТОРНОЙ РАБОТЫ **4**

1.1 Классы 4

1.2 Конструкторы и деструкторы 6

1.3 Наследование 8

1.4 Виртуальные функции и абстрактные классы 11

2 ПРИМЕР ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ **14**

3 СОДЕРЖАНИЕ ОТЧЕТА **18**

4 КОНТРОЛЬНЫЕ ВОПРОСЫ **18**

5 ЗАДАНИЯ НА ЛАБОРАТОРНУЮ РАБОТУ **19**

5.1 Начальный уровень сложности 19

5.2 Средний уровень сложности 20

5.3 Высокий уровень сложности 22

6 БИБЛИОГРАФИЧЕСКИЙ СПИСОК **24**

Цель работы: Изучение синтаксиса и особенностей реализации принципов объектно-ориентированного программирования в C++. Получение навыков применения объектно-ориентированного подхода при решении задач.

1 Теоретические основы лабораторной работы

1.1 Классы

Класс в C++ представляет собой производный структурированный тип, содержащий совокупность данных и функций их обработки. Синтаксис описания класса выглядит следующим образом*:*

```
class [[<имя класса> [[: <список базовых классов>]] ]]
{
    <список членов класса>
} [[<список переменных>]];
```

Здесь <имя класса>- корректный идентификатор, <список базовых классов> - один или несколько базовых классов, от которых наследуется определяемый класс (более подробно об этом будет рассказано позднее), <список членов класса> - описывает поля и методы класса. Здесь же описываются дружественные классы, методы и функции, а также вложенные типы. Необязательный <список переменных> позволяет описать набор переменных, связанных с описываемым классом.

В C++ поля и методы классов часто называют «членами» классов: «данными-членами» и «функциями-членами» классов. В литературе также встречаются термины «компонентные данные» и «компонентные функции» классов.

Синтаксис при объявлении полей класса ничем не отличается от синтаксиса объявления полей структур. Объявление методов класса выполняется почти так же, как и объявление обычных функций. Разница заключается в используемых спецификаторах (**explicit**, **virtual**, **inline**, **const**, спецификатор чистоты «=0», исключений **throw**). Методы класса могут быть определены сразу же внутри класса. Однако, довольно часто при описании класса метод лишь объявляют, вынося его реализацию за пределы класса. При этом объявление и реализация могут содержаться в одном или разных файлах (например, объявление класса содержится в заголовочном *.h файле, а реализация в *.cpp файле). При реализации метода вне класса используется следующий синтаксис:

```
<тип> <имя класса>::<имя функции> (<список параметров>)
```

Методы класса так же, как и обычные функции могут быть перегруженными, иметь параметры по умолчанию или неопределенное число параметров.

Доступ к членам класса регулируется спецификаторами доступа, которых в C++ три: **private**, **protected** и **public**. Спецификаторы доступа указываются в списке членов класса следующим образом:

<спецификатор доступа>:

Действие спецификатора распространяется до следующего спецификатора или до конца описания класса. Собственно, значение спецификаторов следующее:

private – закрытый член класса, доступ к такому члену класса имеют только члены-функции того же класса, а также дружественные функции, методы и классы;

protected – защищенный член класса, доступ к такому члену класса имеют не только члены-функции того же класса и дружественные функции, методы и классы, но и члены-функции классов наследников;

public – открытый член класса, доступ к такому члену класса имеется всюду, где имеется доступ к самому классу.

По умолчанию, члены класса считаются private членами.

Рассмотрим теперь пример многомерного вектора вещественных чисел.

```
class CVector{
  private:
    double *m_pData;
    int     m_iSize;
  public:
    void Init( int size );
    void Uninit();
    double& Elm( int ind );
    int GetSize();
    void Add( const CVector &v );
    void Mul( double m );
};
```

Объявление объектов класса производится так же, как и объявление других переменных. Обращение к членам класса производится с использованием операций доступа «.» или «->». Первая из операций применяется к объектам, а вторая – к указателям на объекты:

```
CVector x, y;
x.Init(2);
y.Init(2);
x.Elm(0) = 1.1;     x.Elm(1) = 1.2;
y.Elm(0) = 2.1;     y.Elm(1) = 2.2;
```

```
x.Add(y);
CVector *px=&x;
px->Mul(3.3);
printf( “x=(%f; %f)”, x.Elm(0), px->Elm(1) );
x.Uninit();
y.Uninit();
```

Из методов самого класса обращаться к членам класса можно напрямую. В этом случае происходит обращение к данным или методам того объекта, относительно которого выполняется вызванный метод.

При выполнении той или иной функции-члена класса в ряде случаев требуется знать, для какого именно объекта выполняется метод. Для таких целей служит указатель **this**, доступный для всех нестатических методов класса. Этот указатель имеет тип:

```
<имя_класса>* const this;
```

1.2 Конструкторы и деструкторы

Конструкторы и деструкторы - это функции класса специального вида. **Конструкторы** служат для инициализации данных объекта и вызываются при создании объекта автоматически. Конструкторы имеют то же имя, что и сам класс, не имеют никакого выходного значения.

Объявление конструктора выглядит следующим образом:

```
<имя класса>(<параметры конструктора>);
```

Для класса вполне допустимо определить набор конструкторов с различными параметрами. В этом случае говорят о перегрузке конструкторов. Конструкторы без параметров называются также **конструкторами по умолчанию**.

Деструктор служит для освобождения ресурсов, выделенных при создании и накопленных за время жизни объекта. Деструкторы вызываются автоматически при уничтожении объектов. Дестукторы не возвращают значений, не имеют параметров и не могут быть перегружены. Объявление деструктора выглядит следующим образом:

```
~<имя класса>();
```

С использованием конструктора и деструктора приведенный выше пример с многомерным вектором действительных чисел будет выглядеть следующим образом:

```
class CVector{
  private:
    double *m_pData;
    int     m_iSize;
  public:
    CVector( int size )  {
```

```
        m_pData = new double[size];
        m_iSize = size;
    }
    CVector( int size, double v )  {
        m_pData = new double[size];
        m_iSize = size;
        for (int i = 0; i < m_iSize; i++)  m_pData[i] = v;
    }
    ~CVector()  {
        delete [] m_pData;
    }
    double& Elm( int ind );
    int GetSize();
    void Add( const CVector &v );
    void Mul( double m );
};
```

В том случае, если в классе не определен ни один конструктор, то компилятором будет автоматически сгенерирован конструктор без параметров. Аналогично, если отсутствует определение деструктора, будет автоматически сгенерирован деструктор. Это делает возможным создание и удаление объектов для которых конструкторы и деструкторы не были определены.

Для указания параметров конструктора при создании объекта используется инициализация объекта в стиле функции. Так, для класса CVector возможны:

```
CVector x1(2);
CVector x2(2, 1.0);
```

В том случае, если объекты создаются в динамической памяти, то:

```
CVector *p1 = new CVector(2);
CVector *p2 = new CVector(2, 1.0);
```

Следует отметить, что функции стандартной библиотеки для работы с динамической памтью (malloc, free и др.) не могут быть использованы для создания и уничтожения объектов класса, так как при их использовании конструкторы и деструкторы не вызываются.

Особую роль играет так называемый **конструктор копирования**. Объявляется конструктор копирования следующим образом:

```
<имя класса>(const <имя класса> &<имя параметра>)
```

Определение конструктора копирования для нашего примера выглядит так:

```
CVector(const CVector& src)
{
    Init( src.m_iSize );
    for (int i = 0; i < m_iSize; i++)
        m_pData[i] = src.m_pData[i];
}
```

В том случае, если конструктор копирования не объявлен явным образом, компилятор автоматически создает конструктор побитового копирования.

Конструктор копирования вызывается в следующих случаях:

1. При инициализации объекта существующим объектом того же типа:

2. При передаче параметра в функцию по значению, например:

3. При возврате значения из функции для хранения возвращаемого значения неявно создается временный безымянный объект, который будет удален, как только значение будет возвращено.

1.3 Наследование

Наследование является одним из основных принципов объектно-ориентированного программирования. Наследование позволяет создать новый класс на основе уже существующего класса. При этом вновь созданный класс может расширять функциональность базового класса, некоторым образом модифицировать или уточнить ее. В C++ реализована концепция множественного наследования, когда базовых классов может быть несколько. В литературе базовый класс называют также предком, родительским классом или суперклассом, а производный от него класс называют потомком, дочерним классом, или подклассом.

Синтаксически определение нового класса на базе уже существующего выглядит следующим образом:

```
class <имя класса>:    [[<атрибут наследования>]] <базовый класс> [ ,
                       [[<атрибут наследования>]] <базовый класс>, … ]
{…};
```

Здесь атрибут наследования позволяет ограничить область видимости полей и методов класса предка в классе наследнике. Хотя атрибут наследования и является необязательным, в большинстве случаев он указывается, так как по умолчанию наследование производится с атрибутом private. Влияние атрибута наследования на видимость полей и методов базового класса можно понять из приведенной ниже таблицы.

Видимость в базовом классе	Атрибут наследования		
	private	**protected**	**public**
	Видимость в производном классе		
Private	Недоступен	Недоступен	Недоступен
Protected	Private	Protected	Protected
Public	Private	Protected	Public

Пусть, например, имеется класс, реализующий стек свободных указателей на основе динамического массива указанного размера:

```
class CStack {
    private:
      void **m_dat;
      int m_cnt, m_sz;
    public:
      CStack(int sz = 100) {
          m_dat = new void*[sz];
          m_sz = sz;
          m_cnt = 0;
      }
      ~CStack(){
          delete [] m_dat;
      }
      // добавление нового указателя в вершину стека
      bool Push( void *data ) {
          if (m_cnt >= m_sz)  return false;
          m_dat[m_cnt++]=data;
          return true;
      }
      // извлечение из вершины стека
      bool Pop() {  return m_cnt ? (m_cnt--): 0;  };
      // доступ к вершине стека
      void* Get()  {  return m_cnt ? m_dat[m_cnt-1] : 0;  }
};
```

Создадим класс-наследник, предназначенный для хранения в стеке строк

```
class CStrStack : protected CStack
{
     public:
```

```
        ~CStrStack()  {
            while (Pop());
        };
        void Push( const char *data ) {
           CStack::Push( strdup(data) );
        };
        bool Pop() {
            if (CStack::Get())  free( CStack::Get() );
            return CStack::Pop();
        };
        const char* Get()  {
            return (const char*)CStack::Get();
        }
};
```

В приведенном примере для класса-наследника мы не определяли конструктор, полагаясь на конструктор по умолчанию, созданный компилятором автоматически. Действительно, созданный компилятором конструктор будет вызывать конструктор класса-предка по умолчанию (будет вызываться конструктор CStack(int sz = 100) со значением параметра sz по умолчанию). В том случае, когда класс-предок не имеет конструктора по умолчанию, конструктор для производного класса уже не может быть создан автоматически. В этом случае конструктор для производного класса должен быть реализован явно, а вызов конструктора класса-предка должен выполняться из списка инициализации конструктора. **Список инициализации** представляет собой список следующего вида:

<имя>(〚<список параметров>〛) 〚, <имя>(〚<список параметров>〛), …〛

Здесь <имя> – имя переменной-члена класса или класса-предка, <список параметров> - список параметров инициализатора соответствующего члена класса или конструктора. Таким образом, в списке инициализации можно обратиться не только к конструктору базового класса по умолчанию, но и конструктору с параметрами. С использованием списка инициализации конструкторы приведенных выше классов могли бы выглядеть следующим образом:

```
CStack::CStack(int sz = 100) : m_sz(sz), m_cnt(0)
{  m_dat = new void*[sz];  }
CStrStack::CStrStack(int sz = 20) : CStack(sz) {}
```

При создании объекта сначала вызывается конструктор базового класса, а уже затем - конструктор производного. Деструкторы вызываются в порядке обратном вызовам конструкторов.

1.4 Виртуальные функции и абстрактные классы

При вызове обычной функции-члена класса выбор самой вызываемой зависит не от фактического класса объекта, на котором установлен указатель, а от типа самого указателя. В тех случаях, когда такое поведение является неприемлемым, используют **виртуальные функции**.

Виртуальные функции-члены класса объявляются с использованием спецификатора virtual, причем если в базовом классе функция объявлена как виртуальная, то в производном классе функция автоматически становится виртуальной, независимо от того, используется ли virtual в производном классе. Виртуальные функции определяются в базовых классах и, как правило, переопределяются в производных. Механизм виртуальных функций является средством реализации концепции динамического полиморфизма (полиморфизм во время выполнения программы) и позволяет однообразно работать с объектами различных типов.

Разница в поведении при обращении к виртуальным и невиртуальным функиям-членам класса обусловлена типом связывания. При обращении к невиртуальным функциям-членам класса, также как и при обращении к статическим функциям класса или обычным функциям, используется так называемое раннее связывание, когда адрес вызываемой функции определяется на этапе компиляции программы. Такое определение выполняется по типу объекта или указателя, имени функции, количеству и типам параметров. При обращении к виртуальным функциям используется позднее (динамическое) связывание, когда адрес вызываемой функции определяется непосредственно при обращении к такой функции во время выполнения программы. В этом случае при определении адреса учитывается фактический тип объекта, для которого вызывается виртуальная функция.

В качестве иллюстрации приведем простой пример иерархии классов.

```
class CShape {
  public:
      virtual void Shift( int x, int y) {}
};
class CPoint: public CShape {
    int m_x, m_y;
  public:
      CPoint(int x, int y): m_x(x), m_y(y){}
      virtual void Shift( int x, int y)
      {  m_x+=x;  m_y+=y;  }
};
class CLine: public CShape {
```

```
    CPoint m_pt1, m_pt2;
  public:
      CLine (int x1, int y1, int x2, int y2)
        : m_pt1(x1,y1), m_pt2(x2, y2){}
      virtual void Shift( int x, int y)
      {  m_pt1.Shift(x,y);  m_pt2.Shift(x,y);  }
};
```

С использованием приведенной иерархии можно определить функцию сдвига набора объектов, представленного массивом указателей:

```
void Shift( int x, int y, CShape **Shapes, int Count)
{
    while (Count)  Shapes[--Count]->Shift(x, y);
}
```

Очень часто в базовом классе определяется лишь интерфейс взаимодействия с объектами, а реализация методов остается пустой. При этом предполагается, что такие методы будут переопределены в классах-потомках. В C++ в таком случае используются так называемые **чисто виртуальные функции**. Синтаксически чисто виртуальные функции обозначаются как виртуальные функции с указанием «=0» после списка параметров метода.

Например, базовый класс геометрических фигур может быть описан следующим образом:

```
class CShape {
  public:
      virtual void Shift( int x, int y) = 0;
      virtual void Rotate( double a) = 0;
      virtual void Show() = 0;
};
```

Чисто виртуальные функции обязательно должны быть переопределены в производных классах и часто вообще не имеют реализации. При необходимости, чисто виртуальный метод может иметь реализацию, например, реализация чисто виртуального метода может быть выполнена вне класса:

```
class CShape {
      bool m_visible;
  public:
      CShape()  m_visible(false)  {}
      virtual void Show() = 0;
```

```
};
void CShape::Show()
{  m_visible = true;  }
```

В том случае, если класс содержит хотя бы одну чисто виртуальную функцию, то такой класс называется **абстрактным**. Экземпляр такого класса не может быть создан, независимо от того, выполнена ли реализация чисто виртуальной функции или нет. Это, в частности, означает, что объекты абстрактных классов не могут быть передаваться в функции и возвращаться из функций по значению. В цепочке наследования производные классы становятся абстрактными до тех пор, пока все чисто виртуальные функции не будут переопределены.

При разработке программных систем часто применяются классы, содержащие только чисто виртуальные функции. Такие классы называют **интерфейсными** или **классами протокола**. Такие классы используют для определения строгого протокола взаимодействия между компонентами программной системы.

2 Пример выполнения лабораторной работы

Задача 1. Написать программу, в которой описана иерархия классов: ошибка в программе **(«ошибка доступа к памяти», «математическая», «деление на ноль», «переполнение»**). Наследники должны иметь поля, содержащие дополнительные сведения об ошибке, если такие имеются. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа объекта для демонстрации.

Решение.

```
#include <stdio.h>
#include <climits>

class EBaseError{
public:
     virtual void Print() =0;
     virtual void Read() =0;
};
class EAccessViolation: public EBaseError{
     void* m_badAddr;
public:
     void Print();
     void Read();
     EAccessViolation(void* badAddr);
};

class EMathError: public EBaseError{};

class EZeroDivide: public EMathError{
     double m_divident;
public:
     void Print();
     void Read();
     EZeroDivide(const double &divident);
};
class EOverflow: public EMathError{
     int m_operand1, m_operand2;
```

```
public:
     void Print();
     void Read();
     EOverflow(const int &operand1, const int &operand2);
};

EAccessViolation::EAccessViolation(void* badAddr){
     m_badAddr = badAddr;
}
void EAccessViolation::Print(){
     printf("Access violation read of address %p!", m_badAddr);
}
void EAccessViolation::Read(){
     //этот класс не может быть инициализирован пользователем
}

EZeroDivide::EZeroDivide(const double &divident){
     m_divident = divident;
}
void EZeroDivide::Print(){
     printf("There was a try to divide %lf by zero!", m_divident);
}
void EZeroDivide::Read(){
     //этот класс не может быть инициализирован пользователем
}

EOverflow::EOverflow(const int &operand1, const int &operand2){
     m_operand1 = operand1;
     m_operand2 = operand2;
}
void EOverflow::Print(){
     printf("There was an overflow during some operation between %d
and %d!", m_operand1, m_operand2);
}
void EOverflow::Read(){
     //этот класс не может быть инициализирован пользователем
```

```
}
void disposeError(EBaseError** e){
     if(*e!=NULL) delete *e;
     *e = NULL;
}
int main(){
     printf("Написать программу, в которой описана иерархия классов:
ошибка в программе\n («ошибка доступа к памяти», «математическая»,
«деление на ноль», «переполнение»).\n Наследники должны иметь поля,
содержащие дополнительные сведения об ошибке, если такие имеются.\n
Продемонстрировать работу всех методов классов,\n предоставив
пользователю выбор типа объекта для демонстрации.\n");
     EBaseError* e = NULL;
     char c = 0, tmp;
     while(c!=27){
          printf("\nвыберите действие:\n 1 - эмулировать ошибку
доступа\n");
          printf(" 2 - попытаться поделить два числа\n 3 - попытаться
умножить два числа\n");
          printf(" <esc> - выйти из программы\n");
          scanf("%c", &c);
          switch(c){
               case '1': e = new EAccessViolation((void*) &c); break;
               case '2':
                    double a, b;
                    printf("\nвведите делимое");
                    scanf("%lf", &a);
                    printf("\nвведите делитель");
                    scanf("%lf", &b);
                    if (b==0.0) e = new EZeroDivide(a);
                    else printf("\nрезультат = %lf", a/b);
                    break;
               case '3':
                    int i, j;
                    long long res;
                    printf("\nвведите первый множитель");
```

```
                    scanf("%d", &i);
                    printf("\nвведите второй множитель");
                    scanf("%d", &j);
                    res = i;
                    res *= j;
                    if (res>INT_MAX || res<INT_MIN) e = new
EOverflow(i, j);
                    else printf("\nрезультат = %d", res);
                    break;
               case '0' :c = 27;
          }
          scanf("%c", &tmp);//считываем Enter оставшийся в буфере
ввода после предыдущего scanf
          if(e != NULL) {
               printf("\n>\t");
               e->Print();
               printf("\n");
          }
          disposeError(&e);
     }
     return 0;
};
```

3 Содержание отчета

Отчет по лабораторной работе должен содержать:

1. Титульный лист.

2. Задание на лабораторную работу.

3. Описание основных алгоритмов и структур данных, используемых в программе:

4. Описание интерфейса пользователя программы.

5. Контрольный пример и результаты тестирования.

6. Листинг программы.

4 Контрольные вопросы

1. Дате определения понятиям класс и объект.

2. Как описывается класс на языке C++?

3. Какие средства языка поддерживают концепцию инкапсуляции?

4. Как определяются конструкторы и деструкторы? Каково их назначение?

5. Что такое конструктор по умолчанию?

6. В каких случаях производится вызов конструктора копирования?

7. Какие типы наследования Вы знаете?

8. Каково назначение атрибутов наследования?

9. Что такое список инициализации конструктора? Что может быть проинициализировано с его использованием?

10. Каков порядок вызова деструкторов при наследовании?

11. Каково предназначение виртуальных функций?

12. В чем особенности чисто виртуальных функций? Дайте определение абстрактному классу и классу протокола.

5 Задания на лабораторную работу

5.1 Начальный уровень сложности

Общие требования: в начале программы вывести задание; в процессе работы выводить подсказки пользователю (что ему нужно ввести, чтобы продолжить выполнение программы). В иерархии классов должен быть общий предок с чисто виртуальными функциями, нужными по заданию, а также с процедурами вывода на экран состояния экземпляра класса (`void Print()`) и ввода полей класса с клавиатуры (`void Read()`). После работы программы вся динамически выделенная память должна быть освобождена.

Варианты заданий:

1. Написать программу, в которой описана иерархия классов: ошибка в программе (**«ошибка доступа к памяти»**, **«математическая», «деление на ноль», «переполнение»**). Наследники должны иметь поля, содержащие дополнительные сведения об ошибке, если такие имеются. Продемонстрировать работу всех методов классов , предоставив пользователю выбор типа объекта для демонстрации.

2. Написать программу, в которой описана иерархия классов: средство передвижения (**велосипед**, **автомобиль, грузовик**). Базовый класс должен иметь поля для хранения средней скорости, названия модели, числа пассажиров, а также методы получения потребления топлива для данного расстояния и вычисления времени движения на заданное расстояние. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа объекта для демонстрации.

3. Написать программу, в которой описана иерархия классов: человек (**«дошкольник»**, **«школьник», «студент», «работающий»**). Базовый класс должен иметь поля для хранения ФИО, возраста, пола, а также методы получения среднего дохода и среднего расхода в денежном эквиваленте. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа объекта для демонстрации.

4. Написать программу, в которой описана иерархия классов: ошибка в программе (**«недостаточно памяти»**, **«ввода/вывода», «ошибка чтения файла», «ошибка записи файла»**). Наследники должны иметь поля, содержащие дополнительные сведения об ошибке, если такие имеются. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа объекта для демонстрации.

5. Написать программу, в которой описана иерархия классов: ошибка в программе (**«ошибочный указатель»**, **«ошибка работы со списком», «недопустимый индекс», «список переполнен»**). Наследники должны иметь поля, содержащие дополнительные сведения об

ошибке, если такие имеются. Продемонстрировать работу всех методов классов , предоставив пользователю выбор типа объекта для демонстрации.

6. Написать программу, в которой описана иерархия классов: ошибка в программе (**«недостаточно привилегий»**, **«ошибка преобразования», «невозможно преобразовать значение», «невозможно привести к интерфейсу»**). Наследники должны иметь поля, содержащие дополнительные сведения об ошибке, если такие имеются. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа объекта для демонстрации.

5.2 Средний уровень сложности

Общие требования: в начале программы вывести задание; в процессе работы выводить подсказки пользователю (что ему нужно ввести, чтобы продолжить выполнение программы). В иерархии классов должен быть общий предок с чисто виртуальными функциями, нужными по заданию, а также с процедурами вывода на экран состояния экземпляра класса (void Print()) и ввода полей класса с клавиатуры (void Read()).

Для всех классов необходимо реализовать конструктор копирования. После работы программы вся динамически выделенная память должна быть освобождена. Взаимодействие с пользователем организовать в виде простого меню, обеспечивающего возможность переопределения исходных данных и завершение работы программы.

Варианты заданий:

7. Написать программу, в которой описана иерархия классов: геометрические фигуры (**круг, прямоугольник, треугольник**). Реализовать методы вычисления площади и периметра фигуры. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа фигуры для демонстрации.

8. Написать программу, в которой описана иерархия классов: геометрические фигуры (**эллипс, квадрат, трапеция**). Реализовать методы вычисления площади и периметра фигуры. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа фигуры для демонстрации.

9. Написать программу, в которой описана иерархия классов: геометрические фигуры (**ромб, параллелепипед, эллипс**). Реализовать методы вычисления площади и периметра фигуры. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа фигуры для демонстрации.

10. Написать программу, в которой описана иерархия классов: геометрические фигуры (**куб, цилиндр, тетраэдр**). Реализовать методы вычисления объема и площади поверхности

фигуры. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа фигуры для демонстрации.

11. Написать программу, в которой описана иерархия классов: геометрические фигуры (**конус, шар, пирамида**). Реализовать методы вычисления объема и площади поверхности фигуры. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа фигуры для демонстрации.

12. Написать программу, в которой описана иерархия классов: числа (**целое, вещественное, комплексное**). Реализовать методы сложения, вычитания, произведения, деления. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа основного и вспомогательного числа для демонстрации.

13. Написать программу, в которой описана иерархия классов: **треугольник** (**равнобедренный, равносторонний, прямоугольный**). Базовый класс должен иметь поля для хранения длины двух сторон и угла между ними. Реализовать методы вычисления площади и периметра фигуры. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа треугольника для демонстрации.

14. Написать программу, в которой описана иерархия классов: **прогрессия** (**арифметическая, геометрическая**). Классы наследники должны иметь поля для хранения параметров прогрессии. Реализовать функции вычисления i-го элемента, суммы элементов до n-го элемента. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа треугольника для демонстрации.

15. Написать программу, в которой описана иерархия классов: геометрические фигуры (**круг, параллелепипед, трапеция**). Реализовать методы вычисления площади и периметра фигуры. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа фигуры для демонстрации.

16. Написать программу, в которой описана иерархия классов: геометрические фигуры (**эллипс, квадрат, треугольник**). Реализовать методы вычисления площади и периметра фигуры. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа фигуры для демонстрации.

17. Написать программу, в которой описана иерархия классов: геометрические фигуры (**шар, цилиндр, пирамида**). Реализовать методы вычисления объема и площади поверхности фигуры. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа фигуры для демонстрации.

18. Написать программу, в которой описана иерархия классов: геометрические фигуры (**куб, конус, тетраэдр**). Реализовать методы вычисления объема и площади поверхности

фигуры. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа фигуры для демонстрации.

19. Написать программу, в которой описана иерархия классов: геометрические фигуры (**ромб, прямоугольник, эллипс**). Реализовать методы вычисления площади и периметра фигуры. Продемонстрировать работу всех методов классов, предоставив пользователю выбор типа фигуры для демонстрации.

5.3 Высокий уровень сложности

Общие требования: Общие требования: в начале программы вывести задание; в процессе работы выводить подсказки пользователю (что ему нужно ввести, чтобы продолжить выполнение программы).

В иерархии классов должен быть общий предок с чисто виртуальными функциями, нужными по заданию, а также с процедурами вывода на экран состояния экземпляра класса (`void Print()`) и ввода полей класса с клавиатуры (`void Read()`).

Для всех классов необходимо реализовать конструктор копирования, конструктор по умолчанию, конструктор с параметрами для инициализации полей класса.

После работы программы вся динамически выделенная память должна быть освобождена.

Взаимодействие с пользователем организовать в виде простого меню, обеспечивающего возможность переопределения исходных данных и завершение работы программы, предусмотреть контроль вводимых пользователем данных.

Варианты заданий:

20. Написать программу, в которой описана иерархия классов: функция от одной переменной (**константа**, **линейная зависимость**, **парабола**). Базовый класс должен иметь методы получения значения функции для данного значения переменной, а также создания экземпляра класса, представляющего собой производную текущего экземпляра. Продемонстрировать работу всех методов классов всех классов.

21. Написать программу, в которой описана иерархия классов: функция от одной переменной (**синус**, **косинус**, **тангенс**). Базовый класс должен иметь методы получения значения функции для данного значения переменной, а также создания экземпляра класса, представляющего собой производную текущего экземпляра. Продемонстрировать работу всех методов классов всех классов.

22. Написать программу, в которой описана иерархия классов: функция от одной переменной (**секанс**, **косеканс**, **котангенс**). Базовый класс должен иметь методы получения значения функции для данного значения переменной, а также создания экземпляра класса,

представляющего собой производную текущего экземпляра. Продемонстрировать работу всех методов классов всех классов.

23. Написать программу, в которой описана иерархия классов: функция от одной переменной (**арксинус**, **арккосинус, а также класс, необходимый для представления производных**). Базовый класс должен иметь методы получения значения функции для данного значения переменной, а также создания экземпляра класса, представляющего собой производную текущего экземпляра. Продемонстрировать работу всех методов классов всех классов.

24. Написать программу, в которой описана иерархия классов: функция от одной переменной (**арктангенс**, **арккотангенс, а также класс, необходимый для представления производных**). Базовый класс должен иметь методы получения значения функции для данного значения переменной, а также создания экземпляра класса, представляющего собой производную текущего экземпляра. Продемонстрировать работу всех методов классов всех классов.

25. Написать программу, в которой описана иерархия классов: функция от одной переменной (**логарифм**, **натуральный логарифм, а также класс, необходимый для представления производных**). Базовый класс должен иметь методы получения значения функции для данного значения переменной, а также создания экземпляра класса, представляющего собой производную текущего экземпляра. Продемонстрировать работу всех методов классов всех классов.

26. Написать программу, в которой описана иерархия классов: функция от одной переменной (**экспонента**, **гиперболический синус, гиперболический косинус**). Базовый класс должен иметь методы получения значения функции для данного значения переменной, а также создания экземпляра класса, представляющего собой производную текущего экземпляра. Продемонстрировать работу всех методов классов всех классов.

27. Написать программу, в которой описана иерархия классов: функция от одной переменной (**степенная**, **показательная**). Базовый класс должен иметь методы получения значения функции для данного значения переменной, а также создания экземпляра класса, представляющего собой производную текущего экземпляра. Продемонстрировать работу всех методов классов всех классов.

6 Библиографический список

1. Страуструп Б. Язык программирования C++. Специальное издание. М.: Радио и связь, 1991. - 349с.

2. Савитч У. Язык C++. Курс объектно-ориентированного программирования. – М.: Вильямс, 2001. – 696с.

3. Вайнер Р., Пинсон Л. C++ изнутри.- Киев:НПИФ «ДиаСофт», 1993. -301с.

4. Программирование на C++ / С. Дьюхарст, К. Старк. - Киев : НИПФ "ДиаСофт", 1993. - 271 с.