МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ УКРАИНЫ
«КИЕВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ»

КЛАССЫ C++

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторной работе № 1

по дисциплине «ООП»

Киев 2016

СОДЕРЖАНИЕ

1 ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ

Цель работы – изучить основные концепции объектно-ориентированного программирования. Изучить особенности использования классов и объектов, а также особенности применения конструкторов и деструкторов.

2 ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Перед рассмотрением теории, непосредственно, относящейся к лабораторной работе рассмотрим некоторые необходимые возможности C++.

2.1. Встраиваемые функции (inline)

Реализация программы как набора функций хороша с точки зрения разработки программного обеспечения, но вызовы функций приводят к накладным расходам во время выполнения. В C++ для снижения этих накладных расходов на вызовы функций – особенно небольших – предусмотрены встраиваемые функции:

```
inline тип имя_функции (параметры)
```

Спецификация `inline` перед указанием типа результата в объявлении функции «советует» компилятору сгенерировать копию кода функции в соответствующем месте, чтобы избежать вызова данной функции. Это эквивалентно объявлению соответствующего макроса. В результате получается множество копий кода функции, вставленных в программу, вместо единственной функции, которой передаётся управление при каждом вызове функции. Компилятор может игнорировать спецификацию `inline`, что обычно и происходит для всех функций, кроме самых небольших. Недостатком встраиваемых функций можно назвать то, что при изменение кода функции может потребоваться перекомпиляция всех «потребителей» этой функции. Поэтому спецификацию `inline` целесообразно применять только для небольших и часто используемых функций. Использование `inline` функций может уменьшить время выполнения программы, но может увеличить её размер. Однако её использование предпочтительней объявления макросов, поскольку в данном случае остаётся возможность оптимизации кода компилятором.

Пример 1.

```
inline int fac(int n) { return (n<2)?1:n*fac(n-1);}
```


Хороший компилятор может сгенерировать константу 720 в месте вызова `fac(6)`, другой – `6*fac(5)`, а третий – обычный вызов `fac(6)`.

Использование `inline` функций недопустимо в блоках `goto`, `while`, `for`, `case`, рекурсивных вызовах и прочих сложных конструкциях.

2.2. Перегрузка функций

C++ позволяет определить несколько функций с одним и тем же именем, если эти функции имеют разные наборы параметров. Эта особенность называется перегрузкой функции. При вызове перегруженной функции компилятор C++ определяет соответствующую функцию путём анализа количества, типов и порядка следований аргументов в вызове. Перегрузка функции обычно используется для создания нескольких функций с одинаковым именем, предназначенных для выполнения сходных задач, но с разными типами данных. Например:

```
void print (int);               //печать целого
void print (const char*);   //печать символьной строки
```

Перегруженные функции различаются компилятором с помощи их сигнатуры – комбинации имени функции и типов её параметров. Компилятор кодирует идентификатор каждой функции по числу и типу её параметров, чтобы иметь возможность осуществлять надёжное связывание типов. Надёжное связывание типов гарантирует, что вызывается надлежащая функция и что аргументы согласуются с параметрами. Компилятор выявляет ошибки связывания и выдаёт сообщения о них.

Для различения функций с одинаковыми именами компилятор использует списки параметров. Перегруженные функции не обязательно должны иметь одинаковое количество параметров. Проверка осуществляется в следующем порядке:

– точное соответствие типов или соответствие, достигаемое тривиальными преобразованиями типов (имя массива и указатель);

– соответствие достигаемое преобразованием bool в int, char в int, float в double.

– соответствие, достигаемое путём неявных преобразований (int в double, double в int);

– соответствие, достигаемое при помощи преобразований, определяемых пользователем;

– соответствие за счёт многоточий в объявлении функций.

Не могут перегружаться функции, отличающиеся только типом возвращаемого значения.

Пример 2.

```
float sqrt(float);
double sqrt(double);
void f(double d, float f);
float fl=sqrt(f);              //вызов sqrt(double)
double do=sqrt(d);            //вызов sqrt(double)
fl=sqrt(f);                   //вызов sqrt(float)
do=sqrt(f);                   //вызов sqrt(float)
```

Если бы при разрешении использовался возвращаемый тип, было бы невозможно определить по вызову `sqrt()`, которую функцию в действительности вызывать.

2.3. nullptr

Раньше, для обнуления указателей использовался макрос NULL, являющийся нулем – целым типом, что, естественно, вызывало проблемы (например, при перегрузке функций). Ключевое слово `nullptr` имеет свой собственный тип `std::nullptr_t`, что избавляет нас от бывших проблем. Существуют неявные преобразования `nullptr` к нулевому указателю любого типа и к `bool` (как `false`), но преобразования к целочисленным типам нет.

Пример 3

```
#include "stdafx.h"
#include <cstdlib>
#include <iostream>

using namespace std;
void func(int a){
  cout <<" int a = "<< a << endl;
};
void func(const int* p){
  if (p){
          cout << " const int* p = " << *p << endl;
  }
  else { cout << " const int* p = nullptr" << endl; }

};
```


```
int main()
{
  int val = 6;
  int arr[5] = { 1, 2, 3, 4, 5 };

  func(3);      // вызов func(int a), верно
  func(0);      // вызов func(int a), верно
  func(NULL);      // вызов func(int a), некорректно, хотели вызвать
func(const int* p) с нулевым указателем
  func(nullptr);// вызов func(const int* p), верно
  func(&val);   // вызов func(const int* p), верно
  func(arr);    // вызов func(const int* p), верно
  system("pause");
}
```

Как видим в данном примере вызов `func(NULL);` приведет к некорректному результату, а вызов `func(nullptr);` будет корректным. Конечно, можно решить эту проблему явным приведением типов `func(static_cast<const int *>(NULL));`, но такое решение громоздко и неэлегантно.

2.4. Rvalue ссылки

Rvalue ссылка – это составной тип, очень похожий на традиционную ссылку в C++. Чтобы различать эти два типа, мы будем называть традиционную C++ ссылку lvalue ссылка.

По семантике lvalue ссылка формируется путём помещения & после некоторого типа.

```
int a;
int& b = a;  // это lvalue ссылка
```

Если после некоторого типа поместить &&, то получится rvalue ссылка.

```
int a;
int&& b = a;  // это rvalue ссылка
```

Rvalue ссылка ведет себя точно так же, как и lvalue ссылка, за исключением того, что она может быть связана с выражением, тогда как lvalue связать с не константным выражением нельзя.

```
int&  a = (3+6);  // ошибка!
int&& a = (3+6);  // допустимо
const int& a = (3+6);  // допустимо
```

Важно понимать, что под «выражением» здесь понимаются самые разные конструкции C++, а не только математические выражения.

Простейшим способом определения является попытка получения адреса выражения; если вы можете получить его адрес и в дальнейшем его

использовать, тогда перед вами lvalue. Если же адрес выражения не может быть получен - перед вами rvalue. Кто-то может поспорить, что адрес все-таки можно получить, и некоторые компиляторы, возможно, дают это сделать. Тем не менее, это является нарушением стандарта C++. Да и если рассуждать логически: этот адрес не будет иметь никакого смысла, так как это адрес памяти, которую вы не контролируете, и она может быть легко перезаписана в течение работы программы. Адрес же lvalue**,** это адрес постоянного хранилища, которое остается под контролем программиста на протяжении всей области жизни объекта.

Комбинация rvalue ссылок и lvalue ссылок – это то, что необходимо для лёгкой реализации семантики перемещения (move semantics). Rvalue ссылка может также использоваться для достижения идеальной передачи (perfect forwarding) (*будет рассмотрено в ЛР 7*), что ранее было нерешенной проблемой в C++. Для большинства программистов rvalue ссылки позволяют создать более производительные библиотеки.

2.5. Классы

Целью введения концепции классов в C++ является предоставление программисту средств создания новых типов, которые настолько же удобны в использовании, как и встроенные. Тип является конкретным представлением некоторой концепции. Например, встроенный тип `float` вместе с операциями +, -, * и т.д. представляет собой конкретное воплощение математической концепции вещественного числа.

Определение 1. Класс - разновидность абстрактного типа данных в объектно-ориентированном программировании, характеризуемый способом своего построения. Наряду с понятием «**объекта**» класс является ключевым понятием в ООП.

Определение 2. Класс – это определяемый пользователем тип. Класс должен быть объявлен до того, как будет объявлена хотя бы одна переменная этого класса.

Классы являются расширениями структур языка Си, дополненные "механизмами" скрытия данных, функциями элементами, наследованием и пр. Ниже показана взаимосвязь старых структур Си, новых структур С++ и классов:

```
Структура Си
typedef struct {
  int wd, ht;
} rect;
```

```
Структура С++
struct rect {
  int wd, ht;
};
```

```
Класс С++
class rect {
public:
  int wd, ht;
};
```

Конкретный экземпляр класса называется **объектом**.

Формат описания класса включает в себя:

– типы данных, являющихся элементами-данными класса (атрибутами);

– спецификациями доступа к данным;

– функции, определяющие методы обработки элементов-данных;

– конструкторы и деструктор.

В общем случае класс имеет следующий формат:

```
class имя_класса {
      спецификация_доступа:                          //public, protected, private
            тип переменная_1;                  //данные-элементы класса
            тип переменная_2;
             …
            тип имя_функции (параметры);       //функции-элементы
             …
      имя_класса(параметры);                    //конструктор
      ~имя_класса();                           //деструктор
};
```

Обычно описание класса производится в заголовочном файле (C++ Header file), имеющим расширение *.h.

2.6. Доступ к данным. Инкапсуляция

Рассмотрим реализацию концепции даты с использованием структуры Date. Эта структура представляет собой дату и содержит набор функций, осуществляющих манипуляции с переменными-датами:

```
struct Date {
   int d,m,y;
};
void init_date(Date &d, int, int, int);     //инициализация даты d
void add_year(Date &d, int n);             //прибавить n лет к d
void add_month(Date &d, int n);             //прибавить n месяцев к d
void add_day(Date &d, int n);              //прибавить n дней к d
```

Здесь нет явной связи между типом данных и функциями. Такую связь можно установить, объявив функции в качестве членов структуры:

```
struct Date {
   int d,m,y;
   void init_date(int dd, int mm, int yy);  //инициализация даты
   void add_year(int n);                    //прибавить n лет
   void add_month(int n);                           //прибавить n месяцев
   void add_day(int n);                     //прибавить n дней
};
```

Функции, объявленные внутри определения структуры называются функциями-элементами, и их можно вызвать только для переменной соответствующего типа, используя стандартный синтаксис доступа к членам структуры.

Пример 4

```
Date my_birthady;
Date today;
togay.init(12,11,2012);
Date tomorrow = today;
tomorrow.add_day(1);
```

Такое объявление `Date` предоставляет набор функций для работы с `Date`. Однако оно не указывает, что только эти функции непосредственно зависят от представления `Date`, и только они могут непосредственно осуществлять доступ к объектам класса `Date`. Эти ограничения можно отразить, воспользовавшись ключевым словом `class` и спецификациями доступа:

```
class Date {
   int d,m,y;
  public:
   void init_date(int dd, int mm, int yy);   //инициализация даты
   void add_year(int n);                    //прибавить n лет
   void add_month(int n);                           //прибавить n месяцев
   void add_day(int n);                     //прибавить n дней
```


```
};
```

Метка public разделяет тело класса на две части. Имена в первой могут использоваться только функциями-элементами. Вторая же образует открытый интерфейс объектов класса. Ограничение доступа имеет несколько преимуществ: например, ошибка, в результате которой Date приняла неверное значение, может быть вызвана только кодом соответствующей функции-элемента. Из этого следует, что локализация ошибки может быть завершена ещё до запуска программы. Защита закрытых данных базируется на ограничении использования имён членов класса, для чего функции элементы и структуры данных, определяющие некоторые свойства данного класса, рассматриваются в качестве единого целого. Такой подход называется инкапсуляцией.

В общем случае, в разных языках программирования термин «**инкапсуляция**» относится к одной из или обеим одновременно следующим нотациям:

– языковая конструкция, позволяющая связать данные с методами, предназначенными для обработки этих данных;

– механизм языка, позволяющий ограничить доступ одних компонентов программы к другим.

Инкапсуляция подразумевает "защиту" данных в пределах класса таким образом, что только элементы класса получают к ним доступ. То есть, для того чтобы получить значение одного из элементов данных класса, нужно вызвать функцию элемент этого класса, который возвращает необходимое значение. Для присвоения элементу значения, вызывается соответствующая функция элемент данного класса. В объектном программировании считается хорошим тоном закрывать все данные и функции элементы описываемого класса для доступа "извне".

C++ предоставляет программистам три уровня доступа к элементам объектов:

– public (общий/открытый),
– private (приватный/закрытый),
– protected (защищенный).

Элементы, объявленные общими, будут доступны любому внешнему элементу класса, любой функции элементу или выражению в программе, когда объект является видимым.

Приватные элементы доступны только другим элементам своего же класса. Они не доступны извне, за исключением специальных функций, называемых "дружественными" (*будет рассмотрено в ЛР 4*).

К защищенным элементам имеют доступ лишь некоторые из объектов. Они доступны только элементам своего класса и любым его потомкам (*будет рассмотрено в ЛР 2*). Поэтому защищенные элементы занимают промежуточное положение между общими и приватными.

Примечание: приватные элементы недоступны потомкам своего класса. Поэтому и понадобились защищенные элементы.

Умелое использование уровней доступа повышает надежность программ и их способность к изменениям, ослабляя взаимозависимость между объектами. Правильно описанными функциями элементами типа `public` можно изменять приватные элементы, не затрагивая программный код других объектов. По умолчанию в классах предполагается спецификация `private`.

2.7. Элементы-данные (атрибуты) и статические элементы-данные

Элементы-данные могут быть любого типа или структурой данных любого типа, в том числе и классового; не могут быть представителями определяемого класса (но могут быть указателями или ссылками на представителя определяемого класса).

Элементы-данные по аналогии с переменными могут быть константными.

Если элемент данных объявлен как private, то доступ к нему осуществляется только через функции-элементы этого же класса. Если элемент

данных объявлен как public, то доступ может осуществляться через объект, или указатель на объект.

Пример 5

```
class point{
  int x=0,y=0;    //private по умолчанию
};


class point{
  public:         //явно указано public
    int x,y;
};
```

Данные всегда целесообразно объявлять в разделе `private`, в редких случаях – в раздел `protected`, чтобы потомки данного класса имели к ним доступ.

Обычно каждый объект класса имеет свою собственную копию всех данных-элементов класса. Но в определённых случаях во всех объектах класса должна фигурировать только одна копия некоторых данных:

– счётчик числа созданных объектов класса,

– если в классе имеются некоторые константы, одинаковые для всех объектов класса, то нерационально хранить в каждом объекте собственные копии этих констант.

Для введения в класс подобных данных используются статические данные, содержащие информацию «*для всего класса*». Таким образом, статическая переменная является частью класса, но не является частью объектов. Формат объявления статического элемента:

```
static тип_данных имя_переменной;
```

Использование статических элементов сокращает затраты памяти и гарантирует единство данных во всех объектах. Также как и обычные элементы, могут быть открытыми, закрытыми или защищёнными. Доступ к открытым статическим переменным класса возможен посредством любого объекта класса или посредством имени класса с помощью бинарной операции

разрешения области действия. Закрытые и защищённые статические элементы класса должны быть доступны открытым функциям-элементам этого класса и друзьям класса.

Статические элементы класса существуют даже тогда, когда не существует никаких объектов этого класса. Начальные значения статических элементов (открытых и закрытых) должны задаваться вне объявления класса. Задать начальное значение статического элемента можно только один раз в файле.

```
int Myclass::D=10;
```

2.8. Функции-элементы (методы)

Функции-элементы класса – это функции, которые имеют доступ к любым другим функциям-элементам и к любым элементам-данным. Наиболее часто используются функции, реализующие арифметические методы, методы отображения и сравнения данных. Особым видом функций-элементов являются конструкторы и деструкторы.

Разделяют встроенные (определённые внутри класса) и определённые вне класса функции-элементы.

Формат определения функции внутри класса:

```
тип_возвращаемого_значения имя_функции(параметры) {тело}
```

Формат определения функции вне класса (внутри класса достаточно определить прототип):

```
тип_возвращаемого_значения имя_класса::имя_функции(параметры) {тело}
```

Поскольку несколько классов могут иметь функции с одинаковым именем, при описании функции вне класса нужно явно указать имя класса которому принадлежит функция, и только потом через оператор расширения видимости, имя функции.

Пример 6

```
//определение функции внутри класса
class point{
  int x=0,y=0;
  public:
    void init(int Val_x, int Val_y){ x = Val_x; y = Val_y; }
```


```
};
//определение функции вне класса
class point{
  int x=0,y=0;
  public:
    void init(int Val_x, int Val_y);
};
void point::init(int Val_x, int Val_y){
  x = Val_x;
  y = Val_y;
}
```

В теле функции имена элементов-данных можно использовать без явного указания объекта, т.к. в теле класса он создан по умолчанию (*детальнее рассмотрено в подразделе 2.12*).

Простые функции чаще определяют внутри класса; фактически они являются встроенными функциями `inline`. Рекомендуется реализацию функций-элементов размещать в отдельном файле реализации, в объявлении класса должны содержаться только прототипы функций. Такая организация программы обеспечивает независимость всех модулей, использующих заголовочный файл с объявлением класса, от каких-то изменений в реализации функций-элементов класса.

Исходя из принципа инкапсуляции, элементы-данные всегда должны быть защищены от несанкционированного доступа. Как правило, доступ к ним осуществляется только через функции, включающие методы чтения и записи полей (**т.н. getter и setter**). В этих функциях записи должна осуществляться проверка данных, чтобы не записать случайно в элементы неверные данные и чтобы не допустить их неверной трактовки. Функции чтения позволяют не переписывать всю программу, если происходит изменение в типе, способе хранения и размещения элементов данных в классе.

Пример 7

```
class MyClass{
  public:
   void SetA(int); //функция записи
   int GetA();   //функция чтения
  private:
      int A;
      double B,C;};
void MyClass::SetA(int Value){
```


```
    if(…)      //проверка корректности
      A=Value;
}
int MyClass::GetA() {return A;}
```

В данном примере функция чтения просто возвращает значение поля, однако в более сложных классах может потребоваться какая-то предварительная обработка данных.

Если нужно запретить функции модифицировать элементы-данные, ее можно объявить как константную. Для того, чтобы объявить функцию класса константной, необходимо указать ключевое слово `const` после прототипа функции, но до начала ее тела.

Пример 8

```
//определение константной функции внутри класса
class point{
  int x = 0, y = 0;
public:
  int get_X() const { return x; }
  void init(int Val_x, int Val_y) const { x = Val_x; y = Val_y; }
//ошибка, попытка модифицировать значения в константной функции
};

//определение константной функции вне класса
class point{
  int x = 0, y = 0;
public:
  int get_X() const;
  void init(int Val_x, int Val_y) const;
};
int point::get_X() const{
  return x;
}
void point::init(int Val_x, int Val_y) const{ //ошибка, попытка
модифицировать значения в константной функции
  x = Val_x;
  y = Val_y;
}
```

Стоит отметить, что при описании функции вне класса перед ее телом необходимо указать ключевое слово `const` повторно.

Константная функция-элемент может вызывать только другие константные функции-элементы.

При необходимости все же можно изменить значение атрибута класса константным методом этого же класса. Для этого соответствующий атрибут следует специфицировать ключевым словом `mutable`.

Пример 9

```
//определение константной функции вне класса
class point{
  mutable int x = 0, y = 0;
public:
  int get_X() const;
  void init(int Val_x, int Val_y) const;
};
int point::get_X() const{
  return x;
}
void point::init(int Val_x, int Val_y) const{ //ошибки не будет
  x = Val_x;
  y = Val_y;
}
```

Функции-элементы, по аналогии с элементами данными, могут быть описаны как `static`. В этом случае они могут быть вызваны без создания объекта класса, что очень удобно, например, при создании классов-преобразователей.

2.9. Конструкторы

Использование функций типа «init()» для инициализации объектов класса неэлегантно и подвержено ошибкам. Так как нигде не сказано, что объект должен быть проинициализирован, программист может сделать это дважды или забыть об этом. Наилучшим подходом будет предоставление программисту возможности объявить функцию, имеющую явное назначение – инициализация объекта. Такая функция называется конструктором и распознаётся по имени, которое совпадает с именем самого класса.

Таким образом, конструктором класса называется открытая функция-элемент, которая вызывается в момент создания класса и должна инициализировать данные указанными в вызове значениями или значениями по умолчанию.

Формат определения конструктора:

– внутри класса `имя_класса(параметры) {//тело конструктора};`
– за телом класса `имя_класса::имя_класса(параметры) {//тело конструктора}.`
Например:

```
class MyClass{
    int A;
  public:
    MyClass(void);
};
MyClass::MyClass(void) {A=0;}
```

Простое задание в конструкторе значений данных в общем случае не гарантирует их целостность, обычно нужна ещё и проверка допустимости данных. Лучше для инициализации данных использовать соответствующие функции записи.

Создание объекта класса в программе может осуществляться объявлением соответствующей переменной или динамическим размещением переменной в памяти (более детально основные способы и их вариации будут рассмотрены в подразделе 2.11):

```
MyClass MC;
MyClass *PMC = new MyClass;
```

В момент выполнения каждого из этих операторов неявным образом выполняется вызов конструктора, устанавливающего начальные значения данных. Недостатком таких конструкторов является то, что все начальные значения данных задаются конструктором. Вызывающая функция никак не может задать другое значение.

Для устранения этого недостатка используются конструкторы, в которых все начальные значения задаются как параметры. В этом случае прототип конструктора имеет вид:

```
MyClass(int);
MyClass:: MyClass(int a){A=a;}
```

В этом случае создание объекта выполняется операторами:

```
MyClass MC(1);
MyClass *PMC = new MyClass(1);
```

Для создания объекта, в классе обязательно должен присутствовать конструктор, поэтому компилятор для каждого класса неявно создает два конструктора: по умолчанию и копирования по умолчанию (см. ниже).

Конструктор по умолчанию – конструктор, который может быть вызван без передачи аргументов (включая конструктор с параметрами, имеющими значение по умолчанию).

Если в классе не определён конструктор по умолчанию, компилятор неявно создаст его. Он будет аналогичен явно объявленному конструктору с пустым телом.

Если у конструктора есть один или несколько параметров по умолчанию – это по прежнему конструктор по умолчанию. Каждый класс может иметь не более одного конструктора по умолчанию: либо без параметров, либо с параметрами, имеющими значения по умолчанию.

В C++ конструкторы по умолчанию имеют существенное значение, поскольку они автоматически вызываются при определенных обстоятельствах, и, следовательно, при определённых условиях класс обязан иметь конструктор по умолчанию, а его отсутствие приведет к ошибке.

Случаи, когда необходим конструктор по умолчанию:

– когда объект объявляется без аргументов (например, `MyClass MC;`) или создаётся новый экземпляр в памяти (например, `new MyClass;` или `new MyClass();`);

– когда объявлен массив объектов, например, `MyClass x[10];` или объявлен динамически, например `new MyClass [10]`. Конструктор по умолчанию инициализирует все элементы;

– когда в классе потомке не указан явно конструктор класса родителя в списке инициализации;

– когда конструктор класса не вызывает явно конструктор хотя бы одного из своих полей-объектов в списке инициализации;

– в стандартной библиотеке (STL) определённые контейнеры заполняют свои значения, используя конструкторы по умолчанию, если значение не указано явно.

Важно! Если определены конструкторы для класса, но среди них нет конструктора по умолчанию, компилятор не создаст неявно таковой что приведет к ошибкам в выше указанных случаях.

Начиная с C++ версии 11 (далее C++ 11) конструктор по умолчанию может быть явно указан и явно удален.

Пример 10

```
class MyClass
{
public:
    MyClass () = default; // явно указан конструктор по умолчанию
};
class MyClass
{
public:
    MyClass () = delete; //удаление конструктора по умолчанию
(предотвращение создания такового)
};
```

При написании конструкторов удобно использовать список инициализации, его описание выглядит следующим образом:

```
имя_класса(параметры):имя_элемента1(значение_элемента1),…{}.
```

Для конструктора

```
MyClass(int);
MyClass:: MyClass(int a){A=a;}
```

список инициализации будет иметь следующий вид:

```
MyClass(int a):A(a);
```

Список инициализации обязателен, если элементами-данными являются константы или ссылки, либо объект, для которого определён конструктор с параметрами. Используя список инициализации можно изменять значения констант в момент создания объекта.

Пример 11

```
class MyClass{
    int A;
    const int MaxA;
    const int MinA;
  public:
    MyClass(int=0);
    void SetA(int);    //функция чтения
};
MyClass::MyClass(int a):MaxA(10),MinA(1){SetA(a);}
```


Использование конструктора, в котором программно задаются значения всех элементов класса может быть довольно громоздким. Поэтому реально используются конструкторы с параметрами по умолчанию, в этом случае объявление и реализация конструктора могут иметь такой вид:

```
MyClass(int =0);
MyClass::MyClass(int a) {SetA(a);}
```

Объект такого класса можно задавать любым из рассмотренных ранее операторов создания объекта. Если при создании указывается аргумент, то его значение присваивается полю. Если аргумент не указывается, то присваивается значение по умолчанию.

При помощи конструктора по умолчанию можно предоставить пользователю возможность изменять значения констант в момент создания объекта. В этом случае в конструкторе по умолчанию нужно предусмотреть для констант соответствующие значения по умолчанию:

```
MyClass(int A=0, int MaxA=10, int MinA=1);
MyClass::MyClass(int a, int j, int i):MaxA(i),MinA(j){SetA(a)};
```

Также используют метод объявления нескольких конструкторов с разными наборами параметров (по аналогии с перегрузкой функций):

```
class Date{
    int d,m,y;
    public:
       Date(int, int, int);        //день, месяц, год
       Date(int, int);           //день, месяц, текущий год
       Date(int);                        //день, текущий месяц и год
       Date();                   //текущая дата
       Date(const char*);                //дата в строковом представлении
};
```

Правила использования конструктора:

– конструктор имеет тоже имя, что и его класс;

– конструктор не содержит оператора return и не возвращает никакого значения (даже void);

– конструктор может определяться явно пользователем или неявно компилятором. Неявно происходит объявление конструктора в таких случаях: определение нового объекта класса, при копировании объекта, при динамическом создании объекта с помощью оператора new;

– определение конструктора может производиться внутри и вне класса;

– конструктор может иметь, а может не иметь параметры. Конструктор с параметрами инициализирует объект при объявлении. Конструктор без параметров инициализирует пустой объект;

– конструкторы не могут быть вложенными;

– конструктор не может быть константным, статическим или виртуальным (`const, static или virtual`);

– ошибки, генерируемые конструктором, обрабатываются только механизмом обработки исключительных ситуаций;

– конструктор не наследуется.

Различают конструкторы инициализации, копирования, перемещения и преобразования.

1. **Конструктор инициализации** – в теле конструктора элементам данных присваиваются значения, инициализация возможна через ввод, присваивание, список инициализации.

2. **Конструктор копирования** – создаёт новый объект на основе существующего (фактически копию). В качестве параметра такой конструктор использует константную ссылку на существующий объект. Формат объявления:

```
имя_класса(имя_класса const & имя_объекта){}
```

Пример 12

```
#include "stdafx.h"
#include <cstdlib>
#include <iostream>

using namespace std;

class point{
  int x, y;
public:
  point();
  point(const point &);
  void setx(int px){ x = px; }
  void sety(int py){ y = py; }
  int getx() { return x; }
  int gety() { return y; }
};
```


```
point::point(){  //конструктор инициализации
  cout << "Input ";
  cin >> x >> y;
}
point::point(const point &k) :x(k.x), y(k.y){} //конструктор копирования

void main(){
  const point a;  // создаем объект a
  point p(a);     // в объект p копируем a
  cout << endl << p.getx() << "  " << p.gety() << endl;
  p.setx(25);     // изменяем значение x
  point c(p);     // в объект c копируем p с измененным x
  cout << endl << c.getx() << "  " << c.gety()<<endl;
  system("pause");
}
```

Если в классе нет явно опасного конструктора копирования, то компилятор генерирует его автоматически. Такой конструктор выполняет поверхностное копирование, то есть поэлементное копирование атрибутов класса.

Автоматически генерируемый конструктор копирования не учитывает особенностей объекта, а потому не всегда может применяться. Известны два случая, когда конструктор копирования необходимо описывать в классе явно:

– если при копировании атрибутов необходимо менять содержание хотя бы некоторых из них;

– если класс содержит хотя бы одно поле с динамически выделяемой памятью.

Во втором случае из-за поверхностного копирования при удалении копии или оригинала объекта, для оставшегося объекта поле с динамически выделяемой памятью будет утеряно, поскольку и них будет один и тот же адрес, в этом случае необходимо глубокое копирование.

3. **Конструктор перемещения** – позволяет создавать объекты на основе временных объектов (выражений), заменив полное копирование объекта перемещением указателя.

Копирование объекта может быть довольно затратным с точки зрения ресурсов. К примеру, для копирования объекта который содержит

динамический массив – это вызов функции, выделение памяти и цикл. Это, конечно, приемлемо, когда нам действительно нужны две копии массива, но во многих случаях это не так: мы часто копируем массив из одного места в другое, а потом удаляем старую копию. Чаще всего так происходит, когда копируемый объект является временным.

Пример 13

```
#include "stdafx.h"
#include <cstdlib>
#include <iostream>

using namespace std;

class Array //класс, который содержит динамический массив
{
private:
  int length; // размер массива
  int* data;  // динамический массив
public:
  Array(int length_p); // конструкторы инициализации
  Array(Array&& k);    // конструктор перемещения
  void Array_out();    // функция вывода массива
  ~Array(){…};         // деструктор, описан в следующем подразделе
};
Array::Array(int length_p) : length(length_p), data(new int[length_p])
{
  cout << "Init Constructor" << endl;
  for (int i = 0; i < length; i++) data[i] = i;
}

Array::Array(Array&& k) : length(k.length), data(k.data) {
  cout << "Move Constructor" << endl;
  k.data = nullptr;   // обнуляем, что бы не вызвать деструктор для временного объекта
  k.length = 0;
}

void Array::Array_out(){ // вывод массива
  for (int i = 0; i < length; i++) cout << data[i] << "  ";
  cout << endl;
}

Array CreateArray(int l) //функция, которая создает объект и возвращает его как временный
{
  Array tmp(l);
  return tmp;
}
int main()
{
  Array a(CreateArray(20));    // вызываем конструктор инициализации и перемещения
  a.Array_out();
  system("pause");
```


```
  return 0;
}
```

4. **Конструктор преобразования** – любой конструктор с параметрами является преобразователем из одного типа в другой. Конструктор, принимающий скалярные параметры, выполняет преобразования к классовому типу; принимающий параметры классового типа – выполняет преобразования из одного класса в другой.

2.10. Деструкторы

Деструктор – это специальная функция-элемент, срабатывающая при уничтожении объекта класса и освобождающая занимаемую им память. Имя деструктора совпадает с именем класса, но перед ним записывают символ ~:

```
~имя_класса(){}.
```

Деструкторы необходимы, если конструктор или какие-то функции-элементы класса динамически распределяют память, создавая в ней какие-то объекты. В остальных случаях можно обойтись без деструктора; если деструктор явным образом в классе не объявлен, то компилятор сам генерирует необходимые коды освобождения памяти.

Деструктор вызывается автоматически, когда объект выходит из зоны видимости программы, но его можно вызвать явно в отличии от конструктора.

Дополним пример из предыдущего раздела деструктором.

Пример 14

```
#include "stdafx.h"
#include <cstdlib>
#include <iostream>

using namespace std;

class Array //класс, который содержит динамический массив
{
private:
  int length; // размер массива
  int* data;  // динамический массив
public:
  Array(int length_p); // конструкторы инициализации
  Array(Array&& k);    // конструктор перемещения
  void Array_out();    // функция вывода массива
  ~Array();            // деструктор
};
Array::Array(int length_p) : length(length_p), data(new int[length_p])
```


```
{
  cout << "Init Constructor" << endl;
  for (int i = 0; i < length; i++) data[i] = i;
}

Array::Array(Array&& k) : length(k.length), data(k.data) {
  cout << "Move Constructor" << endl;
  k.data = nullptr;   // обнуляем, что бы не вызвать деструктор для временного объекта
  k.length = 0;
}

void Array::Array_out(){ // вывод массива
  for (int i = 0; i < length; i++) cout << data[i] << "  ";
  cout << endl;
}

Array CreateArray(int l) //функция, которая создает объект и возвращает его как временный
{
  Array tmp(l);
  return tmp;
}
Array::~Array()
{
  if (data)
  {
          cout << " Deleting!" << endl;
          delete[] data; // удаляем массив
  }
}
int main()
{
  Array a(CreateArray(20));    // вызываем конструктор инициализации и перемещения
  a.Array_out();
  system("pause");
  return 0;
}
```

Правила использования деструктора:

– деструктор имеет тоже имя, что и его класс с символом «~» в начале;

– деструктор не имеет параметров;

– не имеет типа данных;

– не содержит return;

– не наследуется;

– не может быть объявлен как `const, static, void;`

– не может быть перегружен.

2.11. Создание и инициализация объектов и массивов объектов

Как уже было сказано, создание объекта класса в программе может осуществляться объявлением соответствующей переменной или динамическим размещением переменной в памяти, однако эти два основных подхода включают в себя множество особых случаев создания объекта, которые стоит понимать.

Определим класс TPoint, включающий два закрытых атрибута, конструктор по умолчанию и конструктор инициализации, функции инициализации и вывода закрытых атрибутов на экран:

```
class Point
{
private:
  int x, y;
public:
  Point(){}
  Point(int px, int py) : x(px), y(py){ }
  void SetPoint(int px, int py){ x = px; y = py; }
  void Print(){ std::cout << x << " " << y << endl; }
};
```

На примере этого класса рассмотрим основные способы создания объектов.

1) Неинициализированный статический объект (рисунок 2.1).

```
Point a;          // объявление объекта
a.SetPoint(5, 10);       // инициализация атрибутов
a.Print();                // вывод значений атрибутов
```

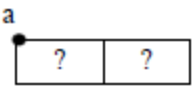

Рисунок 2.1

Поскольку объект «a» – статический, то память под него специально выделять и освобождать не нужно, эта операция будет выполнена автоматически.

2) Инициализированный статический объект (рисунок 2.2).

```
Point b(5,10);           // объявление и инициализация объекта
b.Print();                // вывод значений атрибутов
```

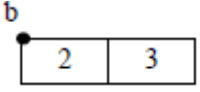

Рисунок 2.2

Объект «b» сразу создается, инициализированным, проблем с памятью тоже нет.

3) Неинициализированный динамический объект (рисунок 2.3).

```
Point *c;          // объявление указателя на объект
c = new Point(3, 4);      // выделение памяти и инициализация
c->Print();               // вывод значений атрибутов
delete c;                 // освобождение памяти
```

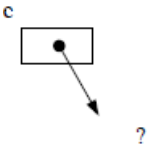

Рисунок 2.3

Под объект «c» необходимо отдельно выделить память, соответственно ее нужно и освободить.

4) Инициализированный динамический объект (рисунок 2.4).

```
Point *d = new Point(3, 4); //объявление указателя на объект, выделение памяти и инициализация объекта
d->Print();                                   // вывод значений атрибутов
delete d;                                        // освобождение памяти
```

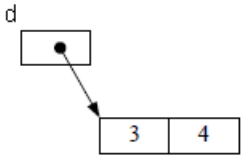

Рисунок 2.4

Под объект «d» необходимо отдельно выделить память, соответственно ее нужно и освободить.

5) Неинициализированный статический массив объектов (рисунок 2.5).

```
Point e[4];                        // объявление массива объектов
for (int i = 0; i<4; i++)
{
  e[i].SetPoint(i*i, i - 5);       // инициализация атрибутов
  e[i].Print();                    // вывод значений атрибутов
}
```

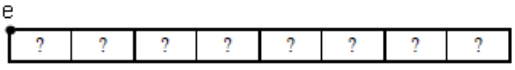

Рисунок 2.5

Поскольку массив объявляется неинициализированным, его объекты приходится инициализировать, после чего значения атрибутов можно выводить. Проблем с памятью нет.

6) Инициализированный статический массив объектов (рисунок 2.6).

```
Point f[2] = { Point(2, 4), Point(4, 5) };      /*     создание     массива
объектов и инициализация их атрибутов */
for (int i = 0; i<2; i++) f[i].Print(); // вывод значений атрибутов
```

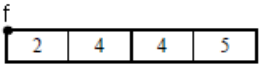

Рисунок 2.6

Массив создается инициализированным, значения атрибутов можно сразу выводить. Проблем с памятью нет.

7) Неинициализированный динамический массив объектов (рисунок 2.7).

```
Point *g = new Point[3];     //создание динамического массива объектов
for (int i = 0; i<3; i++)
{
  g[i].SetPoint(i, i + 1);   // инициализация атрибутов
  g[i].Print();              // вывод значений атрибутов
}
delete[] g;
```

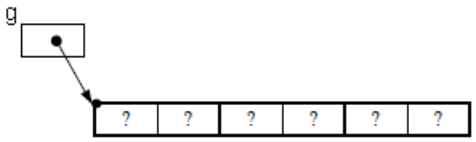

Рисунок 2.7

Память под динамический массив выделяется одним куском, поэтому освобождать ее надо также одним куском.

8) Неинициализированный статический массив указателей на динамические объекты (рисунок 2.8).

```
      Point *h[3];                        //создание статического массива указателей
на объекты
      for (int i = 0; i<3; i++)
      {
        h[i] = new Point(i, i + 1); // выделение памяти и инициализация
атрибутов
        h[i]->Print();                 // вывод значений атрибутов
      }
      for (int i = 0; i<3; i++) delete h[i];
```

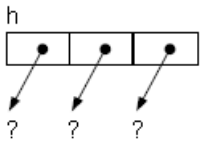

Рисунок 2.8

В данном случае сам массив указателей – статический, то есть память (динамическую) под него выделять не нужно. Память (динамическую) следует выделить под каждый из объектов, а по завершении работы освободить ее.

9) Инициализированный статический массив указателей на инициализированные динамические объекты (рисунок 2.9).

```
      Point *k[] = { new Point(2, 7), new Point(1, 5), new Point(4, 2) };
//создание статического массива указателей на объекты и инициализация
      for (int i = 0; i<3; i++)
      {
        k[i]->Print(); // вывод значений атрибутов
      }
      for (int i = 0; i<3; i++) delete k[i];
```

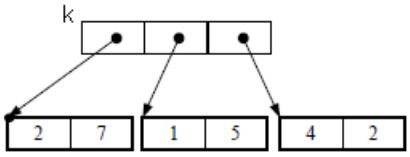

Рисунок 2.9

В этом случае память под объекты также выделяется динамически, а их адреса заносятся в статический массив указателей. Освобождение памяти выполняется в цикле отдельными кусками так, как она и выделялась.

10) Неинициализированный динамический массив указателей на неинициализированные динамические массивы неинициализированных объектов (рисунок 2.10).

```
Point **m;              // указатель 2-го порядка на Point
m = new Point *[3];     // динамический массив указателей
for (int i = 0; i < 3; i++) m[i] = new Point[3];//динамические массивы объектов

for (int i = 0; i < 3; i++)
for (int j = 0; j < 3; j++){
  m[i][j].SetPoint(i, j);  //инициализация атрибутов
  m[i][j].Print();         // вывод значений атрибутов
}

for (int i = 0; i < 3; i++) delete m[i];  //удаление
delete []m;
```

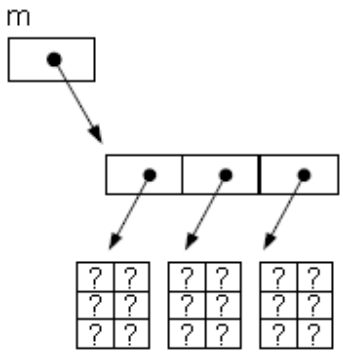

Рисунок 2.10

В этом случае память под массив указателей и массивы объектов выделяется динамически. Освобождение памяти сначала выполняется в цикле для динамических массивов объектов, а потом отдельно для массива указателей.

Были рассмотрены основные способы создания объектов, но на самом деле их может быть несчетное количество.

2.12. Неявный параметр this

Когда метод, принадлежащий классу, вызывается для обработки данных конкретного объекта, ему, кроме явно объявленных параметров, автоматически и неявно передается еще один скрытый параметр: указатель на тот объект, для

которого вызывается этот метод. В С ++ этот указатель имеет специальное имя `this` и неявно определен в каждом методе класса следующим образом:

```
имя класса *const this
```

Явно описать или определить указатель `this` нельзя. Исходя из неявного синтаксиса определения, `this` является константным указателем, то есть изменять его недопустимо. Однако в каждом методе, принадлежащему классу, он указывает именно на тот объект, для которого вызывают этот метод. Говорят, что указатель `this` является дополнительным (скрытым) параметром каждого нестатического метода. Объект, который адресуется указателем `this`, становится доступным внутри принадлежащего класса метода именно с помощью указателя `this`. При работе с членами класса внутри принадлежащего классу метода можно везде использовать этот указатель.

Пусть определен следующий класс:

```
class myClass
{   int c;
void Fun() {...};
// ...
};
```

В теле функции `Fun()` можно использовать такой оператор для присвоения члену «с» значение 5:

```
c = 5;
```

На самом деле предыдущий оператор является сокращенной формой такого оператора:

```
this->c = 5;
```

Соответственно при объявлении некоторого объекта `A` выполняется операция `this = &A`, а при выделении памяти для размещения динамического объекта адресуется указателем `b` – операция `this = b`.

При работе с членами класса этот указатель можно задавать явным образом. Но следует отметить, что в этом случае использование `this` обычно не оказывает никакого преимущества, так как данные большинства конкретных объектов уже доступны функциям, принадлежащим классу, по именам.

Однако иногда явное применение указателя `this` полезно и даже необходимо. Например, без него нельзя обойтись, если в теле принадлежащей классу функции нужно явно задать адрес объекта, для которого она была вызвана.

Пример 15

```
#include "stdafx.h"
#include <cstdlib>
#include <iostream>

using namespace std;

class A
{
  int x, y;
public:
  void print(void)
  {
          cout << x <<" "<< y << endl;
  }
  A *fun1()              // возвращает указатель на объект, для которого вызывается
  {
          x = y = 100;
          return this;
  }
  A fun2()               // возвращает объект, для которого вызывается
  {
          x = y = 200;
          return *this;
  }
};
int main()
{
  A a;
  a.fun1()->print();     // выводит: 100 100
  a.fun2().print();      // выводит: 200 200
  system("pause");
  return 0;
}
```

Кроме того, указатель `this` явно используют для формирования результата при переопределении операций, так как операция переопределяется для конкретного объекта, вызывающего ее (*детальнее будет рассмотрено в ЛР4*).

2.13. Присвоение объектов

Объекты одного и того же класса можно присвоить друг другу при помощи «=».

Пример 16

```
myClass ObjA, ObjB;
ObjB = ObjA;
```

При выполнении операции присваивания по умолчанию осуществляется поверхностное копирование, то есть данные первого объекта поразрядно копируются во второй.

Следует отметить, что присвоение одного объекта другому просто делает значение их атрибутов одинаковыми, но эти два объекта остаются абсолютно независимыми. Следовательно, дальнейшее модифицирование данных одного объекта не влияет на данные другого.

Важно помнить что, из-за поверхностного копирования при удалении копии или оригинала объекта, для оставшегося объекта поле с динамически выделяемой памятью будет утеряно, поскольку и них будет один и тот же адрес, в этом случае необходимо глубокое копирование (пользовательский конструктор копирования).

2.14. Передача объектов функциям

Объекты можно передавать в функции так же, как и переменные любых других типов данных. По умолчанию объекты класса передаются функциям по значению. То есть, в функцию передается не сам объект, а его копия, которая становится параметром функции. Создание копии означает появление нового временного объекта. Все изменения, внесенные в объект-копию в процессе выполнения функции, не влияют на входной объект, который используется как аргумент для функции.

2.15. Возвращение объектов функциями

Если объекты можно передавать в функции, то с таким же успехом функции могут возвращать объекты, как результат. По умолчанию объект класса возвращается из функции по значению.

Относительно механизма возвращения объектов функциями, возникает ситуация, аналогичная ситуации с передачей объектов в функции. Если тип значения, возвращаемого функцией, является объектом класса, то при вызове функции компилятор автоматически генерирует временный объект этого класса и использует значение, которое определено в операторе `return`, для инициализации этого объекта. По возвращении значения из функции немедленно вызывается деструктор временного объекта и этот объект разрушается.

Разрушение временного объекта в некоторых ситуациях может вызвать непредсказуемые побочные эффекты. Так, если возвращенный функцией объект имеет деструктор, который освобождает динамически выделенную область памяти, то эта память будет освобождена даже в том случае, если объект, который получает возвращенное функцией значение, все еще использует эту память.

Хорошим способом обойти эту проблему является создание пользовательского конструктора копирования.

3 ЗАДАНИЯ

Разработать пошаговый алгоритм решения расчётной части задачи и подзадач (при необходимости). Разработать UML диаграмму классов. Выполнить программную реализацию задания согласно варианту. Прототипы классов должны содержаться в отдельном *.h-файле. В программе обязательно предусмотреть вывод информации об исполнителе работы (ФИО), номере варианта, выбранном уровне сложности и задании. Предусмотреть возможность повторного запуска программы (запрос о желании выйти из программы, или продолжить работу). Ключевые моменты программы обязательно должны содержать комментарии.

Уровень А (1 балл)

1. Определить класс «Точка» с закрытыми элементами X и Y (координаты точки). Определить методы класса, обеспечивающие доступ к этим переменным. Задать для класса три конструктора:

– конструктор инициализации с инициализацией через ввод данных с клавиатуры;

– конструктор преобразования;

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Точка», и создающий зеркальную точку относительно оси абсцисс.

Создать неинициализированный статический массив точек, задать значения координат точек и вывести их координаты на экран.

2. Определить класс «Точка» с закрытыми элементами X и Y (координаты точки). Определить методы класса, обеспечивающие доступ к этим переменным. Задать для класса такие конструкторы:

– конструктор инициализации с инициализацией данных при помощи передачи параметров (по умолчанию точку создавать в начале координат);

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Точка», и создающий зеркальную точку относительно оси ординат.

Создать инициализированный статический массив точек и вывести их координаты на экран.

3. Определить класс «Точка» с закрытыми элементами X и Y (координаты точки). Определить методы класса, обеспечивающие доступ к этим переменным. Задать для класса такие конструкторы:

– конструктор инициализации с инициализацией данных при помощи передачи параметров (по умолчанию точку создавать в координатах (10,10));

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Точка», и создающий зеркальную точку относительно оси абсцисс и ординат.

Создать неинициализированный динамический массив точек, задать значения координат точек и вывести их координаты на экран.

4. Определить класс «Точки» с закрытыми элементами X1, Y1 и X2, Y2 (координаты точек). Определить методы класса, обеспечивающие доступ к этим переменным. Определить метод находящий длину отрезка, который образовывают эти точки. Задать для класса такие конструкторы:

– конструктор по умолчанию, создающий точки с координатами 0,0 и 5,5;

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Точки», и создающий отрезок, удаленный от исходного (X1, Y1 и X2, Y2) на 3.

Создать неинициализированный статический массив указателей на точки, задать значения координат точек и вывести их координаты на экран.

5. Определить класс «Точка» с закрытыми элементами X и Y (координаты точки). Определить методы класса, обеспечивающие доступ к этим переменным. Задать для класса такие конструкторы:

– конструктор по умолчанию, создающий точку с координатами 5,5;

– конструктор перемещения, создающий объект на основе временного;

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Точка», и создающий точку, сдвинутую относительно исходной на 5 и 8 точек.

Создать инициализированный статический массив указателей на точки и вывести их координаты на экран.

6. Определить класс «Круг» с закрытыми элементами X, Y (координаты центра) и R (радиус круга). Определить методы класса, обеспечивающие доступ к этим переменным. Создать метод, определяющий площадь круга. Задать для класса конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Круг», и создающий круг с центром в той же точке, но имеющим радиус в два раза больше. По умолчанию создавать круг с единичным радиусом в начале координат.

Создать неинициализированный статический массив кругов, задать значения координат центра и радиус, вывести на экран площадь каждого круга.

7. Определить класс «Круг» с закрытыми элементами X, Y (координаты центра) и R (радиус круга). Определить методы класса, обеспечивающие доступ к этим переменным. Создать метод, определяющий площадь круга. Задать для класса конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Круг», и создающий круг с центром в точке, зеркальной относительно оси абсцисс, и таким же радиусом. По умолчанию создавать круг с единичным радиусом в точке (10,10).

Создать инициализированный статический массив кругов и вывести на экран площадь каждого круга.

8. Определить класс «Окружность» с закрытыми элементами X, Y (координаты центра) и R (радиус окружности). Определить методы класса, обеспечивающие доступ к этим переменным. Создать метод, определяющий длину окружности. Задать для класса конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Окружность», и создающий окружность с центром в той же точке, но имеющей радиус в два раза меньше. Задать конструктор инициализации с инициализацией данных через ввод.

Создать неинициализированный динамический массив окружностей, задать значения координат центра и радиус, вывести на экран длину каждой окружности.

9. Определить класс «Окружность» с закрытыми элементами X, Y (координаты центра) и R (радиус окружности). Определить методы класса, обеспечивающие доступ к этим переменным. Создать метод, определяющий длину окружности. Задать для класса конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Окружность», и создающий окружность с центром в точке, зеркальной относительно оси ординат, и таким же радиусом. Задать конструктор инициализации с инициализацией данных через ввод.

Создать неинициализированный статический массив указателей на окружности, задать значения координат центра и радиус, вывести на экран длину каждой окружности.

10.Определить класс «Прямоугольник» с закрытым элементами a и b – стороны прямоугольника. Определить методы класса, обеспечивающие доступ к этим переменным. Создать метод, определяющий площадь прямоугольника. Задать такие конструкторы для класса:

– конструктор инициализации с инициализацией данных при помощи передачи параметров (по умолчанию задавать прямоугольник со сторонами 2 и 4);

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Прямоугольник», и создающий прямоугольник со сторонами, равными меньшей стороне копируемого объекта.

Создать инициализированный статический массив указателей на прямоугольники и вывести площадь каждого прямоугольника на экран.

11.Определить класс «Прямоугольник» с закрытым элементами a и b – стороны прямоугольника. Определить методы класса, обеспечивающие доступ к этим переменным. Создать метод, определяющий площадь прямоугольника. Задать такие конструкторы для класса:

– конструктор инициализации с инициализацией данных через ввод;

– конструктор перемещения, создающий объект на основе временного;

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Прямоугольник», и создающий прямоугольник со сторонами, равными большей стороне копируемого объекта.

Создать неинициализированный статический массив прямоугольников, задать значения сторон и вывести на экран площадь каждого прямоугольника.

12.Определить класс «Прямоугольник» с закрытым элементами a и b – стороны прямоугольника. Определить методы класса, обеспечивающие доступ к этим переменным. Создать метод, определяющий диагональ прямоугольника. Задать такие конструкторы для класса:

– конструктор инициализации с инициализацией данных через ввод;

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Прямоугольник», и создающий прямоугольник со сторонами, равными диагонали копируемого объекта.

Создать инициализированный статический массив прямоугольников и вывести на экран площадь каждого прямоугольника.

13.Определить класс «Строка» с закрытыми элементами s (указатель на строку) и len(длина строки). Определить методы класса, обеспечивающие доступ к этим переменным. Создать такие конструкторы для класса:

– конструктор инициализации, принимающий в качестве параметра строку;

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Строка», и создающий перевёрнутую строку.

Определить деструктор класса.

Создать неинициализированный динамический массив строк, задать их, вывести на экран каждую строку.

14.Определить класс «Строки» с закрытыми элементами s и q (указатели на строку) и len1 и len2 (длины строк). Определить методы класса, обеспечивающие доступ к этим переменным. Создать такие конструкторы для класса:

– конструктор инициализации, принимающий в качестве параметров строки;

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Строки», и меняющий местами строки s и q.

Определить деструктор класса.

Создать неинициализированный статический массив указателей на строки, задать их и вывести на экран каждую строку.

15.Определить класс «Строка» с закрытыми элементами s (указатель на строку) и len (длина строки). Определить методы класса, обеспечивающие доступ к этим переменным. Создать такие конструкторы для класса:

– конструктор инициализации с инициализацией данных через ввод;

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Строка», и создающий строку, состоящую только из чётных символов исходной строки.

Определить деструктор класса.

Создать инициализированный статический массив указателей на строки и вывести на экран каждую строку.

16.Определить класс «Строка» с закрытыми элементами s (указатель на строку) и len(длина строки). Определить методы класса, обеспечивающие доступ к этим переменным. Создать такие конструкторы для класса:

– конструктор инициализации с инициализацией данных через ввод;

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Строка», и создающий строку, состоящую только из нечётных символов исходной строки.

Определить деструктор класса.

Создать неинициализированный статический массив строк, задать их и вывести на экран каждую строку.

17.Определить класс «Треугольник» с закрытыми элементами a, b и c (стороны треугольника). Определить методы класса, обеспечивающие доступ к этим переменным. Определить метод, вычисляющий площадь треугольника. Создать такие конструкторы для класса:

– конструктор инициализации с инициализацией данных через ввод;

– конструктор перемещения, создающий объект на основе временного;

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Треугольник», и создающий треугольник, стороны которого в два раза больше сторон исходного треугольника.

Создать инициализированный статический массив треугольников, задать их стороны и вывести на экран площадь каждого треугольника.

18.Определить класс «Треугольник» с закрытыми элементами a, b и c (стороны треугольника). Определить методы класса, обеспечивающие доступ к этим переменным. Определить метод, вычисляющий периметр треугольника. Создать такие конструкторы для класса:

– конструктор инициализации с инициализацией данных при помощи передачи параметров;

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Треугольник», и создающий треугольник, стороны которого в два раза меньше сторон исходного треугольника.

Создать неинициализированный динамический массив треугольников, задать их стороны и вывести на экран периметр каждого треугольника.

19.Определить класс «Треугольник» с закрытыми элементами a, b и c (стороны треугольника). Определить методы класса, обеспечивающие доступ к этим переменным. Определить метод, определяющий, является ли треугольник прямоугольным. Создать такие конструкторы для класса:

– конструктор инициализации (по умолчанию создавать равносторонний треугольник со сторонами равными единице);

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Треугольник», и создающий треугольник, стороны которого равны максимальной стороне исходного треугольника.

Создать неинициализированный статический массив указателей на треугольники, задать их стороны и вывести на экран является ли треугольник прямоугольным.

20.Определить класс «Треугольник» с закрытыми элементами a, b и c (стороны треугольника). Определить методы класса, обеспечивающие доступ к этим переменным. Определить метод, определяющий, является ли треугольник равнобедренным. Создать такие конструкторы для класса:

– конструктор инициализации (по умолчанию создавать равносторонний треугольник со сторонами равными единице);

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Треугольник», и создающий треугольник, стороны которого равны среднему арифметическому сторон исходного треугольника.

Создать инициализированный статический массив указателей на треугольники и вывести на экран является ли треугольник равнобедренным.

21.Определить класс «Треугольник» с закрытыми элементами a, b и c (стороны треугольника). Определить методы класса, обеспечивающие доступ к этим переменным. Определить метод, определяющий, является ли треугольник Египетским. Создать такие конструкторы для класса:

– конструктор инициализации (по умолчанию создавать равносторонний треугольник со сторонами равными единице);

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Треугольник», и создающий треугольник, стороны которого равны модулю разности сторон исходного треугольника, но не 0 (в этом случае стоит принимать стороны равные 1).

Создать неинициализированный статический массив треугольников, задать их стороны и вывести на экран является ли треугольник Египетским.

22.Определить класс «Одномерный массив чисел» с закрытыми элементами mas – массив чисел и n – число элементов массива. Определить методы класса, обеспечивающие доступ к этим переменным. Создать такие конструкторы для класса:

– конструктор инициализации с инициализацией n через ввод, а mas – при помощи генератора случайных чисел;

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Одномерный массив чисел», и создающий перевёрнутый массив.

Определить метод, определяющий максимальный элемент в массиве.

Определить деструктор класса.

Создать инициализированный статический массив объектов (массивов) и вывести на экран максимальные элементы каждого массива.

23.Определить класс «Одномерный массив чисел» с закрытыми элементами mas – массив чисел и n – число элементов массива. Определить методы класса, обеспечивающие доступ к этим переменным. Создать такие конструкторы для класса:

– конструктор инициализации с инициализацией данных при помощи передачи параметров;

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Одномерный массив чисел», и создающий массив, состоящий только из чётных элементов исходного массива.

Определить метод, определяющий минимальный элемент в массиве.

Определить деструктор класса.

Создать неинициализированный динамический массив объектов (массивов), инициализировать их и вывести на экран минимальные элементы каждого массива.

24.Определить класс «Одномерный массив чисел» с закрытыми элементами mas – массив чисел и n – число элементов массива. Определить методы класса, обеспечивающие доступ к этим переменным. Создать такие конструкторы для класса:

– конструктор инициализации с инициализацией n через ввод (по умолчанию n=10), а mas – при помощи генератора случайных чисел;

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Одномерный массив чисел», и создающий массив, состоящий только из нечётных элементов исходного массива.

Определить метод, определяющий среднее арифметическое элементов в массиве.

Определить деструктор класса.

Создать неинициализированный статический массив указателей на массивы, инициализировать массивы и вывести на экран среднее арифметическое каждого массива.

25.Определить класс «Одномерный массив чисел» с закрытыми элементами mas – массив чисел и n – число элементов массива. Определить методы класса, обеспечивающие доступ к этим переменным. Создать такие конструкторы для класса:

– конструктор инициализации с инициализацией n через ввод (по умолчанию n=5), а mas – при помощи генератора случайных чисел;

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Одномерный массив чисел», и создающий массив, элементы которого являются суммой смежных элементов исходного массива.

Определить метод, определяющий число элементов в массиве, которые меньше числа заданного пользователем.

Определить деструктор класса.

Создать инициализированный статический массив указателей на массивы и вывести на экран число элементов каждого массива, которые меньше числа заданного пользователем.

26.Определить класс «Одномерный массив чисел» с закрытыми элементами mas – массив чисел и n – число элементов массива. Определить методы класса, обеспечивающие доступ к этим переменным. Создать такие конструкторы для класса:

– конструктор инициализации (по умолчанию задавать массив из 10 элементов, каждый элемент в диапазоне от 20 до 80);

– конструктор копирования, принимающий в качестве параметра ссылку на объект класса «Одномерный массив чисел», и создающий массив, элементы которого являются произведением смежных элементов исходного массива.

Определить метод, определяющий число элементов в массиве, которые больше числа заданного пользователем.

Определить деструктор класса.

Создать неинициализированный статический массив объектов (массивов), инициализировать массивы и вывести на экран число элементов каждого массива, которые больше числа заданного пользователем.

27.Определить класс «Меню» с закрытыми элементами str_men – массив названий пунктов меню и n – число элементов массива. Определить методы класса, обеспечивающие доступ к этим переменным. Создать такие конструкторы для класса:

– конструктор по умолчанию (создающий массив из пунктов меню «Открыть», «Закрыть», «Сохранить»);

– конструктор копирования, получающий в качестве параметра ссылку на объект класса «Меню», и добавляющий к нему пункты «Выход» и «О программе».

Реализовать возможность добавления нового пункта меню.

Создать инициализированный статический массив меню и отобразить их.

28.Определить класс «Множество» с закрытыми элементами mas – массив с элементами множества и n – число элементов массива. Определить методы класса, обеспечивающие доступ к этим переменным. Создать такие конструкторы для класса:

– конструктор инициализации (по умолчанию создавать пустое множество);

– конструктор копирования, получающий в качестве параметра ссылку на объект класса «Множество», и создающий множество, в котором ни один элемент не принадлежит исходному множеству.

Реализовать методы добавления во множество новых элементов.

Создать неинициализированный динамический массив множеств и и вывести на экран все множества.

29.Определить класс «Множество» с закрытыми элементами mas – массив с элементами множества и n – число элементов массива. Определить методы класса, обеспечивающие доступ к этим переменным. Создать такие конструкторы для класса:

– конструктор инициализации (по умолчанию создавать множество из 10 случайных элементов);

– конструктор копирования, получающий в качестве параметра ссылку на объект класса «Множество», и создающий множество, в котором все элементы сдвинуты на одну позицию по отношению к исходному множеству (использовать побитовую операцию <<).

Реализовать методы удаления из множества указанного элемента.

Создать неинициализированный статический массив указателей на множества, инициализировать множества и вывести на экран все множества.

30.Определить класс «Множество» с закрытыми элементами mas – массив с элементами множества и n – число элементов массива. Определить методы класса, обеспечивающие доступ к этим переменным. Создать такие конструкторы для класса:

– конструктор инициализации с инициализацией через ввод;

– конструктор копирования, получающий в качестве параметра ссылку на объект класса «Множество», и создающий множество, в котором все

элементы сдвинуты на одну позицию по отношению к исходному множеству (использовать побитовую операцию >>).

Реализовать метод поиска заданного элемента во множестве.

Создать инициализированный статический массив указателей на множества и вывести на экран все множества.

Уровень Б (+1 балл)

1. Определить класс «Квадратная матрица», обладающий такими элементами: размерность матрицы, указатель на тип данных элементов матрицы. Определить такие конструкторы:

– конструктор инициализации (по умолчанию создавать единичную матрицу);

– конструктор копирования, принимающий в качестве параметров ссылку на объект класса «Квадратная матрица» и коэффициент пропорциональности, и создающий матрицу, являющую произведением коэффициента и исходной матрицы.

Определить метод класса, вычисляющий детерминант матрицы. Вычисление детерминанта должно проводиться за приемлемое время для матриц размерности (100x100).

Определить метод отображения матрицы на экран и записи в файл. Определить деструктор класса.

2. Разработать класс «Комплексные числа». Предусмотреть в классе несколько конструкторов: с возможностью задания только действительной части, только комплексной, конструктор по умолчанию, и конструктор инициализатор с инициализацией при помощи списка инициализации для пары комплексных чисел. Предусмотреть возможность выведения комплексного числа в стандартном виде.

Реализовать для класса возможности: сложения, вычитания, умножения, деления, возведения в степень (целую, положительную) комплексных чисел.

3. Определить класс «Человек» с заданием атрибутов: рост, вес, возраст. Предусмотреть для класса конструктор-инициализатор (с инициализацией при помощи списка инициализации), конструктор копии (создающий объект с «идеальным весом» при заданном росте и возрасте).

Разработать метод класса, который рассчитывает идеальный вес для указанного возраста и роста. Реализовать метод, отображающий разницу между фактическим и идеальным весом.

4. Определить класс «Список». Реализовать для класса:

– конструктор инициализации, с инициализацией через передачу параметров;

– конструктор по умолчанию, создающий пустой список;

– конструктор копирования, создающий список, обратный исходному.

Реализовать в классе методы добавления нового элемента (в указанную позицию), поиска заданного элемента, вывод на печать элемента с заданным индексом, вывод списка в прямом и обратном порядке.

5. Определить класс «Система линейных уравнений», обладающий такими элементами: размерность системы, указатель на тип данных элементов матрицы коэффициентов, указатель на тип данных элементов вектора свободных членов. Предусмотреть в классе:

– конструктор по умолчанию (размерность системы 3, коэффициенты и свободные члены задаются случайным образом);

– конструктор инициализации с инициализацией через ввод.

Реализовать в классе метод нахождения корней системы уравнений и метод отображения результатов. Определить деструктор класса.

6. Определить класс «Слова», с двумя закрытыми элементами-строками s1 и s2. Предусмотреть в классе:

– конструктор инициализации, для ввода строк с клавиатуры (по умолчанию строки пусты).

Реализовать в классе метод проверки можно ли из букв, входящих в одно слово, составить другое (каждая буква используется только один раз). Словарь задать самостоятельно или подключить существующий.

Определить деструктор класса.

7. Определить класс «Рифмы», с закрытым элементом-строкой s1. Предусмотреть в классе:

– конструктор инициализации, для ввода строки с клавиатуры (по умолчанию строка пуста);

– конструктор копирования, создающий перевернутую строку.

Реализовать в классе метод вывода на экран всех пар слов, образующих рифму.

Определить деструктор класса.

8. Определить класс «Квадратная матрица», обладающий такими элементами: размерность матрицы, указатель на тип данных элементов матрицы. Определить такие конструкторы:

– конструктор инициализации (по умолчанию создавать единичную матрицу);

– конструктор копирования, принимающий в качестве параметров ссылку на объект класса «Квадратная матрица», и создающий матрицу с обратным знаком.

Определить метод класса, вычисляющий ранг матрицы.

Определить метод отображения матрицы на экран и записи в файл. Определить деструктор класса.

9. Определить класс «Алгебраическое выражение», с закрытым элементом-строкой f – которая представляет собой алгебраическое выражение. Предусмотреть в классе:

– конструктор инициализации, для ввода строки с клавиатуры (по умолчанию строка пуста).

Реализовать в классе метод проверки, допустимым ли образом расставлены скобки в алгебраическом выражении *f*, учитывать минимум два типа скобок («(…)» и «[…]»).

Определить деструктор класса.

10. Определить класс «Система линейных уравнений», обладающий такими элементами: размерность системы, указатель на тип данных элементов матрицы коэффициентов, указатель на тип данных элементов вектора свободных членов. Предусмотреть в классе:

– конструктор по умолчанию (размерность системы 3, коэффициенты и свободные члены задаются случайным образом);

– конструктор инициализации с инициализацией через ввод.

Реализовать в классе метод нахождения количества решений системы. Определить деструктор класса.

11. Определить класс «Квадратная матрица», обладающий такими элементами: размерность матрицы, указатель на тип данных элементов матрицы.

Определить такие конструкторы:

– конструктор инициализации (по умолчанию создавать единичную матрицу);

– конструктор копирования, принимающий в качестве параметров ссылку на объект класса «Квадратная матрица» и создающий транспонированную матрицу к исходной.

Определить метод класса, приводящий матрицу к треугольному виду.

Определить метод отображения матрицы на экран и записи в файл. Определить деструктор класса.

12. Определить класс «Квадратная матрица», обладающий такими элементами: размерность матрицы, указатель на тип данных элементов матрицы. Определить такие конструкторы:

– конструктор инициализации (по умолчанию создавать единичную матрицу);

– конструктор копирования, принимающий в качестве параметров ссылку на объект класса «Квадратная матрица» и коэффициент пропорциональности, и создающий матрицу, являющую произведением коэффициента и исходной матрицы.

Определить метод класса, вычисляющий обратную матрицу, за приемлемое время для размерности (100x100).

Определить метод отображения матрицы и обратной матрицы на экран и записи в файл. Определить деструктор класса.

13. Определить класс «Быки и коровы», с закрытым элементом: *r* – случайное число. Определить такие конструкторы:

– конструктор инициализации случайным образом задающий пятизначное целое число с разными цифрами.

Определить метод для угадывания пользователем числа, согласно правилам. На каждом шаге пользователь вводит пятизначное число, а программа сообщает, сколько цифр числа угадано (быки) и сколько цифр угадано и стоит не на нужном месте (коровы).

Реализовать подсчет очков: +2 за быка, +1 за корову.

14. Определить класс «Ханойские башни», с закрытыми элементами: *t1, t2, t3* – массивы заданной размерности, *n* – размерность массивов. Определить такие конструкторы:

– конструктор инициализации, динамически создающий массивы и записывающий в *t1* числа от *n* до 1 по убыванию;

– конструктор копирования, принимающий в качестве параметров ссылку на объект класса «Ханойские башни», сохраняющий текущее состояние массивов *t1, t2, t3*.

Массивы *t1, t2, t3* – представляют собой три стержня; на *t1* нанизаны круги разного диаметра в виде пирамидки.

Определить метод для перекладывания кругов из *t1* на *t3* используя *t2*. При этом круги нужно класть так, чтобы круг с меньшим диаметром всегда был сверху.

Определить деструктор класса.

15. Определить класс «Тренажёр клавиатуры», с закрытым элементом-строкой s1 и закрытым элементом *p* – счет пользователя. Предусмотреть в классе:

– конструктор инициализации, для генерации случайной строки, не менее 50-ти символов;

– конструктор копирования, создающий перевернутую строку.

Реализовать в классе метод для проверки умения пользователя печатать, не глядя на клавиатуру.

За каждую ошибку пользователю начисляется штрафной балл.

Вывести итоговый счет пользователя.

Определить деструктор класса.

16. Определить класс «Квадратная матрица», обладающий такими элементами: размерность матрицы *n*, указатель на тип данных элементов матрицы. Определить такие конструкторы:

– конструктор инициализации (по умолчанию создавать единичную матрицу);

– конструктор копирования, принимающий в качестве параметров ссылку на объект класса «Квадратная матрица» и создающий транспонированную матрицу к исходной.

Определить метод класса, транспонирующий матрицу относительно обратной диагонали.

Определить метод отображения матрицы на экран. Определить деструктор класса.

17. Определить класс «Выражения», с двумя закрытыми элементами-строками s1 и s2. Предусмотреть в классе:

– конструктор инициализации, для ввода строк с клавиатуры (алгебраических выражений высокого порядка).

Реализовать в классе методы сложения и вычитания этих выражений.

Пользователь задает выражения в следующем виде, один из примеров: *x^5+3x^3-x^2-8+5x^2=0.*

Порядок выражений ограничен 10.

Определить деструктор класса.

18. Определить класс «Лабиринт», с открытым элементом: массив клеточек – *A* (40x40). Определить такие конструкторы:

– конструктор-инициализатор генерирует случайным образом лабиринт (*A*[*i*,*j*]=0, если клетка [*i*,*j*] "проходима"; *A*[*i*,*j*]=1, если клетка [*i*,*j*] "непроходима"), боковые строки и столбцы всегда "непроходимы", кроме одной клеточки.

Определить метод для поиска выхода из лабиринта из заданной клеточки или сообщить о его отсутствии. Лабиринт и путь отобразить в текстовом файле.

19. Определить класс «Генератор судоку», обладающий такими элементами: матрицы клеток – A (9x9). Определить такие конструкторы:

– конструктор инициализации, позволяющий задать сложность головоломки «Судоку» (процент заполненных клеток) и генерирующий решаемую головоломку «Судоку».

Определить методы отображения головоломки и сохранения ее в текстовый файл.

При необходимости, определить деструктор класса.

20. Определить класс «Ориентированный граф», обладающий такими элементами: размерность матрицы смежности n, указатель на тип данных элементов матрицы смежности. Определить такие конструкторы:

– конструктор инициализации, позволяющий пользователю задать весовую матрицу смежности (расстояния) определенного размера.

Определить метод класса, находящий кратчайший путь и его стоимость с первой вершины в последнюю, или выводящий сообщение об отсутствии такого пути.

Определить деструктор класса.

21. Определить класс «Ориентированный граф», обладающий такими элементами: размерность матрицы смежности n, указатель на тип данных элементов матрицы смежности. Определить такие конструкторы:

– конструктор инициализации, позволяющий пользователю задать матрицу смежности определенного размера.

Определить метод класса, находящий все циклы в графе или выводящий сообщение об их отсутствии.

Определить деструктор класса.

22. Определить класс «Ориентированный граф», обладающий такими элементами: размерность матрицы смежности n, указатель на тип данных элементов матрицы смежности. Определить такие конструкторы:

– конструктор инициализации, позволяющий пользователю задать весовую матрицу смежности (расстояния) определенного размера.

Определить метод класса, находящий максимальный поток транспортной сети, заданной этим ориентированным графом, из источника в сток.

Определить деструктор класса.

23. Определить класс «Ориентированный граф», обладающий такими элементами: размерность матрицы смежности *n*, указатель на тип данных элементов матрицы смежности. Определить такие конструкторы:

– конструктор инициализации, позволяющий пользователю задать матрицу смежности определенного размера.

Определить метод класса, цикл, который включает в себя все вершины ровно по 1 разу или выводящий сообщение о его отсутствии.

Определить деструктор класса.

24. Определить класс «Ориентированный граф», обладающий такими элементами: размерность матрицы смежности *n*, указатель на тип данных элементов матрицы смежности. Определить такие конструкторы:

– конструктор инициализации, позволяющий пользователю задать матрицу смежности определенного размера.

Определить метод класса, цикл, который включает в себя все ребра ровно по 1 разу или выводящий сообщение о его отсутствии.

Определить деструктор класса.

25. Определить классы «Каталог игр для PC», «Клиенты», и «Заказы». В классе «Каталог игр для PC» реализовать структуру для хранения информации об: идентификаторе игры, названии игры, годе выпуска, системных требованиях (одной строкой), цене. В классе «Клиенты» реализовать структуру для хранения информации об: идентификаторе клиента, ФИО, номер телефона. В классе «Заказы» реализовать структуру для хранения информации об: игре (идентификатор), клиенте (идентификатор) и дате заказа. Определить такие конструкторы:

– конструктор по умолчанию, создающий пустые структуры.

Определить методы добавления данных в структуру каждого из классов. Предусмотреть проверки правильности добавления. Определить методы отображения данных структур каждого класса.

Определить метод позволяющий отобразить данные о: названии игры, ФИО пользователя и дате заказа игры пользователем.

Определить деструкторы классов.

26. Определить класс «Квадратная матрица», обладающий такими элементами: размерность матрицы, указатель на тип данных элементов матрицы. Определить такие конструкторы:

– конструктор инициализации (по умолчанию создавать единичную матрицу);

– конструктор копирования, принимающий в качестве параметров ссылку на объект класса «Квадратная матрица» и создающий транспонированную матрицу к исходной.

Определить метод класса, приводящий матрицу к ортогональному виду.

Определить метод отображения матрицы на экран. Определить деструктор класса.

Уровень В (+2 балла)

1. Определить класс «Игра в кости», с закрытым элементом: массив игральных костей (по 6 граней) размерности *n*. Определить такие конструкторы:

– конструктор инициализации, создающий доску (стол) для игры в кости (вариацию игры выбрать самостоятельно), в виде матрицы необходимой размерности.

Определить методы для имитации игры в кости между двумя пользователями компьютерами. Доказать или опровергнуть экспериментальным путем теорию, о том что при случайном выпадении значений на костях процент побед каждого игрока будет ~ 50%.

Собранную статистику сохранить в файл.

Определить деструктор класса.

2. Определить класс «Игра в кости», с закрытым элементом: массив игральных костей (по 6 граней) размерности *n*. Определить такие конструкторы:

– конструктор инициализации, создающий доску (стол) для игры в кости (вариацию игры выбрать самостоятельно), в виде матрицы необходимой размерности.

Определить методы для игры в кости между пользователем и ЭВМ.

Реализовать подсчет статистики игрока и учет рекордов с сохранением в файл.

Определить деструктор класса.

3. Определить класс «Судоку», обладающий такими элементами: матрицы клеток – *A* (9x9). Определить такие конструкторы:

– конструктор по умолчанию, генерирующий решаемую головоломку «Судоку», головоломка должна быть заполнена не более чем на 35%;

– конструктор инициализации, позволяющий задать условия головоломки «Судоку».

Определить метод для нахождения решения головоломки, с сохранением условия и решения в файл.

При необходимости, определить деструктор класса.

4. Определить класс «Японский кроссворд», обладающий такими элементами: размерность *n*, матрицы клеток – *A* (*n*x*n*). Определить такие конструкторы:

– конструктор инициализации, позволяющий задать размерность и условие кроссворда.

Определить метод для нахождения решения кроссворда, с сохранением условия и решения в файл.

Символы для закрашенных клеточек выбрать самостоятельно.

Определить деструктор класса.

5. Определить класс «Генератор кроссвордов», обладающий такими элементами: размерность кроссворда *n*, количество слов кроссворда *m*, указатель на тип данных элементов матрицы клеток – *A*. Определить такие конструкторы:

– конструктор инициализации, позволяющий задать размерность и количество слов в кроссворде.

Определить метод генерации кроссворда, на основании введенных пользователем данных или вывести сообщение о невозможности генерации такого кроссворда. Сохранить в файл кроссворд и описание слов.

Слова и их описание брать с подключаемого или собственного словаря.

Определить деструктор класса.

6. Определить класс «Пять в ряд», обладающий такими элементами: размерность поля *n*, указатель на тип данных элементов матрицы поля – *A*. Определить такие конструкторы:

– конструктор инициализации, создающий матрицу для игры в «пять в ряд» заданного размера.

Определить методы для имитации игры в «пять в ряд» (вариация игры крестики-нолики, где нужно поставить горизонтально, вертикально или по диагонали 5 крестиков или ноликов) между двумя пользователями компьютерами. Доказать или опровергнуть экспериментальным путем теорию, о том что при правильной расстановке крестиков и ноликов, побеждать будет или игрок начинающий первым, или постоянно будет ничья. В эксперименте можно учитывать и парную/непарную размерность поля.

Собранную статистику сохранить в файл.

Определить деструктор класса.

7. Определить класс «Блэкджек», с закрытым элементом: массив игроков размерности *n*, не более 8 (элементами массива должны выступать игроки с набором карт и фишек, для этого стоит создать еще один класс

«Игрок» с соответствующими атрибутами и методами). Определить такие конструкторы:

– конструктор инициализации, задающий начальную раздачу карт для игроков и дилера.

Определить методы для игры в Блэкджек между пользователем и другими игроками-компьютерами. За дилера играет компьютер.

Дополнительно предусмотреть все необходимые атрибуты.

Реализовать запись процесса игры в текстовый файл.

Определить деструктор класса.

8. Определить класс «Длинная арифметика», с двумя закрытыми элементами *a* и *b*. *a* и *b* – длинные целые числа, т.е. числа, значения которых превышают максимально (минимально) допустимое значение целочисленного типа (MIN_INT и MAX_INT). Предусмотреть в классе:

– конструктор инициализатор, для ввода чисел с клавиатуры (по умолчанию числа равны нулю).

– конструктор копирования, создающий числа с обратным знаком.

Реализовать в классе методы нахождения результатов операций: +,-,*,/ для чисел *a* и *b*, возвести в степень, найти квадратный корень для числа *a*.

Определить деструктор класса.

9. Определить класс «Системы счислений», обладающий такими закрытыми элементами: число *a* – целое число в некоторой системе счисления (2,8,10,16), *q* – основание системы счисления. Предусмотреть в классе:

– конструктор инициализации, для ввода числа с клавиатуры (по умолчанию число равно нулю, система десятичная).

– конструктор копирования, создающий число с обратным знаком.

Реализовать в классе методы доступа к числу *a* и преобразования числа в систему счисления с любым другим основанием от 2 до 16 включительно.

Определить деструктор класса.

10. Определить класс «Криптография», с закрытым элементом *t* – который представляет собой некоторый текст. Предусмотреть в классе:

– конструктор инициализации, для ввода текста с клавиатуры (задать некоторый текст по умолчанию).

Класс должен содержать методы для шифровки и расшифровки текста по трем алгоритмам или более. Сымитировать атаку криптоаналитика (имеется зашифрованный текст, делается предположение о методе, которым он зашифрован, подбирается ключ и расшифровывается текст). Сравнить время, затраченное на подбор ключей для разных методов.

Определить деструктор класса.

11. Определить класс «Точки», обладающий такими элементами: массив точек на плоскости – *A*, количество точек – *n*. Определить такие конструкторы:

– конструктор инициализации;

– конструктор копии, принимающий в качестве параметров ссылку на объект класса «Точки», и создающий массив зеркальных точек.

Определить метод для нахождения уравнения прямой, которая делит это множество точек на два равномощных подмножества.

Определить деструктор класса.

12. Определить класс «Пятнашки», с закрытым элементом: массив ячеек – *A* (4x4). Определить такие конструкторы:

– конструктор инициализации случайным образом расставляющий 15 фишек от 1 до 15 по ячейкам, одна ячейка остается свободной.

Определить метод для расставления фишек по возрастанию их номеров, при том, что передвигать фишки можно только на соседнюю свободную ячейку.

13. Определить класс «Морской бой», с закрытыми элементами: две матрицы клеток – *A* (10x10). Определить такие конструкторы:

– конструктор инициализации случайным образом расставляющий 4 корабля по 1 клетке, 3 корабля по 2 клетки, 2 корабля по 3 клетки, 1 корабль в 4 клетки (вертикально, горизонтально или по диагонали) для двух игроков, с тем условием, что между кораблями должна быть пустая клетка.

Определить методы для игры в морской бой между пользователем и ЭВМ.

Реализовать подсчет статистики игрока и учет рекордов с сохранением в файл.

Клетки для атаки противником игрока выбираются случайно, однако если есть фактор попадания во вражеский корабль, то следующей берется случайная соседняя клетка, а если корабль уничтожен, то соседние с кораблем клетки больше не выбираются.

Определить деструктор класса.

Предусмотреть возможность повтора игры по желанию пользователя.

14. Определить класс «Морской бой», с закрытыми элементами: два массива клеток – *A* (10x10). Определить такие конструкторы:

– конструктор инициализации случайным образом расставляющий 4 корабля по 1 клетке, 3 корабля по 2 клетки, 2 корабля по 3 клетки, 1 корабль в 4 клетки (вертикально, горизонтально или по диагонали) для двух игроков, с тем условием, что между кораблями должна быть пустая клетка.

Определить методы для имитации игры в морской бой между двумя игроками компьютерами.

Доказать или опровергнуть экспериментальным путем теорию, о том что, при случайной расстановке кораблей и случайном выборе клеток для атаки противников процент побед будет равным 50.

Клетки для атаки противника выбираются случайно, однако если есть фактор попадания во вражеский корабль, то следующей берется случайная

соседняя клетка, а если корабль уничтожен, то соседние с кораблем клетки больше не выбираются.

Определить деструктор класса.

15. Определить класс «Алгебраическое выражение», с закрытым элементом-строкой *fx* – которая представляет собой алгебраическое выражение одной переменной. Предусмотреть в классе:

– конструктор инициализации, для ввода строки с клавиатуры (по умолчанию строка пуста).

Реализовать в классе метод нахождения значения алгебраического выражения при заданной переменной. Для упрощения выполнения использовать список (стек).

Определить деструктор класса.

16. Определить класс «Алгебраическое выражение», с закрытым элементом-строкой *fx* – которая представляет собой алгебраическое выражение одной переменной. Предусмотреть в классе:

– конструктор инициализации, для ввода строки с клавиатуры (по умолчанию строка пуста).

Реализовать в классе метод упрощения алгебраического выражения, если это возможно.

Например: $x+x \rightarrow 2x$ и т.д.

Определить деструктор класса.

17. Определить класс «Алгебраическое выражение», с закрытым элементом-строкой *fx* – которая представляет собой алгебраическое выражение одной переменной. Предусмотреть в классе:

– конструктор инициализации, для ввода строки с клавиатуры (по умолчанию строка пуста).

Реализовать в классе метод нахождения производной алгебраического выражения.

Определить деструктор класса.

18. Определить класс «Алгебраическое выражение», с закрытым элементом-строкой *fx* – которая представляет собой алгебраическое выражение одной переменной. Предусмотреть в классе:

– конструктор инициализации, для ввода строки с клавиатуры (по умолчанию строка пуста).

Реализовать в классе метод нахождения первообразной алгебраического выражения.

Определить деструктор класса.

19. Определить класс «Ориентированный граф», обладающий такими элементами: размерность матрицы смежности *n*, указатель на тип данных элементов матрицы смежности. Определить такие конструкторы:

– конструктор инициализации, позволяющий пользователю задать весовую матрицу смежности (расстояния) определенного размера.

Определить метод класса, находящий все пути между двумя заданными пользователем вершинами в графе, при этом они не должны иметь общих вершин кроме начальной и конечной.

Определить деструктор класса.

20. Определить класс «Шахматная доска», с закрытым элементом: массив клеточек – *A* (8x8). Определить такие конструкторы:

– конструктор инициализации, помещающий в заданную клеточку шахматную фигуру – конь.

Определить метод для поиска путей коня к четырем угловым клеточкам доски за наименьшее число ходов. Шахматная фигура конь перемещается на 1 клеточку по горизонтали и 2 по вертикали или на 2 клеточки по горизонтали и 1 клеточку по вертикали.

Если за начальную клеточку задана угловая, найти пути к трем оставшимся.

21. Определить класс «Ферзи», с закрытым элементом: массив клеточек – *A* (8x8). Определить такие конструкторы:

– конструктор инициализации, помещающий в случайные 8 клеточек 8 шахматных фигур – ферзей.

Определить метод для поиска такого расположения ферзей, чтобы они не били друг друга. Шахматная фигура ферзь может бить фигуры по вертикали, горизонтали и диагонали.

За одну итерацию можно двигать только одного ферзя. Сохранить все промежуточные перестановки ферзей в текстовый файл.

22. Определить класс «Неизвестное слово», с закрытым элементом-строкой s1 и закрытым элементом *p* – счет пользователя. Предусмотреть в классе:

– конструктор инициализации, для выбора слова из подключенного словаря (может быть обычным текстовым файлом).

Реализовать в классе метод угадывания пользователем слова по буквам.

За каждую правильно названную букву пользователю начисляется от 100 до 1000 очков, за неправильно названную снимается 200. Если в слове несколько одинаковых букв, при угадывании они открываются сразу, а число очков, умножается на количество этих букв.

Для каждого слова из словаря организовать подсказки.

Вывести итоговый счет пользователя.

Определить деструктор класса.

23. Определить класс «Лабиринт», с открытым элементом: массив клеточек – *A* (40x40). Определить такие конструкторы:

– конструктор инициализации, генерирующий случайным образом лабиринт (*A*[*i*,*j*]=0, если клетка [*i*,*j*] "проходима"; *A*[*i*,*j*]=1, если клетка [*i*,*j*] "непроходима"), боковые строки и столбцы всегда "непроходимы", кроме одной клеточки.

Конструктор обязательно должен генерировать проходимый лабиринт.

Определить метод для поиска всех, разных, путей выхода из лабиринта из заданной клеточки. Лабиринт и пути сохранить в текстовом файле.

24. Определить класс «Кольца» обладающий такими элементами: количество колец *n,* указатель на тип данных элементов матрицы взаимодействия колец *A*. Определить такие конструкторы:

– конструктор инициализации, генерирующий случайным образом матрицу взаимодействия колец (*A*[*i,j*]=1, если кольца [*i,j*] " сцеплены друг с другом"; *A*[*i,j*]=0, если кольца [*i,j*] "не сцеплены").

Определить метод, удаляющий минимальное количество колец для получения цепочки (каждое кольцо сцеплено с двумя другими). Начальное условие и результат сохранить в текстовый файл.

4 ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет подается после полной сдачи и защиты лабораторной работы в электронном виде (документ Word).

Отчет должен быть оформлен согласно ДСТУ 3008-95.

В отчет должен содержать следующие пункты:

- титульный лист;
- содержание;
- цель работы;
- постановка задачи;
- аналитические выкладки, если необходимо;
- пошаговый алгоритм решения расчётной части задачи и подзадач (при необходимости);
- UML-диаграмму классов;
- исходный код программы с комментариями;
- примеры работы программы;
- выводы.

Отчет оценивается максимально в 0,5 балла.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое встраиваемая функция?
2. В чем состоит механизм перегрузки функций?
3. Для чего нужен nullptr?
4. Что такое rvalue ссылка?
5. Что такое класс?
6. Что такое инкапсуляция данных?
7. Что такое элементы-данные и функции-элементы?
8. Назначение и виды конструкторов.
9. Назначение деструктора.
10. Способы создания объектов классов.
11. Неявный параметр this.