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)

Шаблоны в C++

Электронные методические указания
к лабораторной работе № 5

Самара
2011

Составители: МЯСНИКОВ Евгений Валерьевич
ПОПОВ Артем Борисович

В лабораторной работе № 5 по дисциплине "Языки и методы программирования" изучаются принципы работы с шаблонами в языке 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

Цель работы: Изучение синтаксиса и принципов работы с шаблонами классов в C++. Изучение особенностей специализации шаблонов классов и методов.

1 Теоретические основы лабораторной работы

1.1 Шаблоны классов

Шаблон класса позволяет создать семейство родственных классов. Создаваемые классы имеют одинаковое поведение и состав свойств, однако отличаются используемыми типами данных, задаваемыми в качестве параметров шаблона.

Синтаксически шаблон класса определяется следующим образом:

```
template < <список параметров шаблона> >
<определение класса>
```

<список параметров шаблона> представляет собой последовательность параметров, разделенных запятыми

```
<описание параметра> [,<описание параметра> , …]
```

В качестве параметров шаблона (<описание параметра>) могут использоваться:

- типы данных

```
class <имя> [=<тип по умолчанию>]
typename <имя> [=<тип по умолчанию>]
```

- константы

```
<тип_константы> <идентификатор>[=<значение по умолчанию>]
```

- другие шаблоны

```
template < <список параметров шаблона> > class <имя шаблона класса>
[=<тип по умолчанию>]
```

Любые параметры шаблона могут быть заданы значениями по умолчанию.

Шаблоны целесообразно использовать в тех случаях, когда типы используемых в классе данных не оказывают существенного влияния на реализацию класса. В частности, шаблоны целесообразно применять при создании контейнерных классов.

Рассмотрим, например, как может быть реализован шаблон класса стек.

```
template < class T, int Size = 256 >
class TFixedStack{
    private:
      T m_dat[Size];
      int m_cnt;
    public:
      TFixedStack()  : m_cnt( 0 ) {}
```

```
    // добавление в вершину стека
    bool Push( const T &data ) {
        if (m_cnt >= Size)  return false;
        m_dat[m_cnt++] = data;
        return true;
    }
    // извлечение из вершины стека
    bool Pop(T &data) {
        if (! m_cnt)  return false;
        data = m_dat[m_cnt--];
        return true;
    };
    // доступ к вершине стека
    bool Get(T &data) {
        if (! m_cnt)  return false;
        data = m_dat[m_cnt];
        return true;
    };
};
```

В отличие от шаблонов функций при создании объекта шаблона класса параметры шаблона всегда требуется указывать явным образом, например:

```
TFixedStack <double, 10> s;
```

После того, как объект создан, работа с ним ничем не отличается от работы с объектами обычных классов:

```
s.Push(105.01);
double val;
s.Get(val);
s.Pop(val);
```

В случае если часть параметров шаблона имеет значения по умолчанию, при создании объекта они могут не указываться, например:

```
TFixedStack <double> s;
```

Шаблоны классов могут участвовать в наследовании, как в качестве базовых, так и в качестве производных классов:

```
template < class T >
class TAbstractStack {
```

```
    public:
      // виртуальный деструктор
      virtual ~TAbstractStack(){}
      // добавление в вершину стека
      virtual bool Push( const T &data ) = 0;
      // извлечение из вершины стека
      virtual bool Pop(T &data) = 0;
      // доступ к вершине стека
      virtual bool Get(T &data) = 0;
};
template < class T, int Size = 256 >
class TFixedStack : public TAbstractStack<T> {
    private:
      T m_dat[Size];
      int m_cnt;
    public:
      TFixedStack()  : m_cnt( 0 ) {}
      bool Push( const T &data );
      bool Pop(T &data);
      bool Get(T &data);
};
```

Здесь сначала объявлен класс протокола стека, затем от него наследуется класс стека на основе массива фиксированного размера (реализация ничем не отличается от приведенной выше, поэтому не приводится). Ниже описан еще один производный класс стек, работа которого основана на односвязной списковой структуре данных.

```
template <class T>
class TDynamicStack: public TAbstractStack<T>
{
    struct list{
        T data;
        list* link;
    } *m_head;
  public:
    TDynamicStack() {  m_head = NULL;  }
    ~TDynamicStack() {
         list *p;
```

```
        while (p = m_head) { m_head = p->link; delete p; }
    }
    bool Push( const T &data ) {
        list* p = new list;
        p->data = data;
        p->link = m_head;
        m_head = p;
        return true;
    };
    bool Pop( T &data );
    bool Get(T &data) {
          if (! m_head)  return false;
          data = m_head->data;
          return true;
      };

};
```

На практике часто возникает необходимость выноса реализации какого-либо метода за границу класса. Вынесение реализации за границы описания шаблона класса выглядит следующим образом:

```
template <class T>
bool TDynamicStack<T>::Pop( T &data)
{
    if (! m_head)  return false;
    list* p = m_head;
    m_head = p->link;
    data = p->data;
    delete p;
    return true;
}
```

Следует отметить, что в случае указания всех параметров шаблона при наследовании от него может быть унаследован обычный класс.

1.2 Специализация шаблонов класса и методов

В том случае, если реализация шаблона для каких-то типов данных оказывается неэффективной или некорректной, выполняется специализация (специальная реализация). Она может быть выполнена для шаблона класса целиком или для отдельного метода класса.

При специализации всего класса, после описания базового варианта класса размещается полное описание специализированного класса. Например.

```
template <>
class TDynamicStack<char*>: public TAbstractStack<char*>
{
    struct list{
        char* data;
        list* link;
    } *m_head;
  public:
    TDynamicStack() {  m_head = NULL;  }
    ~TDynamicStack() {
         list *p;
         while (p = m_head) { m_head = p->link; delete p; }
    }
    bool Push( char* const &data ) {
         list* p = new list;
         p->data = strdup(data);
         p->link = m_head;
         m_head = p;
         return true;
    };
    bool Pop( char* &data ) {
        if (! m_head)  return false;
        list* p = m_head;
        m_head = p->link;
        data = p->data;
        delete p;
        return true;
    }
    bool Get(char* &data) {
```

```
          if (! m_head)  return false;
          data = strdup(m_head->data);
          return true;
      };
};
```

При специализации отдельного метода размещают специальную реализацию метода. Например.

```
template <>
bool TDynamicStack<char*>::Push( char* const &data ) {
         list* p = new list;
         p->data = strdup(data);
         p->link = m_head;
         m_head = p;
         return true;
    };
```

2 Пример выполнения лабораторной работы

```
#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;

template <class T>
class CCollectionOfPointers{
     T** m_arr;
     int m_cnt;
     bool m_ownsPointers;
public:
     CCollectionOfPointers(int Count, bool ownsPointers){
          m_cnt = Count;
          m_ownsPointers = ownsPointers;
          m_arr = new T*[m_cnt];
          for(int i=0; i<m_cnt; i++)
               m_arr[i] = NULL;
     }
     ~CCollectionOfPointers(){
          if(m_ownsPointers){
               for(int i=0; i<m_cnt; i++)
                    if(m_arr[i] != NULL)delete m_arr[i];
          }
          delete [] m_arr;
     }
     T*  operator[] (int n) const{
```

```
          if(n<0 || n>=m_cnt) throw EAccessViolation((void*)
&m_arr[n]);
          return m_arr[n];
     }
     T*& operator[] (int n){
          if(n<0 || n>=m_cnt) throw EAccessViolation((void*)
&m_arr[n]);
          return m_arr[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");

     CCollectionOfPointers<int> intList(10, false);
     intList[0] = &argc;

     EBaseError* e = NULL;
     CCollectionOfPointers<EBaseError> errList(10, true);
     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. Каким образом определяется шаблон класса? Объясните назначение шаблонов в C++.
2. Каким образом создать экземпляр шаблона класса?
3. Как описать функцию-член шаблона класса вне объявления шаблона класса.
4. Каким образом выполняется специализация шаблона класса и метода?

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 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 №4.

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 **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

2. Написать программу, в которой описана иерархия классов: средство передвижения (**велосипед**, **автомобиль, грузовик**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

3. Написать программу, в которой описана иерархия классов: человек (**«дошкольник»**, **«школьник», «студент», «работающий»**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

4. Написать программу, в которой описана иерархия классов: ошибка в программе (**«недостаточно памяти»**, **«ввода/вывода», «ошибка чтения файла», «ошибка записи файла»**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

5. Написать программу, в которой описана иерархия классов: ошибка в программе (**«ошибочный указатель»**, **«ошибка работы со списком», «недопустимый индекс», «список переполнен»**). Описать **шаблонный класс** для хранения массива указателей на объекты

произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

6. Написать программу, в которой описана иерархия классов: ошибка в программе (**«недостаточно привилегий»**, **«ошибка преобразования», «невозможно преобразовать значение», «невозможно привести к интерфейсу»**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

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 №4. 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 **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

8. Написать программу, в которой описана иерархия классов: геометрические фигуры (**эллипс, квадрат, трапеция**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

9. Написать программу, в которой описана иерархия классов: геометрические фигуры (**ромб, параллелепипед, эллипс**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]».

Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

10. Написать программу, в которой описана иерархия классов: геометрические фигуры (**куб, цилиндр, тетраэдр**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

11. Написать программу, в которой описана иерархия классов: геометрические фигуры (**конус, шар, пирамида**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

12. Написать программу, в которой описана иерархия классов: числа (**целое, вещественное, комплексное**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

13. Написать программу, в которой описана иерархия классов: **треугольник** (**равнобедренный, равносторонний, прямоугольный**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

14. Написать программу, в которой описана иерархия классов: **прогрессия** (**арифметическая, геометрическая**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

15. Написать программу, в которой описана иерархия классов: геометрические фигуры (**круг, параллелепипед, трапеция**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

16. Написать программу, в которой описана иерархия классов: геометрические фигуры (**эллипс, квадрат, треугольник**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]».

Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

17. Написать программу, в которой описана иерархия классов: геометрические фигуры (**шар, цилиндр, пирамида**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

18. Написать программу, в которой описана иерархия классов: геометрические фигуры (**куб, конус, тетраэдр**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

19. Написать программу, в которой описана иерархия классов: геометрические фигуры (**ромб, прямоугольник, эллипс**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

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 №4. 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 **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Описать класс-**итератор** для итерации по элементам коллекции.

Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

21. Написать программу, в которой описана иерархия классов: функция от одной переменной (**синус**, **косинус**, **тангенс**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Описать класс-итератор для итерации по элементам коллекции. Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

22. Написать программу, в которой описана иерархия классов: функция от одной переменной (**секанс**, **косеканс**, **котангенс**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Описать класс-итератор для итерации по элементам коллекции. Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

23. Написать программу, в которой описана иерархия классов: функция от одной переменной (**арксинус**, **арккосинус, а также класс, необходимый для представления производных**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Описать класс-итератор для итерации по элементам коллекции. Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

24. Написать программу, в которой описана иерархия классов: функция от одной переменной (**арктангенс**, **арккотангенс, а также класс, необходимый для представления производных**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Описать класс-итератор для итерации по элементам коллекции. Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

25. Написать программу, в которой описана иерархия классов: функция от одной переменной (**логарифм**, **натуральный логарифм, а также класс, необходимый для представления производных**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Описать класс-итератор для итерации по элементам коллекции. Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

26. Написать программу, в которой описана иерархия классов: функция от одной переменной (**экспонента**, **гиперболический синус, гиперболический косинус**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Описать класс-итератор для итерации по

элементам коллекции. Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

27. Написать программу, в которой описана иерархия классов: функция от одной переменной (**степенная**, **показательная**). Описать **шаблонный класс** для хранения массива указателей на объекты произвольного класса, в шаблонном классе перегрузить операцию «[ ]». Описать класс-итератор для итерации по элементам коллекции. Продемонстрировать работу операторов и использование шаблонного класса с различными классами.

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.