Цель работы: Изучение правил перегрузки операций и принципов обработки исключений в C++.

1 Теоретические основы лабораторной работы

1.1 Перегрузка операций

Для того чтобы перегрузить операцию в C++ следует объявить функцию или функцию-член класса следующего вида:

```
<тип> operator <символ> ([<список параметров>]);
```

Здесь <тип> - тип результата операции, <символ> - символьное обозначение операции, <список параметров> - параметры функции перегрузки операции. Количество параметров в списке зависит от того является ли операция ***унарной*** или ***бинарной***, а также от способа перегрузки операции. В случае, когда перегрузка выполняется функцией вне пользовательского класса, количество параметров функции совпадает с количеством операндов перегружаемой операции. Так, для унарной операции <символ> <аргумент>, перегружаемой вне класса функция имеет единственный параметр:

```
<тип> operator <символ> (<тип1> <парам1>);
```

Для бинарной операции <аргумент1> <символ> <аргумент2>, перегружаемой вне класса, функция имеет следующий вид:

```
<тип> operator <символ> (<тип1> <парам1>, <тип2> <парам2>);
```

Например, для операции сложения векторов:

```
CVector operator +(const CVector &v1, const CVector &v2);
```

В случае, когда перегрузка выполняется функцией-членом класса количество параметров функции на единицу меньше числа операндов перегружаемой операции. В этом случае первый из операндов операции передается в функцию-член класса в качестве указателя this и отсутствует в списке параметров функции. Для унарной операции, перегружаемой внутри класса, функция имеет следующий вид:

```
<тип> operator <символ>();
```

Для бинарной операции, перегружаемой внутри класса, функция имеет один параметр:

```
<тип> operator <символ>( <тип2> <парам2>);
```

Из сказанного следует, что операция, в которой первый аргумент не является объектом пользовательского класса, не может быть перегружена функцией-членом класса, а только внешней по отношению к классу функцией. К сожалению, ввести новое символьное обозначение для операции нельзя. Также нельзя перегрузить оператор для базовых типов данных.

Перегрузку операций внутри и вне класса рассмотрим на примере пользовательского класса для работы со строками:

```
class CMyString{
    private:
        char* m_data;
    public:
        // конструктор по-умолчанию
        CMyString() : m_data(NULL) {};
        // конструктор копирования
        CMyString( const CMyString &Src )
            { m_data = strdup(Src.m_data); };
        // деструктор
        ~CMyString() {  if ( m_data )  free(m_data); }
        // функция определения длины строки
        int Len() const {
            if ( m_data ) return strlen( m_data );
            else return 0;
        }
        // перегрузка унарного оператора NOT
        //(обращение цепочки символов)
        CMyString operator ! ()  {
            CMyString res;
            if ( ! Len() )  return res;
            res.m_data = (char*)malloc( Len() + 1 );
            char *p = res.m_data + Len(),
                 *q = m_data;
            *p-- = 0;
            while ( *q != 0 ) *p-- = *q++;
            return res;
        }
        // перегрузка бинарной операции сравнения
        int operator == ( const CMyString & Src )  {
            if ( m_data && Src.m_data )
                return ( 0 == strcmpi(m_data, Src.m_data) );
            else if ( m_data == Src.m_data )  return 1;
            else return 0;
        }
```

```
        // перегрузка операции сложения (конкатенация строк)
        // дружественной функцией
        friend CMyString operator + (const CMyString& s1,
                                     const CMyString& s2);
};
CMyString operator + (const CMyString& s1, const CMyString& s2)  {
    if ( ! s1.Len() ) return s2;
    if ( ! s2.Len() ) return s1;
    CMyString res;
    res.m_data = (char*)malloc( s1.Len() + s2.Len() + 1 );
    strcpy( res.m_data, s1.m_data );
    strcat( res.m_data, s2.m_data );
    return res;
}
// перегрузка операции сравнения
int operator != (CMyString& s1, CMyString& s2){
    return !( s1 == s2 );
}
```

Приведенные выше правила справедливы для большинства стандартных операций языка, однако некоторые из операций имеют особенности. Так не допускается перегрузка следующих операций:

- ::
- ?:
- .*
- .
- typeid
- sizeof

Существуют операции, которые могут быть перегружены только с использованием нестатических функций-членов классов. Это операции =, (), [], ->.

Операция присваивания «=». Как мы уже сказали, операция присваивания может быть перегружена только функцией-членом класса. Операция присваивания – единственная из всех перегружаемых операций не наследуется классами-потомками. Однако, в том случае, если операция не определена для класса явным образом, компилятором будет создана реализация операции присваивания по умолчанию, которая будет выполнять почленное копирование объектов класса. Как и в случае с конструктором копирования, автоматическая реализация операции присваивания может принести больше вреда. Поэтому, операцию присваивания для классов, содержащих данные в динамической памяти, лучше корректно перегрузить, или хотя бы закрыть. При реализации операции присваивания следует иметь в виду, что возможен

случай самоприсваивания объектов, когда оба аргумента операции являются на самом деле одним и тем же объектом.

```
class CMyString{
        ...
        CMyString& operator = ( const CMyString & Src )  {
            if ( this == (&Src) )  return *this;
            if ( m_data )  free(m_data);
            if ( Src.m_data )  m_data = strdup( Src.m_data );
            else  m_data = NULL;
            return *this;
        }
};
```

Операция обращения по индексу «**[]**» является бинарной и может быть перегружена только как функция-член класса. В качестве второго операнда допускается объект любого типа, а не обязательно целое число. Так, для класса, содержащего набор пар «строковый (const char *) ключ – вещественное (double) значение», операция доступа по индексу может быть определена как

```
class CMyString{
        ...
        char& operator[](int ind) {
            return m_data[ind];
        }
};
```

Операция вызова функции «()». Эта операция рассматривается как бинарная операция, второй аргумент которой представляет собой список параметров вызова (содержащий ноль или более параметров). Перегружается операция также только как функция-член класса. При перегрузке следует иметь в виду, что в зависимости от решаемой задачи функция-член класса может иметь как набор фиксированных параметров, так и иметь неопределенное число параметров. Конечно же, перегруженная операция применяется только к объектам класса, а не к функциям, как ее базовая реализация. В качестве примера можно привести перегрузку этой операции для обращения к элементу матрицы:

```
class CMatrix { …
    double operator () (int m, int n);
};
```

Операции преобразования типа (функции преобразования). Операции преобразования типа позволяют выполнять явное преобразование объекта класса к любому другому типу. Для перегрузки операции преобразования типа объявляется особая нестатическая функция-член класса вида:

```
operator <тип> ();
```

Здесь функция-член класса не имеет никаких параметров, тип возвращаемого значения для таких функций также не указывается, а определяется частью объявления <тип>. В части <тип> помимо собственно типа могут фигурировать спецификаторы типа и обозначения указателя (например, const char*). Перегруженные операции приведения типа наследуются и могут быть переопределены в классах потомках. При этом будут переопределены только операции с полностью совпадающей сигнатурой. Например, если в предке были определены функции operator float () и operator double (), то определенная в потомке operator float() переопределит только первую из функций, а вторая функция будет унаследована потомком.

```
class CMyString{
        ...
        operator const char*() {
            return m_data;
        }
};
```

Операции обращения к члену класса (->) и ***разыменования указателя*** (*) перегружаются сравнительно редко. Перегрузка этих операции используется для создания класса «умных указателей» (smart pointers). Кроме того, с использованием операции обращения к члену класса может быть реализована концепция делегирования. Мы не будем здесь подробно останавливаться на перегрузке этих операций.

Операции инкремента и декремента (++ и --). Особенность перегрузки этих операций связана с тем, что обе операции могут быть как префиксными, так и постфиксными. Для того чтобы отличить префиксные и постфиксные операции друг от друга, для постфиксных операций вводится дополнительный целочисленный параметр, который, как правило, не используется. Поэтому при объявлении в классе рассматриваемые префиксные операции имеют вид:

```
<тип> operator ++();
<тип> operator --();
```

Соответствующие постфиксные операции:

```
<тип> operator ++ (int);
<тип> operator ++ (int);
```

При объявлении вне класса в функции первым параметром добавляется объект пользовательского класса, относительно которого выполняется операция.

При вычислении выражений дополнительный целочисленный параметр принимает значение 0. Передать иное значение можно только при явном вызове операции

```
<объект>.operator++(<значение>);
```

Такой способ вызова, однако, практически не используется.

При перегрузке любых операций часто возникает вопрос о том, как должен передаваться в функцию тот или иной объект – по ссылке или по значению. Аналогичные вопросы возникают и в отношении возвращаемого значения. Здесь можно дать следующие общие рекомендации.

Если предполагается, что операнд при выполнении операции не изменяется (например, операнды в операциях +, -, &&, ||), то в функцию его лучше передавать по значению или константной ссылке. Причем, если операндом является объект, занимающий большой объем памяти, то его лучше передавать по константной ссылке. Для базовых типов данных вполне подойдет передача по значению.

Если операнд изменяется при выполнении операции (например, первый операнд в операциях +=, &=, ++), то такой операнд должен передаваться по неконстантной ссылке.

Когда мы говорим о возвращаемом значении, то здесь можно придерживаться такого правила. Если возвращается объект, который не существовал до выполнения операции (например, результат операции +,*,|), то такой объект лучше возвращать по значению. В том случае, если объект уже существовал до выполнения операции (например, ++,+=,[]), то такой объект нужно возвращать по ссылке. Конечно, выше приведены лишь общие рекомендации и в некоторых случаях, например, в целях оптимизации, можно откланяться от них.

1.2 Исключения

При работе с исключениями используются три ключевых слова try, catch и throw. При этом первые два обозначают так называемый блок try-catch и связаны с обработкой возникших исключений, а throw – с выбрасыванием исключений. Синтаксически блок try-catch выглядит следующим образом:

```
try  {
    <операторы>
}
catch(<описание исключения>) {
    <операторы обработки исключения>
}
[catch(<описание исключения>) {
```

    <операторы обработки исключения>
} … ⟧

Здесь <операторы> в блоке try представляют собой команды, во время выполнения которых может произойти исключительная ситуация. В том случае если некоторый набор операторов заключен в блок try, говорят, что операторы находятся в защищенном блоке. <описание исключения> показывает тип исключения, которое будет обрабатываться в блоке catch. Здесь может быть представлен любой тип данных, как базовым, так и производный. Кроме того здесь может быть описана переменная указанного типа, которую в дальнейшем можно будет использовать при обработке исключения. Помимо описания типа или переменной здесь может стоять троеточие, которое показывает, что данный блок catch обрабатывает исключения любого типа. <Операторы обработки исключения> в блоке catch предназначены для обслуживания возникшей исключительной ситуации определенного типа. Однако в том случае, если при обработке исключения обнаруживается ошибка, или полностью нейтрализовать возникшую ситуацию не представляется возможным, блок catch сам может быть источником исключений, которые должны отлавливаться и обрабатываться в каком-либо внешнем блоке try-catch.

Для генерации исключения внутри блока try должен быть выполнен оператор throw, параметром которого является объект-исключение желаемого типа.

throw ⟦<выражение>⟧

Если при выполнении программы достигнут блок try-catch, то происходит выполнение операторов, находящихся внутри блока try. Если при выполнении этих операторов не возникает исключительной ситуации (не выполняется оператор throw), то выполнение продолжается с оператора, следующего за последним из блоков try, то есть никакие команды из блоков try не выполняются.

Если при выполнении операторов в try производится генерация исключения с использованием ключевого слова throw, то на основе объекта, переданного throw, создается объект - исключение и производится поиск соответствующего блока обработки catch. При поиске обработчики просматриваются в порядке следования блоков catch в программном коде. При этом, если соответствующий по типу обработчик не найден, то выполняется поиск во внешнем блоке try-catch (если таковой имеется), по отношению к рассматриваемому. В том случае, если обработчик найти не удается, то производится аварийное завершение программы.

Если соответствующий обработчик найден и его формальный параметр указан, как параметр, передаваемый по значению (catch (<тип> <параметр>)), то параметр инициализируется копией объекта-исключения. В том случае, если параметр указывается по

ссылке (catch (<тип> &<параметр>)), ссылка инициализируется адресом объекта-исключения.

После инициализации параметра запускается процесс так называемой *раскрутки стека*. В этом процессе выполняется уничтожение (вызываются деструкторы) всех объектов, созданных внутри блока try и находящихся в автоматической памяти. При этом уничтожение объектов производится в порядке обратном их созданию.

После раскрутки стека выполняются <операторы обработки исключения>, располагающиеся в соответствующем блоке catch. В том случае, если при их выполнении никаких исключений не происходит, выполнение продолжается с оператора, следующего за последним из блоков try.

Чтобы облегчить работу с исключениями и сделать ее более элегантной классы исключений должны образовывать некоторую иерархию. Например:

```
class EMyException {
  protected:
    char* m_msg;
  public:
    EMyException( const char *Msg )
    {  m_msg = strdup(Msg);  }
    EMyException( const EMyException &src )
    {  m_msg = strdup(src.m_msg);  }
    ~EMyException()
    {  free(m_msg);  }
    const char* Message() { return m_msg; };
    virtual void Print()  {  puts( m_msg );  }
};
class EInvalidArg:  public EMyException {
  protected:
    char *m_name;
  public:
    EInvalidArg( const char *Name ):
        EMyException("Недопустимый параметр")
    {  m_name = strdup(Name);  }
    EInvalidArg( const EInvalidArg &src ):
        EMyException(src)
    {  m_name = strdup(src.m_name);  }
```

```
    ~EInvalidArg ()
    {  free(m_name);  }
    virtual void Print()
    {  printf( "Недопустимый параметр <%s>", m_name );  }
};
class EInvalidIndex:  public EMyException {
  protected:
    int m_min, m_max, m_ind;
  public:
    EInvalidIndex ( int Min, int Max, int Ind ):
        EMyException("Неверный индекс")
    {  m_min=Min; m_max=Max; m_ind=Ind;  }
    EInvalidIndex ( const EInvalidIndex &src ):
        EMyException(src)
    {  m_min=src.m_min; m_max=src.m_max; m_ind=src.m_ind;  }
    virtual void Print()
    {  printf( "Значение индекса (%d) выходит за диапазон"
               " [%d;%d]", m_ind, m_min, m_max );  }
};
```

Напомним, что при работе с исключениями создаются копии объектов-исключений. Поэтому для тех классов исключений, для которых автоматически создаваемый компилятором конструктор копирования некорректен, конструктор копирования должен быть реализован явным образом.

С использованием введенных здесь типов исключений операции с объектами класса вектор могут быть реализованы, например, следующим образом:

```
class CVector{
  private:
    double *m_pData;
    int     m_iSize;
  public:
    …
    CVector operator +( const CVector &v2)  {
        if (m_iSize != v2.m_iSize)  throw EInvalidArg("v2");
        CVector res(m_iSize);
        for ( int i = 0; i < m_iSize; i++)
```

```
            res.m_pData[i] = m_pData[i]+v2.m_pData[i];
        return res;
    }
    double& operator []( int ind )  {
        if (ind < 0 || ind >= m_iSize)
            throw EInvalidIndex(0,m_iSize-1,ind);
        return m_pData[ind];
    }
};
```

Фрагмент кода работы с реализованными выше операциями приведен ниже.

```
try {
    CVector v1(3,3.3), v2(2,2.2), v3;
    if (exc_no==1)  v1[3] = 1.1;
    else if (exc_no==2)  v3=v1+v2;
    else puts(”Исключений не будет”);
}
catch (const EMyException &e)
{ e.Print(); }
catch ( … )
{ puts( "Неизвестное исключение" ); }
```

Здесь в зависимости от значения целочисленной переменной exc_no будет создана исключительная ситуация, соответствующая одному из введенных выше типов. Обратите внимание, что обработчик исключений здесь принимает параметр не по значению, а по ссылке, что гарантирует появление правильного сообщения при вызове виртуального метода Print.

Следует отметить, что, так как при поиске обработчиков они просматриваются в том порядке, в котором записаны, и обработчик считается найденным, если объект-исключение является производным от типа, указанного в обработчике, то типы в обработчике исключений следует располагать в порядке расширения типа. При этом обработчик catch(…), который будет ловить все исключения, следует располагать последним.

В ряде случаев обработчик не в состоянии до конца обработать возникшую ошибку. В этом случае он может выбросить новое исключение или повторно сгенерировать то же самое исключение путем вызова throw без параметров, передавая, таким образом, исключение во внешний блок try-catch.

2 Пример выполнения лабораторной работы

Задание 1. Написать программу, в которой описана иерархия классов: ошибка в программе (**«ошибка доступа к памяти»**, **«математическая», «деление на ноль», «переполнение»**). Описать класс для хранения коллекции ошибок (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

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);
     bool operator== ( const EAccessViolation& right){
          return m_badAddr == right.m_badAddr;
     }
     bool operator!= ( const EAccessViolation& right){
          return !(*this==right);
     }
     EAccessViolation& operator= (const EAccessViolation& right){
          m_badAddr = right.m_badAddr;
          return *this;
     }
};

class EMathError: public EBaseError{};
```

```
class EZeroDivide: public EMathError{
     double m_divident;
public:
     void Print();
     void Read();
     EZeroDivide(const double &divident);
     bool operator== ( const EZeroDivide& right){
          return m_divident == right.m_divident;
     }
     bool operator!= ( const EZeroDivide& right){
          return !(*this==right);
     }
     EZeroDivide& operator= (const EZeroDivide& right){
          m_divident= right.m_divident;
          return *this;
     }
};
class EOverflow: public EMathError{
     int m_operand1, m_operand2;
public:
     void Print();
     void Read();
     EOverflow(const int &operand1, const int &operand2);
     bool operator== ( const EOverflow& right){
          return (m_operand1 == right.m_operand1)&&(m_operand2 ==
right.m_operand2);
     }
     bool operator!= ( const EOverflow& right){
          return !(*this==right);
     }
     EOverflow& operator= (const EOverflow& right){
          m_operand1 = right.m_operand1;
          m_operand2 = right.m_operand2;
          return *this;
     }
```

```
};
typedef EBaseError* pError;
class CErrors{
     pError* m_errs;
     int m_cnt;
public:
     CErrors(int errCount){
          m_errs = new pError[errCount];
          for(int i=0; i<errCount; i++)
               m_errs[i] = NULL;
          m_cnt = errCount;
     }
     ~CErrors(){
          for(int i=0; i<m_cnt; i++)
               if(m_errs[i] != NULL)delete m_errs[i];
          delete [] m_errs;
     }
     EBaseError*  operator[] (int n) const{
          if(n<0 || n>=m_cnt) throw EAccessViolation((void*)
&m_errs[n]);
          return m_errs[n];
     }
     EBaseError*& operator[] (int n){
          if(n<0 || n>=m_cnt) throw EAccessViolation((void*)
&m_errs[n]);
          return m_errs[n];
     }
};

EAccessViolation::EAccessViolation(void* badAddr){
     m_badAddr = badAddr;
}
void EAccessViolation::Print(){
     printf("Access violation read of address %p!", m_badAddr);
}
void EAccessViolation::Read(){
```

```
     printf("Simulate EAccessViolation, enter badAddress");
     scanf("%p", &m_badAddr);
}

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(){
     printf("Simulate EZeroDivide, enter divident");
     scanf("%lf", &m_divident);
}

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(){
     printf("Simulate EOverflow, enter two operands");
     scanf("%d %d", &m_operand1, &m_operand1);
}
void disposeError(EBaseError** e){
     if(*e!=NULL) delete *e;
     *e = NULL;
}
int _tmain(int argc, _TCHAR* argv[])
{
     printf("Написать программу, в которой описана иерархия классов:
ошибка в программе\n («ошибка доступа к памяти», «математическая»,
«деление на ноль», «переполнение»).\n Описать класс для хранения
```

```
коллекции ошибок (массива указателей на базовый класс),\n в котором
перегрузить операцию «[ ]». Для базового класса и его потомков\n
перегрузить операции «==», «!=», «=».\n Продемонстрировать работу
операторов.\n");
     EBaseError* e = NULL;
     CErrors errList(10);
     int curErrIndex = 0;
     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(" 4 - получить ошибку по номеру\n 5 - сравнить две
ошибки\n");
          printf(" 6 - напечатать все ошибки\n");
          printf(" 0  - выйти из программы\n");
          try{
               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 '4':{
                    int inx;
                    printf("\nвведите номер ошибки ");
                    scanf("%d", &inx);
                    EBaseError* tmpe = errList[inx];
                    if (tmpe == NULL) printf("\nОшибка
пуста\n");
                    else {
                         printf("\n>\t");
                         tmpe->Print();
                    }}
                    break;
               case '5':{
                    EAccessViolation ea1(0), ea2(0);
                    ea1.Read();
                    ea2.Read();
                    if (ea1 == ea1) printf("\nошибки равны\n");
                    else printf("\nошибки не равны\n");}
                    break;
               case '6':
                    for(int i=0; i<curErrIndex; i++)
                         if(errList[i]!=NULL){
                              printf("\n%d\t>\t", i);
                              errList[i]->Print();
                         }
                    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");
                     if(curErrIndex<10){
                          errList[curErrIndex++] = e;
                          e = NULL;
                     }
                     else disposeError(&e);
                }
           }
           catch( EBaseError &re){
                printf("\n ошибка времени выполения\t");
                re.Print();
                printf("\n");
           }
           catch(...){
                printf("\n неизвестная ошибка времени выполения\n");
           }

     }
     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. Назовите основные правила перегрузки унарных и бинарных операций.

2. Какие особенности имеет перегрузка операций присваивания, обращения по индексу, вызова функции, преобразования тип

3. Каким образом осуществляется перегрузка префиксных и постфиксных операций инкремента (декремента).

4. Как перегрузить операции потокового ввода-вывода для пользовательских типов данных?

5. Каким образом можно сгенерировать исключение? Какие типы могут использоваться при этом?

6. Как поймать и обработать исключение? Опишите механизм обработки исключений.

7. В чем отличие передачи объекта-исключения в обработчик по ссылке и по значению?

8. Зачем создавать иерархии классов для описания исключений? Каким должен быть порядок записи обработчиков исключений?

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). Иерархию классов следует взять из лабораторной работы №3.

Класс коллекция может не иметь методов для изменения количества хранимых объектов. При обращении к элементам с несуществующим индексом должно выбрасываться исключение. 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»**). Описать класс для хранения коллекции ошибок (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.
2. Написать программу, в которой описана иерархия классов: средство передвижения (**велосипед**, **автомобиль, грузовик**). Описать класс для хранения коллекции средств передвижений (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.
3. Написать программу, в которой описана иерархия классов: человек (**«дошкольник»**, **«школьник», «студент», «работающий»**). Описать класс для хранения коллекции людей (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.
4. Написать программу, в которой описана иерархия классов: ошибка в программе (**«недостаточно памяти»**, **«ввода/вывода», «ошибка чтения файла», «ошибка записи файла»**). Описать класс для хранения коллекции ошибок (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.
5. Написать программу, в которой описана иерархия классов: ошибка в программе (**«ошибочный указатель»**, **«ошибка работы со списком», «недопустимый индекс», «список переполнен»**). Описать класс для хранения коллекции ошибок (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

6. Написать программу, в которой описана иерархия классов: ошибка в программе (**«недостаточно привилегий»**, **«ошибка преобразования», «невозможно преобразовать значение», «невозможно привести к интерфейсу»**). Описать класс для хранения коллекции ошибок (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

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). Иерархию классов следует взять из лабораторной работы №3. После работы программы вся динамически выделенная память должна быть освобождена. Класс коллекция должна иметь методы для изменения количества хранимых объектов: добавление в конец, вставка, усечение, удаление из середины. При обращении к элементам с несуществующим индексом или при некорректном изменении количества должно выбрасываться исключение.

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**). Описать класс для хранения коллекции фигур (массива указателей на базовый класс), в котором перегрузить операцию «[ ]», а также реализовать функции подсчёта общей площади и периметра. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

8. Написать программу, в которой описана иерархия классов: геометрические фигуры (**эллипс, квадрат, трапеция**). Описать класс для хранения коллекции фигур (массива указателей на базовый класс), в котором перегрузить операцию «[ ]», а также реализовать функции подсчёта общей площади и периметра. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

9. Написать программу, в которой описана иерархия классов: геометрические фигуры (**ромб, параллелепипед, эллипс**). Описать класс для хранения коллекции фигур (массива указателей на базовый класс), в котором перегрузить операцию «[ ]», а также реализовать функции подсчёта общей площади и периметра. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

10. Написать программу, в которой описана иерархия классов: геометрические фигуры (**куб, цилиндр, тетраэдр**). Описать класс для хранения коллекции фигур (массива указателей на базовый класс), в котором перегрузить операцию «[ ]», а также реализовать функции подсчёта общего объема и площади поверхности. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

11. Написать программу, в которой описана иерархия классов: геометрические фигуры (**конус, шар, пирамида**). Описать класс для хранения коллекции фигур (массива указателей на базовый класс), в котором перегрузить операцию «[ ]», а также реализовать функции подсчёта общего объема и площади поверхности. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

12. Написать программу, в которой описана иерархия классов: числа (**целое, вещественное, комплексное**). Описать класс для хранения коллекции чисел (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

13. Написать программу, в которой описана иерархия классов: **треугольник** (**равнобедренный, равносторонний, прямоугольный**). Описать класс для хранения коллекции треугольников (массива указателей на базовый класс), в котором перегрузить операцию «[ ]», а также реализовать функции подсчёта общей площади и периметра. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

14. Написать программу, в которой описана иерархия классов: **прогрессия** (**арифметическая, геометрическая**). Описать класс для хранения коллекции прогрессий (массива указателей на базовый класс), в котором перегрузить операцию «[ ]», а также реализовать функции подсчёта общей суммы. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

15. Написать программу, в которой описана иерархия классов: геометрические фигуры (**круг, параллелепипед, трапеция**). Описать класс для хранения коллекции фигур (массива указателей на базовый класс), в котором перегрузить операцию «[ ]», а также реализовать функции подсчёта общей площади и периметра. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

16. Написать программу, в которой описана иерархия классов: геометрические фигуры (**эллипс, квадрат, треугольник**). Описать класс для хранения коллекции фигур (массива указателей на базовый класс), в котором перегрузить операцию «[ ]», а также реализовать функции подсчёта общей площади и периметра. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

17. Написать программу, в которой описана иерархия классов: геометрические фигуры (**шар, цилиндр, пирамида**). Описать класс для хранения коллекции фигур (массива указателей на базовый класс), в котором перегрузить операцию «[ ]», а также реализовать функции подсчёта общей площади и периметра. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

18. Написать программу, в которой описана иерархия классов: геометрические фигуры (**куб, конус, тетраэдр**). Описать класс для хранения коллекции фигур (массива указателей на базовый класс), в котором перегрузить операцию «[ ]», а также реализовать функции подсчёта общей площади и периметра. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

19. Написать программу, в которой описана иерархия классов: геометрические фигуры (**ромб, прямоугольник, эллипс**). Описать класс для хранения коллекции фигур (массива указателей на базовый класс), в котором перегрузить операцию «[ ]», а также реализовать функции подсчёта общей площади и периметра. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

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). Иерархию классов следует взять из лабораторной работы №3. После работы программы вся динамически выделенная память должна быть освобождена. Класс коллекция должна иметь методы для изменения количества хранимых объектов: добавление в конец, вставка, усечение, удаление из середины. При обращении к элементам с несуществующим индексом или при некорректном изменении количества должно выбрасываться исключение. Исключение также должно пониматься, если значение функции не существует для данного значения переменной.

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

Варианты заданий:

20. Написать программу, в которой описана иерархия классов: функция от одной переменной (**константа**, **линейная зависимость**, **парабола**). Описать класс для хранения коллекции функций (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Описать класс-итератор для итерации по элементам коллекции. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

21. Написать программу, в которой описана иерархия классов: функция от одной переменной (**синус**, **косинус**, **тангенс**). Описать класс для хранения коллекции функций (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Описать класс-итератор для итерации по элементам коллекции. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

22. Написать программу, в которой описана иерархия классов: функция от одной переменной (**секанс**, **косеканс**, **котангенс**). Описать класс для хранения коллекции функций (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Описать класс-итератор для итерации по элементам коллекции. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

23. Написать программу, в которой описана иерархия классов: функция от одной переменной (**арксинус**, **арккосинус, а также класс, необходимый для представления производных**). Описать класс для хранения коллекции функций (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Описать класс-итератор для итерации по элементам коллекции. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

24. Написать программу, в которой описана иерархия классов: функция от одной переменной (**арктангенс**, **арккотангенс, а также класс, необходимый для представления производных**). Описать класс для хранения коллекции функций (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Описать класс-итератор для итерации по элементам коллекции. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

25. Написать программу, в которой описана иерархия классов: функция от одной переменной (**логарифм**, **натуральный логарифм, а также класс, необходимый для представления производных**). Описать класс для хранения коллекции функций (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Описать класс-итератор для итерации по элементам коллекции. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

26. Написать программу, в которой описана иерархия классов: функция от одной переменной (**экспонента**, **гиперболический синус, гиперболический косинус**). Описать класс для хранения коллекции функций (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Описать класс-итератор для итерации по элементам коллекции. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

27. Написать программу, в которой описана иерархия классов: функция от одной переменной (**степенная**, **показательная**). Описать класс для хранения коллекции функций (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Описать класс-итератор для итерации по элементам коллекции. Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

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.