МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МУРМАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Б.О.09.02 Объектно-ориентированное программирование
(код и наименование дисциплины)

для направления подготовки 09.03.01 Информатика и вычислительная техника
(код и наименование направления подготовки /специальности)

направленности/профиля Программное обеспечение вычислительной техники и автоматизированных систем
(наименование направленности (профиля) образовательной программы)

Кафедра-разработчик: цифровых технологий, математики и экономики
(наименование кафедры-разработчика рабочей программы)

Мурманск
2021

Составитель - Лясникова Светлана Михайловна, доцент кафедры цифровых технологий, математики и экономики

Методические указания по освоению дисциплины рассмотрены и одобрены на заседании кафедры-разработчика
цифровых технологий, математики и экономики
название кафедры

21.06.2021 протокол № 12
дата

ОГЛАВЛЕНИЕ

Общие организационно-методические указания

Обязательный минимум содержания дисциплины:

программирование в классах, роли класса, синтаксис класса, класс как тип данных, конструкторы, деструкторы, дружественные функции, структуры, отношения между классами, наследование, абстрактные классы, простое наследование, множественное наследование, виртуальные функции, виртуальные базовые классы, шаблоны функций и шаблоны классов, обработка исключительных ситуаций.

Целью дисциплины «Объектно-ориентированное программирование» является изучение основ классической теории объектно-ориентированного программирования: инкапсуляция, полиморфизм, абстракция, наследование, понятие класса, объекта, отношения между классами и др.

Задачей дисциплины «Объектно-ориентированное программирование» является изучение технологии объектно-ориентированного подхода в инструментальных языках и формирование практических навыков использования объектно-ориентированного программирования.

В результате изучения дисциплины студенты должны:

- знать и уметь использовать:

основы технологии объектно-ориентированного программирования; особенности построения объектно-ориентированных программных средств.

уметь реализовать принципы ООП при разработке программных средств; создавать диаграммы классов.

владеть приемами объектно-ориентированного решения различных задач.

Тематический план дисциплины

Таблица 1. Содержание учебной дисциплины

№ п\п	**Содержание разделов (модулей), тем дисциплины**	Количество часов, выделяемых на виды учебной подготовки	
		Лекции	ЛР
1	2	3	4
	II семестр		
Тема 1. Введение в ООП. Классы и ООП			
1	Для чего нужно ООП. Недостатки процедурного подхода.Объектно-ориентированный подход. Свойства ООП. Классы и ООП. Роли класса.Синтаксис класса. Поля класса. Методы класса. Методы свойства. Встроенное определение методов класса. Внешнее определение методов класса. Конструкторы. Создание и использование объектов. Деструкторы. Конструктор копирования. Указатель this. Константные поля. Статические элементы класса.	4	4
Тема 2. Перегрузка операций			
1	Ограничения на перегрузку. Прототип функции-операции. Перегрузка внешними функциями. Перегрузка операций методами класса.	2	4
Тема 3. Управляемый и неуправляемый код и данные			
1	Управляемый код и данные. Сборщик мусора. Ссылочные типы и типы значения.	0	0
Тема 4. Дружественные функции и классы			
1.	Дружественные функции: основные свойства, номенклатура. Ситуации, определяющие их необходимость и полезность. Наборы дружественных функций, дружественные классы. Объявление дружественной функции: синтаксис, размещение, семантика, требования к параметрам и типу возвращаемого значения. Размещение определения. Правила задания областей видимости. Вызов функции.	2	2
Тема 5. Отношения между классами. Простое наследование.			
1.	Отношения между классами. Отношения «является» и «имеет». Наследование: две роли. Простое наследование. Наследование членов базового класса в производном классе. Доступ к элементам базового класса в классе наследнике. Конструкторы, деструкторы и наследование. Порядок вызова конструкторов и деструкторов. Поля и методы при наследовании. Вложенные классы и наследование. Принцип подстановки. Закрытое наследование.	4	6
Тема 6. Виртуальные функции			
1	Зачем нужны виртуальные функции. Раннее (статическое) связывание. Определение виртуальных функций. Два типа полиморфизма в с++: статический полиморфизм, динамический полиморфизм. Правила описания и использования виртуальных функций. Переопределение и перегрузка виртуальных функций. Виртуальные функции в конструкторах и деструкторах. Чистые виртуальные функции. Виртуальные деструкторы.	4	0

	Чистые виртуальные деструкторы. Виртуализация внешних функций.		
Тема 7. Множественное наследование			
1	Множественное наследование. Принцип подстановки при открытом множественном наследовании. Принцип подстановки при открытом множественном наследовании. Виртуальное наследование. Принцип доминирования. Финальный класс. Размеры классов при множественном наследовании.	2	0
Тема 8. RTTI			
1	Составляющие механизма динамической идентификации типа. Типы преобразований. typeid().Класс type_info. Перекрестные ссылки. Мультиметоды. Двойной диспетчер для двух типов.	2	2
Тема 9. Шаблоны функций и классов			
1	Шаблоны функций: примеры. Параметры по умолчанию. Перегрузка шаблонов функций. Специализация шаблона функции. Определение шаблона класса. Объявление объекта класса. Внешнее определение методов. Параметры шаблона класса по умолчанию. Параметры шаблона – не типы. Специализация: частичная и полная. Ограничения. Поле-шаблон. Метод-шаблон. Параметр-шаблон. Шаблоны и наследование. Шаблоны и дружественность.	4	2
Тема 10. Диаграммы классов			
1	UML. Диаграмма классов. Изображение класса. Описание атрибута. Операции. Виды отношений между классами. Ассоциация. Агрегация. Композиция. Наследование. Зависимость.	2	0
Тема 11. Обработка исключительных ситуаций			
1	Принципы обработки исключений. Генерация исключений. Перехват и обработка исключений. Спецификация исключений. Исключения и конструкторы. Исключения и деструкторы. Стандартные исключения.	2	4
Тема 12. Стандартная библиотека шаблонов STL			
1	Библиотека STL. Шаблон vector. Возможности vector. Итераторы. Полезные методы. Последовательные контейнеры. Различные наборы операций для последовательных контейнеров. Инициализация контейнеров. Алгоритмы. Виды итераторов. Ассоциативные контейнеры.	2	4
Тема 13. Введение в паттерны проектирования			
	Основные элементы паттерна проектирования: имя, задача, решение, результаты. Классификация паттернов. Паттерн Singleton. Структура Singleton. Достоинства Singleton. Реализация Singleton.	1	2
Тема 14. Рефакторинг			
	Определение рефакторинга. Цель рефакторинга. Принципы рефакторинга.	1	0
Тема 15. Умные указатели			
	Умные указатели. Семантика перемещения. unique_ptr, weak_ptr, auto_ptr.	2	4
	Итого:	34	34

Перечень лабораторных занятий

Таблица 2. Перечень лабораторных работ

№ п/п	Наименование и содержание лабораторных работ	№ темы по таблице 1	Количество часов
1	2	3	4
4 семестр			
1	Разработка программы «Построение графика» средствами ООП	1	4
2	Перегрузка операций	2	2
3	Дружественные функции	5	2
4	Разработка класса полином	1, 2, 5	2
5	Разработка классов квадратная и прямоугольная матрица	6	6
6	Двойной диспетчер для двух типов	7,9	2
7	Шаблоны классов	10	2
8	Обработка исключений	12	4
9	Стандартная библиотека шаблонов STL	13	4
10	Семинар «Класс auto_ptr»	15	4
11	Семинар «Паттерн Singleton»	14	2
Итого			34

Методические указания к выполнению лабораторных работ

Лабораторная работа 1,2

Разработка программы «Построение графика» средствами ООП

Задача

Имеется бинарный файл, в котором хранятся значения типа int (4 байта) (файл samples). Необходимо, считывать эти значения и выводить результат в виде графика на экран. Таким образом, по оси *x* графика – номер отсчета, по оси *y* – значение из файла samples.

Учесть, что бинарный файл может быть большого размера, поэтому необходимо организовать простую навигацию по файлу: предусмотреть кнопки «вперед» и «назад». Нажатие на кнопку «вперед» сдвигает график на один отсчет вперед. Нажатие на кнопку «назад» сдвигает график на один отсчет назад. Реализовать два класса:

1. Класс **ReadFile** – для считывания отсчетов из бинарного файла.
2. Класс **Plot** – для построения графика по считанным отсчетам.

Средства разработки:
Qt Creator, C++.

Ход работы

1. Создать новый проект в среде Qt Creator, GUI приложение; Назвать проект «MainReview».
2. Использовать следующие компоненты для формы (Рис.1):

- QGraphicsView (для отображения графика);
- 2 QPushButton;
- Action (выпадающее меню для выбора директории с файлом).

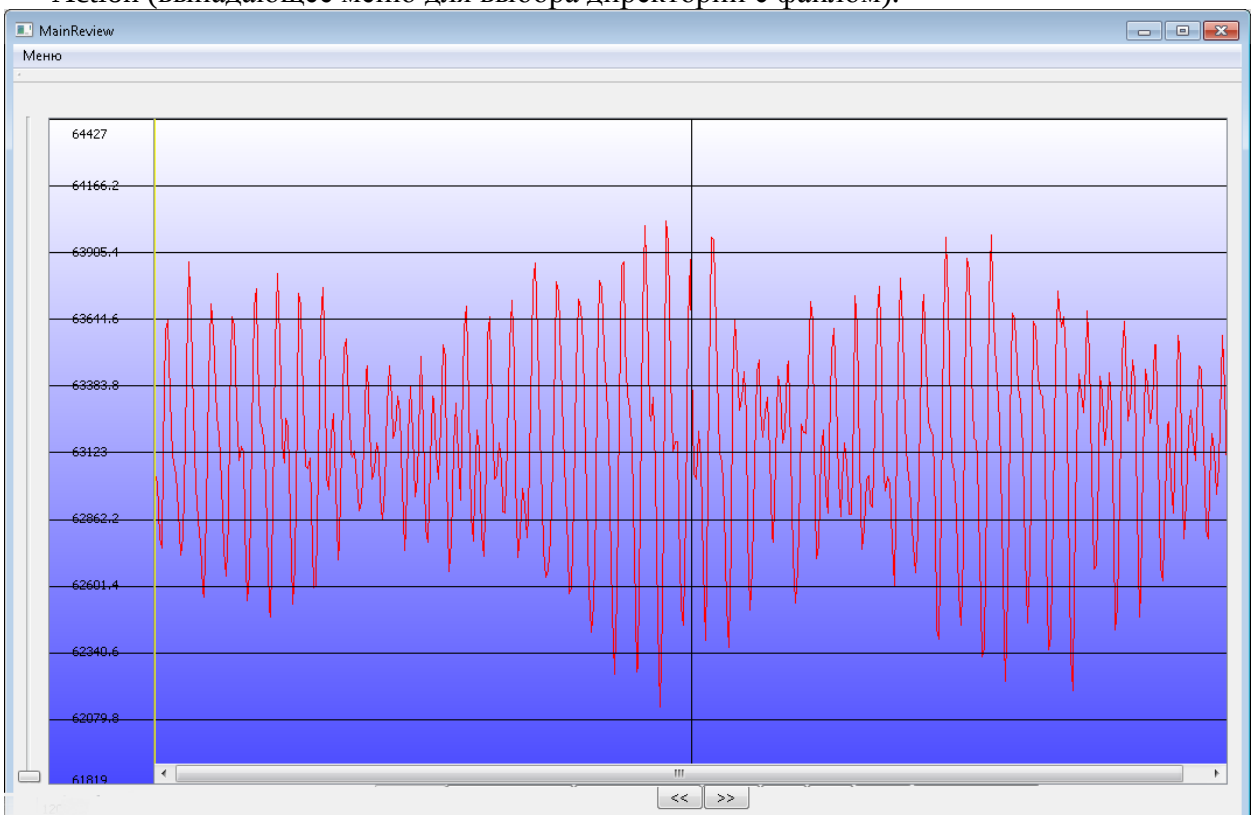

Рисунок 1 – Примерный интерфейс программы

3. Для создания нового класса нажмите правой кнопкой мыши по проекту и выберите пункт меню «Добавить новый»-> «Класс C++».

4. Разрабатываем класс для рисования графика (файл plot.h):

```
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QObject>
#include <QPen>
class Plot
{
    public:
  //конструктор
  Plot()
  {
  }
  //деструктор
 ~Plot()
  {
  }
   private:
    //закрытые поля класса и методы класса
   public:
    //открытые методы класса

};
```

5. Разрабатываем класс для чтения данных из бинарного файла:

```
#include <QString>
#include <QDataStream>
#include <QFile>

class ReadFile
{
    public:
         ReadFile()
         {};
         ~ReadFile()
         {};
// следующие два метода корректны при условии, //если весь файл
считывается сразу же в некоторый //массив. Вам нужно переработать
эти методы, чтобы //считывалось количество значений на ширину экрана
//(вначале, при загрузке файла), а потом при //перемещении по
кнопкам «вперед» «назад» считывалось //по одному значению.
Используйте file.seek().
       //метод переработать!
       void readStream(QDataStream &stream,  …)
       {
          stream.setVersion(QDataStream::Qt_2_0);
                  stream.setByteOrder(QDataStream::LittleEndian);
         ……..
          int n;
          //в переменную n (4 байта)считываем нужное нам //значение
          while(!stream.atEnd())
          {
                n = 0;
                stream >> n;
```

```
            ……….
        }
        }
        //метод необходимо переработать
        void Readf(QString fn,…)
        {
              QFile file(fn);
              if (file.open(QIODevice::ReadOnly))
              {
                  QDataStream stream(&file);
                  readStream(stream);

              }
              else if (!file.open(QIODevice::ReadOnly))
              {
                 …..
              }
              ……….
         }
};
```

5. Для загрузки файла используйте следующее:

```
QString dir = QFileDialog::getOpenFileName();
 dir.replace(QString('/'), QString("\\"));
```

6. Написать программу, протестировать, оформить отчет о проделанной работе.

Лабораторная работа 3

«Перегрузка операций»

Ограничения на перегрузку

Запрещена перегрузка операций:

1. .
2. .*
3. ?:
4. ::
5. Sizeof
6. #

Эти операции разрешается перегружать только методами класса:

1. =
2. ()
3. [ ]
4. ->

Прототип функции-операции

```
тип operator@(список параметров);
```

Перегрузка операций методами класса

- Унарная операция

```
тип operator@()
```

- Бинарная операция

`тип operator@(par1)`
Перегрузка внешними функциями
Перегрузка внешними функциями
`тип operator @ (par1, par2)`
Обращение к операции
`par1@par2` (инфиксная форма)
`operator@(par1, par2)` (функцион. форма)

Задача 1

Создать класс Matrix для работы с матрицами. Сделать перегрузку операций: +, *, []. Написать программу, протестировать, оформить отчет о проделанной работе.
Отчет должен содержать: краткие теоретические сведения, посвященные перегрузке операций, класс с комментариями, результаты тестирования программы.

Лабораторная работа 4

«Дружественные функции»

Свойства дружественных функций

- не связана с объектом и не является членом класса для которого она дружественна;
- задается и вызывается так же как и обычная глобальная функция;
- при вызове не получает указателя this;
- объявляется внутри класса к элементам которого ей нужен доступ;
- может быть обычной функцией или методом ранее определенного класса.

Задание 1

Реализовать класс точка, который описывает точку на плоскости. Закрытые поля класса: имя точки, координата x, координата y. Реализовать методы для перемещения точек на плоскости. Реализовать дружественную функцию, которая определяет расстояние между двумя заданными точками.

Лабораторная работа №5

«Разработка класса полином»

Задача 1

Создать описание класса многочленов от одной переменной, задаваемых степенью многочлена и массивом коэффициентов. Предусмотреть методы для вычисления значения многочлена для заданного аргумента, операции сложения, вычитания и умножения многочленов с получением нового объекта-многочлена, вывод на экран многочлена.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

Лабораторная работа №6,7,8

«Разработка классов квадратная и прямоугольная матрица»

Задача

Разработать класс «Прямоугольная матрица». Определить в классе 3 конструктора – это конструктор без аргументов; конструктор с двумя параметрами, задающими размерность матрицы и инициализирующий элементы матрицы нулями; конструктор копирования. Определите деструктор. Сделать перегрузку операций – сложения двух матриц (+), вычитания двух матриц (-), умножения двух матриц, умножения матрицы на число (*) , присваивания (=), получение строки матрицы с заданным номером ([]), операцию транспонирования матрицы (!). Написать две дружественные функции для ввода и вывода матрицы.

Так как квадратная матрица («является») частный случай прямоугольной матрицы, то реализуйте класс «Квадратная матрица» как наследник класса прямоугольная матрица. Добавьте в этот класс методы, которые можно использовать только для квадратной матрицы – получение обратной матрицы и вычисление детерминанта.

Помните применение методов базового класса для выполнения арифметических операций с квадратными матрицами связано с преобразованием типов. Например, метод для сложения матриц в базовом класса должен в качестве параметра получить ссылку на объект класса «Прямоугольная матрица» и возвратить объект этого типа. Соответственно применение этого метода для сложения квадратных матриц связано с двумя преобразованиями. Учите, что результатом работы метода должна быть квадратная матрица (т.е. нужно осуществить понижение типа - из типа базово класса к производному). Дружественные функции ввода и вывода переопределять не нужно (объясните в отчете почему).

Отчет должен содержать теоретические сведения, разработанные классы с комментариями, тестирование программы.

Лабораторная работа № 9

«Двойной диспетчер для двух типов»

Задача 1

Разберите представленный код программы, т.е. опишите, что делает данная программа. Добавьте в предложенную ниже программу третий класс и покажите механизм использования оператора **typeid()** для распознавания типов. Не изменяйте код представленный ниже!

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include "typeinfo"
using namespace std;
class Base
{
   public:
       virtual ~Base()=0;
   virtual bool Operator (Base &R)=0;
};
Base:: ~Base(){}
class Derived_a: public Base
{
     public:
  virtual bool Operator (Base &R);
};
class Derived_b: public Base
{
     public:
virtual bool Operator (Base &R);
};
bool Derived_a::Operator (Base &R)
{
    if (Derived_a *pa =dynamic_cast<Derived_a *>(&R))
```

```
    {
        cout<<"A_A"<<endl;
        return true;
    }
  else if (Derived_b *pb =dynamic_cast<Derived_b *>(&R))
  {
        cout<<"A_B"<<endl;
        return true;
  }
 }
bool Derived_b::Operator (Base &R)
{
      if (Derived_a *pa = dynamic_cast<Derived_a *>(&R))
      {
          cout<<"B_A"<<endl;
          return true;
      }
    else if (Derived_b *pb =dynamic_cast<Derived_b *>(&R))
    {
          cout<<"B_B"<<endl;
          return true;
    }
}
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Derived_a A;
    Derived_b B;
    cout<<A.Operator(A)<<endl;
    cout<<A.Operator(B)<<endl;
    cout<<B.Operator(A)<<endl;
    cout<<B.Operator(B)<<endl;
    return a.exec();
}
```

Задача 2

Разберите представленный код программы, т.е. опишите, что делает данная программа.

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include "typeinfo"
using namespace std;
class Derived_a;
class Derived_b;
class Base
{
   public:
       virtual ~Base()=0;
       virtual bool Operator (Base &R)=0;
       virtual bool Operator (Derived_a &R)=0;
       virtual bool Operator (Derived_b &R)=0;
};
Base:: ~Base(){}
class Derived_a: public Base
{
     public:
     virtual bool Operator (Base &R)
     {return R.Operator(*this); }
     virtual bool Operator (Derived_a &R)
     {
         cout<<"A_A"<<endl;return true;
     }
     virtual bool Operator (Derived_b &R)
     {
        cout<<"A_B"<<endl;return true;
     }
```

```
};
class Derived_b: public Base
{
     public:
     virtual bool Operator (Base &R)
     {return R.Operator(*this);}
     virtual bool Operator (Derived_a &R)
     {
         cout<<"B_A"<<endl;return true;
     }
     virtual bool Operator (Derived_b &R)
     {
        cout<<"B_B"<<endl;return true;
     }
};
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Derived_a A;
    Derived_b B;
   // cout<<A.Operator(A)<<endl;
   // cout<<A.Operator(B)<<endl;
   // cout<<B.Operator(A)<<endl;
   //cout<<B.Operator(B)<<endl;
    Base &D = A;
    cout<<A.Operator(D)<<endl;

    return a.exec();
}
```

Добавьте третий класс `Derived_c` (не изменяйте код представленный выше!) и в main() выполните следующий фрагмент программы:

```
Derived_a A;
Derived_b B;
Derived_c C;
Base &tempA = A;
Base &tempB = B;
Base &tempC = C;
cout<<A.Operator(C)<<endl;
cout<<B.Operator(C)<<endl;
cout<<tempA.Operator(tempC)<<endl;
cout<<tempB.Operator(tempC)<<endl;
```

Объясните полученные результаты.

Отчет должен содержать: краткие теоретические сведения, листинг программ, примеры работы программ, пояснения полученных результатов.

Лабораторная работа № 10

Шаблоны классов

Цель работы: получить навык работы с шаблонами классов.

Задача 1

```
#include <cstddef>
```

```
template<typename T, std::size_t N>
class MyArray
{
    T arr[N];
public:
    MyArray(const T &t)
    {
        for(int i=0; i<N; i++)
            arr[i] = t;
    }

    size_t getSize()
    {
        return N;
    }

  T& operator[](const size_t i)
    {
        if (i<getSize())
            return arr[i];

    }

};
```

Дописать шаблонный класс MyArray, который можно использовать вместо встроенных массивов:

а) необходимо разрешить присвоение меньшего массива большому как по размеру, так и по типу. Для этого реализуйте операцию присваивания, в которой аргументы правого и левого типов представляют собой переменные MyArray разных типов и размеров. Таким образом, необходимо определить метод-шаблон. Прототип метода может выглядеть так:

```
template <typename T1, size_t N1>
MyArray<T1,N1>& operator=(MyArray<T1,N1>& rhs);
```

Замечание: можно воспользоваться для преобразования типа `static_cast<T> (...)`

б) реализуйте дружественную функцию вывода элементов массива на экран. Определите эту функцию внутри шаблонного класса. *Замечание*: дружественная функция определенная внутри шаблонного класса не является функцией шаблоном. Это обычная функция несмотря на то что её аргументы зависят от параметра шаблона. Зависимость дружественной функции от аргументов шаблона является обязательным требованием при внутреннем определении дружественной функции. Если это не учесть, то при каждом инстанцировании шаблона сгенерируются одинаковые определения и возникнет ошибка повторного определения функции.

Задача 2

В одной из предыдущих работ Вы создавали класс Квадратная Матрица. Используя предыдущие наработки, создайте шаблон класса Квадратная Матрица, в котором в качестве обобщенного типа данных выступает тип элемента матрицы.

```
template<class T> class Matrix
{
```

```
   private:
       T** a; //массив элементов
       int N; //размерность матрицы
//……..
};
```

Максимальное количество баллов:5.

Лабораторная работа № 11, 12

«Обработка исключений»

Цель работы: научиться обрабатывать исключения.

Задача

В одной из предыдущих работ Вами разрабатывались классы «Матрица» и «Квадратная матрица». При этом исключительные ситуации могли возникнуть в следующих случаях:

1. Пользователь случайно или нарочно ввел неположительные значения строк или столбцов;
2. Обращение к элементу матрицы происходит по некорректным индексам;
3. При сложении(вычитании) двух матриц их размеры не совпадают;
4. При умножении матриц количество столбцов первой матрицы не совпадает с количеством строк второй матрицы;
5. При вычислении обратной матрицы исходная матрица вырождена (определитель равен 0).

Обработайте данные исключительные ситуации в своей программе.

- Используйте подход, основанный на иерархии классов исключений. Сделайте **базовый класс абстрактным**.
- Определите в базовом классе **чисто виртуальную функцию**, предназначенную **для вывода сообщения об ошибке**.
- Для каждого вида исключительной ситуации создайте свой класс производный от базового, в котором переопределите виртуальную функцию для вывода сообщения об ошибке.

Замечание: можно конечно обрабатывать каждый вид исключения своим обработчиком catch, но лучше сделать один обработчик (подумайте как и объясните почему! При этом catch(…) не использовать).

Постройте диаграмму классов исключений.

Лабораторная работа № 13, 14

«Стандартная библиотека шаблонов STL»

Задача 1

Известно, что алгоритм sort не работает со списками. Поэтому для списков определена функция-член sort. Также у списков существует функция-член unique, которая удаляет повторяющиеся последовательные элементы. Создайте список, заполните его некоторыми значениями и продемонстрируйте работу sort и unique.

Решение

```
list<int> t;
t.push_back(23);
t.push_back(44);
t.push_back(23);
t.push_back(23);
t.push_back(78);
t.push_back(23);
t.unique();
t.sort();
list<int>::iterator i;
for(i=t.begin(); i!=t.end(); i++)
{
    cout<<*i<<endl;
}
```

Удалите из полученного списка заданный пользователем элемент. Воспользуйтесь для этого функцией-членом remove (заметьте у вектора и двусторонней очереди такой функции-члена нет).

Класс list содержит функцию член merge. Разберитесь для чего используется эта функция и приведите пример её использования.

Задача 2

Создайте вектор A. Пусть A состоит из 4 элементов каждый из которых является вектором содержащим целые числа 0 и 1. Данный вектор A можно считать двумерным массивом или матрицей. Выведите элементы A на экран**.**

Решение

```
vector <int>v;
v.push_back(0);
v.push_back(1);
vector<vector<int> > A;
for(int i=0; i<4; i++)
     A.push_back(v);
for(int i=0; i<4; i++)
{
    for(int j=0; j<2; j++)
        cout<<A[i][j]<<" ";
    cout<<endl;
}
```

Задача 3

Используйте вместо двумерных массивов (как Вы это сделали в задаче 2) массивы указателей. Создайте вектор указателей A. Каждый элемент вектора A, A[i] должен являться указателем на вектор из двух элементов целого типа. Доступ к этим элементам может осуществляться так: *(A[i])[j]. Выведите элементы A на экран. Не забудьте после завершения работы освободить память.

Решение

```
vector<vector<int>* > A;
int b[] = {0,1};
for(int i=0;i<4;i++)
   A.push_back(new vector<int>(b,b+2));
for(int i=0; i<4; i++)
{
```

```
        for(int j=0; j<2; j++)
            cout<<(*A[i])[j]<<" ";
        cout<<endl;
    }
    for(int i=0;i<4;i++)
        delete A[i];
```

Задача 4

Напишите функцию для нахождения пересечения двух множеств *A* и *B*. Прототип функции может иметь вид:

```
typedef set<int, less<int> > type_set;

type_set operator*(const type_set& A, const type_set& B);
```

Для пересечения множеств используйте алгоритм set_intersection. Пример использования данного алгоритма приведен ниже (I – это результат пересечения множеств):

```
set_intersection(A.begin(), A.end(),
      B.begin(),B.end(),inserter(I,I.begin()));
```

Поясните, что такое `inserter(I,I.begin())`.

Решение

```
#include <QtCore/QCoreApplication>
#include "iostream"
#include <set>
#include <algorithm>
using namespace std;

typedef set<int, less<int> > type_set;

type_set operator*(const type_set& A, const type_set& B)
{
    type_set I;
    set_intersection(A.begin(), A.end(),
      B.begin(),B.end(),inserter(I,I.begin()));
    return I;
}
int main(int argc, char *argv[])
{
   QCoreApplication a(argc, argv);
   int s1[]={7,45};
   int s2[]={3,45,2,5,7};
   type_set A(s1,s1+2);
   type_set B(s2, s2+5);
   type_set I;
   I = A*B;
   type_set::iterator i1;
   for(i1=I.begin(); i1!=I.end();i1++)
   {
       cout<<*i1<<endl;
   }
  return a.exec();
}
```

Задача 5

Напишите функцию для нахождения объединения двух множеств *A* и *B*. Прототип функции может иметь вид:

```
typedef set<int, less<int> > type_set;

type_set operator+(const type_set& A, const type_set& B);
```

Для объединения множеств используйте алгоритм set_union. Пример использования данного алгоритма приведен ниже (I – это результат объединения множеств):

```
set_union(A.begin(), A.end(),
       B.begin(),B.end(),inserter(I,I.begin()));
```

Решение

```
#include <QtCore/QCoreApplication>
#include "iostream"
#include <set>
#include <algorithm>
using namespace std;

typedef set<int, less<int> > type_set;

type_set operator+(const type_set& A, const type_set& B)
{
    type_set I;
    set_union(A.begin(), A.end(),
      B.begin(),B.end(),inserter(I,I.begin()));
    return I;
}
int main(int argc, char *argv[])
{
   QCoreApplication a(argc, argv);
   int s1[]={7,45};
   int s2[]={3,45,2,5,7};
   type_set A(s1,s1+2);
   type_set B(s2, s2+5);
   type_set I;
   I = A+B;
   type_set::iterator i1;
   for(i1=I.begin(); i1!=I.end();i1++)
   {
       cout<<*i1<<endl;
   }
  return a.exec();


}
```

Лабораторная работа № 15, 16

Семинар «Класс auto_ptr»

В STL определен специальный класс, который делает более безопасным использование оператора new в сложных случаях.

Задание

Изучите класс auto_ptr. Подготовьте презентацию с примерами использования auto_ptr.

Задача 1

Изучите приведенный ниже код.

```
#include <QtCore/QCoreApplication>
#include <memory>
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    auto_ptr<double> aup1(new double), aup2;
    *aup1 = 5.8;
    cout<<*aup1;
    aup2 = aup1;
    return a.exec();
}
```

Выясните ответ на следующий вопрос: Почему нельзя использовать выражение *aup1 после присваивания aup2 = aup1. Попробуйте вывести после данного присваивания *aup1, как повела себя программа?

Задача 2

Почему приведенный ниже код является некорректным?

```
auto_ptr<double> aup1(new double[1000]);
```

Можно ли создать эквивалент auto_ptr для динамических массивов? Если да, то опишите, что для этого нужно сделать.

Задача 3

Проверьте работоспособность представленного ниже кода. Почему код является некорректным?

```
auto_ptr<string> m[2] =
{
    auto_ptr<string> (new string("Hi" )),
    auto_ptr<string> (new string("Hello" )) ,
};
auto_ptr<string> p(m[1]);

for(int i=0; i<2;i++)
       cout<<*m[i];
```

Задача 4

Почему третья строка приведенного ниже кода является некорректной. Как исправить данный код?

```
auto_ptr<double> aup1 ;
double  *aup2 =  new double ;
aup1 = aup2;
```

Краткое решение

```
aup1 = auto_ptr<double>(aup2);
```

Отчет должен содержать презентацию с примерами и ответы к приведенным выше задачам.

Лабораторная работа № 17

Семинар «Паттерн Singleton»

Задание

Изучите следующие вопросы:
1. Как решается проблема удаления объекта Singleton. Приведите пример.
2. Каких правил следует придерживаться при наследовании от класса Singleton. Приведите примеры программных решений.

Список цитируемой литературы

При разработке материалов практических занятий **использовались и цитировались** следующие источники:

1. Саттер, Герб Решение сложных задач на C++. Серия C++ In-Depth, т.4: Пер. с англ. – М.: Издательский дом «Вильямс», 2002. – 400 с.
2. Лаптев, В.В. C++.Объектно-ориентированное программирование: Учебное пособие. – Спб: Питер, 2008. – 464 с.
3. Леен Аммераль STL для программистов на C++. – М: ДМК, 1999. – 240 с.
4. Прата Стивен Язык программирования C++. Лекции и упражнения, 5-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2007. – 1184 с.
5. Вендоров А.М. Проектирование программного обеспечения экономических информационных систем. – М.: Финансы и статистика, 2002. – 349 с.
6. Павловская Т.А., Щупак Ю.А. C++. Объектно-ориентированное программирование: Практикум. – Спб: Питер, 2006. – 265 с.
7. Язык C++: Учеб. пособие/ И. Ф. Астахова, С.В. Власов, В.В Фертиков, А.В. Ларин. – Мн.: Новое знание, 2003. – 203 с.
8. Гамма Э., Джонсон Р., Хелм Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – М.: Питер, 2010. – 368 с.
9. http://qt-doc.ru/qthread-potoki-v-qt.html.
10. Стрикелева, Л. В. Программирование. Курс лекций (РФиКТ, БГУ (Минск), 1-ый курс), 2012, http://elib.bsu.by/handle/123456789/8654.
11. Андрианова А.А., Исмагилов Л.Н., Мухтарова Т.М. Объектно-ориентированное программирование на C++: Учебное пособие/А.А. Андриянова, Л.Н. Исмагилов, Т.М. Мухтарова. – Казань: Казансикй (Приволжского) федерального университет, 2010. – 230 с.

Рекомендуемая литература

- Антони Синтес Освой самостоятельно объектно-ориентированное программирование за 21 день. - М.: Вильямс, 2002.
- Гамма Э., Джонсон Р., Хелм Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – М.: Питер, 2010. – 368 с.
- Грэхем Иан Объектно-ориентированные методы. Принципы и практика = Object-Oriented Methods: Principles & Practice. - М.: Вильямс, 2004.
- Джосьютес Н. C++. Стандартная библиотека. Для профессионалов. – Спб: Питер, 2004. – 730 с.
- Иванова Г.С, Ничушкина Т.Н., Пугачев Е.К. Объектно-ориентированное программирование. -М.: Изд-во МГТУ им. Н.Э. Баумана, 2001.
- Лафоре Р. Объектно-ориентированное программирование в C++. – Питер, 2004.

- Павловская Т. А. C/C++. Программирование на языке высокого уровня. - ЗАО Издательский дом «Питер», 2003.
- Прата Стивен Язык программирования C++. Лекции и упражнения, 5-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2007. – 1184 с.
- Скотт Мейерс Эффективное использование STL. – Питер, 2002.
- Эрик Гамма, Ральф Джонсон, Ричард Хелм, Джон Влиссидес Приемы объектно-ориентированного проектирования. Паттерны проектирования. - Питер, 2007.
- Язык C++: Учеб. пособие/ И. Ф. Астахова, С. В. Власов, В.В. Фертиков, А. В. Ларин. – Мн: Новое знание, 2003. – 203 с.