

№ 2 Программирование объектов и классов

Определить пользовательский класс в соответствии с вариантом задания (смотри варианты ниже).

- 1) определить в классе следующие конструкторы: без параметров, с параметрами, копирования;**
- 2) определить закрытый конструктор; предложите варианты его вызова;**
- 3) определить в классе деструктор, компоненты-функции для просмотра и установки полей данных: `setТип()`, `getТип()` (помним про инкапсуляцию и проверку корректности задания параметров);**
- 4) определить константную переменную в классе - пояснить как она может быть использована;**
- 5) определите статическую переменную и статический метод - поясните их назначение;**
- 6) написать демонстрационную программу, в которой создаются и разрушаются объекты (от 3 до 5 шт.) и указатели на объекты пользовательского класса и каждый вызов конструктора и деструктора сопровождается выдачей соответствующего сообщения (какой объект какой конструктор или деструктор вызвал).**
- 7) не забывайте выполнять задание выделенное курсивом в варианте.**

Вариант 1	<p>Создать класс Вектор, который имеет указатель на int, число элементов и переменную состояния. Определить функцию, которая присваивает элементу массива некоторое значение (параметр по умолчанию), функцию, которая получает некоторый элемент массива. В переменную состояния устанавливать код ошибки, при нехватке памяти, выхода за пределы массива и т.п.. Определить функции: вывода консоль, сложения и умножения, которые производят эти арифметические операции с данными класса вектор и числом int.</p> <p>Создать массив объектов. Вывести:</p> <p>a) список векторов, содержащих 0;</p> <p>b) список векторов с наименьшим модулем.</p>
Вариант 2	<p>Создать класс Стек вещественных. Функции-члены вставляют элемент в стек, вытаскивают элемент из стека, печатают стек, проверяет вершину стека.</p> <p>Создать массив объектов. Вывести:</p> <p>a) стеки с наименьшим/наибольшим верхним элементом;</p> <p>b) список стеков, содержащих отрицательные элементы.</p>
Вариант 3	<p>Создать класс типа - Множество. Функции-члены: добавляют элемент к множеству, удаляют элемент, отображают элементы множества от начала и от конца, выводят текущее количество элементов множества.</p> <p>Создать массив объектов. Вывести:</p> <p>a) множества с наименьшей/наибольшей суммой элементов;</p> <p>b) список множеств, содержащих отрицательные элементы.</p>
Вариант 4	<p>Создать класс типа - Дата с полями: день (1-31), месяц (1-12), год (целое число). Функции-члены класса: установки дня, месяца и года; получения дня, месяца и года, а также две функции-члены печати: печать по шаблону: "5 января 1997 года" и "05/01/1997". Функции-члены установки полей класса и конструкторы должны проверять корректность задаваемых параметров.</p> <p>Создать массив объектов. Вывести:</p> <p>a) список дат для заданного года;</p> <p>b) список дат, которые имеют заданное число;</p>
Вариант 5	<p>Создать класс Строка (динамическая). Функции-члены класса: вывода строки, вывода длины строки, проверки существует ли в строке заданный символ, функция замены одного символа в строке на другой.</p> <p>Создать массив объектов. Вывести:</p> <p>a) список строк определенной длины;</p> <p>b) список строк, которые содержат заданное слово.</p>

Вариант 6	<p>Определить класс Булева матрица (BoolMatrix). Реализовать методы для логического сложения (дизъюнкции), умножения и инверсии матриц. Реализовать методы для подсчета числа единиц в матрице и упорядочения строк в лексикографическом порядке. Создать массив объектов. Вывести:</p> <p><i>a) список матриц с наибольшим/наименьшим количеством единиц;</i></p> <p><i>b) список матриц с равным количеством нулей в каждой строке</i></p>
Вариант 7	<p>Создать класс типа - Окружность. Поля – координаты центра, радиус. Функции-члены вычисляют площадь, длину окружности, устанавливают поля и возвращают значения. Функции-члены установки полей класса должны проверять корректность задаваемых параметров. Создать массив объектов. Вывести:</p> <p><i>a) группы окружностей, центры которых лежат на одной прямой;</i></p> <p><i>b) наибольший и наименьший по площади (периметру) объект;</i></p>
Вариант 8	<p>Создать класс типа - Прямоугольник. Поля - высота и ширина. Функции-члены вычисляют площадь, периметр, устанавливают поля и возвращают значения. Функции-члены установки полей класса должны проверять корректность задаваемых параметров. Создать массив объектов. Вывести:</p> <p><i>a) количество четырехугольников разного типа (квадрат, прямоугольник, ромб, произвольный)</i></p> <p><i>b) определить для каждой группы наибольший и наименьший по площади (периметру) объект.</i></p>
Вариант 9	<p>Определить класс Вектор. Реализовать методы для вычисления модуля вектора, скалярного произведения, сложения, вычитания, умножения на константу. Написать метод, который для заданной пары векторов будет определять, являются ли они коллинеарными или ортогональными. Создать массив объектов. Вывести:</p> <p><i>a) вектора с заданным модулем;</i></p> <p><i>b) определить вектор с наибольшей/наименьшей суммой элементов.</i></p>

Вариант 10	<p>Построить класс Булев вектор (BoolVector). Реализовать методы для выполнения поразрядных конъюнкции, дизъюнкции и отрицания векторов, а также подсчета числа единиц и нулей в векторе.</p> <p><i>Создать массив объектов. Вывести:</i></p> <p><i>a) вектора с заданным числом единиц/нулей;</i></p> <p><i>b) определить и вывести равные вектора.</i></p>
Вариант 11	<p>Создать класс типа - Время с полями: часы (0-23), минуты (0-59), секунды (0-59). Класс имеет конструктор. Функции-члены установки времени, получения часа, минуты и секунды, а также две функции-члены печати: по шаблону: "16 часов 18 минут 3 секунды" и "4:18:3". Функции-члены установки полей класса должны проверять корректность задаваемых параметров.</p> <p><i>Создать массив объектов. Вывести:</i></p> <p><i>a) время с заданным числом часов;</i></p> <p><i>b) список времен по группам: ночь, утро, день, вечер.</i></p>
Вариант 12	<p>Создать класс - Одномерный массив целых чисел (вектор). Функции-члены обращаются к отдельному элементу массива по индексу(ввод-вывод значений), вывода значений массива на экран, поэлементного сложения и вычитания со скалярным значением.</p> <p><i>Создать массив объектов. Вывести:</i></p> <p><i>a) массивы только с четными/нечетными элементами;</i></p> <p><i>b) массив с наибольшей суммой элементов.</i></p>
Вариант 13	<p>Создать класс – Треугольник, заданного тремя точками. Функции-члены изменяют точки, обеспечивают вывод на экран координат, рассчитывают длины сторон и периметр треугольника.</p> <p><i>Создать массив объектов.</i></p> <p><i>a) подсчитать количество треугольников разного типа (равносторонний, равнобедренный, прямоугольный, произвольный).</i></p> <p><i>b) определить для каждой группы наибольший и наименьший по периметру объект.</i></p>
Вариант 14	<p>Создать класс - Множество. Функции-члены реализуют добавление и удаление элемента, пересечение и разность множеств, вывод множества.</p> <p><i>Создать массив объектов. Вывести:</i></p> <p><i>a) множества только с четными/нечетными элементами;</i></p> <p><i>b) множества, содержащие отрицательные элементы.</i></p>

Вариант 15	<p>Создать класс Airline: Пункт назначения, Номер рейса, Тип самолета, Время вылета, Дни недели. Функции-члены реализуют запись и считывание полей (проверка корректности).</p> <p><i>Создать массив объектов. Вывести:</i></p> <p><i>a) список рейсов для заданного пункта назначения;</i></p> <p><i>b) список рейсов для заданного дня недели;</i></p>
Вариант 16	<p>Создать класс Student: id, Фамилия, Имя, Отчество, Дата рождения, Адрес, Телефон, Факультет, Курс, Группа. Функции-члены реализуют запись и считывание полей (проверка корректности), расчет возраста студента</p> <p><i>Создать массив объектов. Вывести:</i></p> <p><i>a) список студентов заданного факультета;</i></p> <p><i>d) список учебной группы.</i></p>
Вариант 17	<p>Создать класс Customer: id, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки, баланс. Функции-члены реализуют запись и считывание полей (проверка корректности), зачисление и списание сумм на счет.</p> <p><i>Создать массив объектов. Вывести:</i></p> <p><i>a) список покупателей в алфавитном порядке;</i></p> <p><i>b) список покупателей, у которых номер кредитной карточки находится в заданном интервале.</i></p>
Вариант 18	<p>Создать класс Abiturient: id, Фамилия, Имя, Отчество, Адрес, Телефон, Оценки. Функции-члены реализуют запись и считывание полей (проверка корректности), расчета среднего балла.</p> <p><i>Создать массив объектов. Вывести:</i></p> <p><i>a) список абитуриентов, имеющих неудовлетворительные оценки;</i></p> <p><i>b) список абитуриентов, у которых сумма баллов выше заданной;</i></p>
Вариант 19	<p>Создать класс Book: id, Название, Автор (ы), Издательство, Год издания, Количество страниц, Цена, Тип переплета. Функции-члены реализуют запись и считывание полей (проверка корректности).</p> <p><i>Создать массив объектов. Вывести:</i></p> <p><i>a) список книг заданного автора;</i></p> <p><i>b) список книг, выпущенных после заданного года.</i></p>

Вариант 20	<p>Создать класс House: id, Номер квартиры, Площадь, Этаж, Количество комнат, Улица, Тип здания, Срок эксплуатации. Функции-члены реализуют запись и считывание полей (проверка корректности), расчета возраста здания (необходимость в кап. ремонте). Создать массив объектов. Вывести: a) список квартир, имеющих заданное число комнат; b) список квартир, имеющих заданное число комнат и расположенных на этаже, который находится в заданном промежутке;</p>
Вариант 21	<p>Создать класс Phone: id, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки, Дебет, Кредит, Время городских и междугородных разговоров. Функции-члены реализуют запись и считывание полей (проверка корректности), расчет балланса на текущий момент. Создать массив объектов. Вывести: a) сведения об абонентах, у которых время внутригородских разговоров превышает заданное; b) сведения об абонентах, которые пользовались междугородной связью;</p>
Вариант 22	<p>Создать класс Car: id, Марка, Модель, Год выпуска, Цвет, Цена, Регистрационный номер. Функции-члены реализуют запись и считывание полей (проверка корректности), вывода возраста машины. Создать массив объектов. Вывести: a) список автомобилей заданной марки; b) список автомобилей заданной модели, которые эксплуатируются больше n лет;</p>
Вариант 23	<p>Создать класс Product: id, Наименование, UPC, Производитель, Цена, Срок хранения, Количество. Функции-члены реализуют запись и считывание полей (проверка корректности), вывод общей суммы продукта. Создать массив объектов. Вывести: a) список товаров для заданного наименования; b) список товаров для заданного наименования, цена которых не превосходит заданную;</p>

Вариант 24	<p>Создать класс Train: Пункт назначения, Номер поезда, Время отправления, Число мест (общих, купе, плацкарт, люкс). Функции-члены реализуют запись и считывание полей (проверка корректности), вывод общего числа мест в поезде.</p> <p><i>Создать массив объектов. Вывести:</i></p> <p><i>a) список поездов, следующих до заданного пункта назначения;</i></p> <p><i>b) список поездов, следующих до заданного пункта назначения и отправляющихся после заданного часа;</i></p>
Вариант 25	<p>Создать класс Bus: Фамилия и инициалы водителя, Номер автобуса, Номер маршрута, Марка, Год начала эксплуатации, Пробег. Функции-члены реализуют запись и считывание полей (проверка корректности), вывод возраста автобуса.</p> <p><i>Создать массив объектов. Вывести:</i></p> <p><i>a) список автобусов для заданного номера маршрута;</i></p> <p><i>b) список автобусов, которые эксплуатируются больше заданного срока;</i></p>
Вариант 26	<p>Создать класс Airline: Пункт назначения, Номер рейса, Тип самолета, Время вылета, Дни недели. Функции-члены реализуют запись и считывание полей (проверка корректности).</p> <p><i>Создать массив объектов. Вывести:</i></p> <p><i>a) список рейсов для заданного пункта назначения;</i></p> <p><i>b) список рейсов для заданного дня недели;</i></p>

Примеры программ

Матрица

Matrix.h

```
#pragma once
#include <iostream>
#include <ctime>

class Matrix
{
    int **storage;
    int hsize,vsize;
    static const int maxsize=20;
public:
    Matrix();//
    Matrix(int v,int h);//
    Matrix(const Matrix &);//
    ~Matrix();//
    void sethsize(int);//
    void setvsize(int);//
    int gethsize();//
```

```

    int getvsize();//
    void printall();//
    void print(int,int);//
    void checksize();//
    void randomfill();//
};

```

Matrix.cpp

```

#include "StdAfx.h"
#include "matrix.h"

```

```
Matrix::Matrix()
```

```

{
    std::cout << "Вызов конструктора по умолчанию\n";
    storage=new int*[maxsize];
    for(int i=0;i<maxsize;i++)
        storage[i]=new int[maxsize];
    hsize=vsize=maxsize;
    this->randomfill();
}

```

```
Matrix::Matrix(int v,int h)
```

```

{
    std::cout << "Вызов конструктора с параметрами\n";
    storage=new int*[maxsize];
    for(int i=0;i<maxsize;i++)
        storage[i]=new int[maxsize];
    hsize=h;
    vsize=v;
    this->randomfill();
}

```

```
Matrix::Matrix(const Matrix& m)
```

```

{
    std::cout << "Вызов конструктора копирования\n";
    storage=new int*[maxsize];
    for(int i=0;i<maxsize;i++)
        storage[i]=new int[maxsize];
    hsize=m.hsize;
    vsize=m.vsize;
    for(int i=0;i<maxsize;i++)
        for(int j=0;j<maxsize;j++)
            storage[i][j]=m.storage[i][j];
}

```

```
Matrix::~Matrix()
```

```

{
    std::cout << "Вызов деструктора\n";
    for(int i=0;i<maxsize;i++)
        delete[] storage[i];
    delete[] storage;
}

```



```

int Matrix::gethsize() {return hsize;}

int Matrix::getvsize() {return vsize;}

void Matrix::randomfill()
{
    srand(time(NULL));
    for(int i=0;i<hsize;i++)
        for(int j=0;j<vsize;j++)
            if(storage[i][j]<0) storage[i][j]=rand()%10;
}

void Matrix::printall()
{
    for(int i=0;i<hsize;i++)
    {
        for(int j=0;j<vsize;j++)
            std::cout << storage[i][j] << " ";
        std::cout << std::endl;
    }
    std::cout << std::endl;
}

void Matrix::checksize()
{
    for(int i=0;i<maxsize;i++)
        for(int j=0;j<maxsize;j++)
            if(i>=hsize||j>=vsize) storage[i][j]=-1;
}

void Matrix::sethsize(int h)
{
    hsize=h;
    this->checksize();
    this->randomfill();
}

void Matrix::setvsize(int v)
{
    vsize=v;
    this->checksize();
    this->randomfill();
}

void Matrix::print(int v,int h)
{
    for(int i=0;i<h;i++)
    {
        for(int j=0;j<v;j++)
            std::cout << storage[i][j] << " ";
        std::cout << std::endl;
    }
}

```

```

        std::cout << std::endl;
    }

```

Main_module.cpp

```

#include "stdafx.h"
#include "matrix.h"
#include <iostream>
#include <locale>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, ".1251");
    cout << "Матрица a\n";
    Matrix *a=new Matrix(4,5);
    cout << "Матрица b\n";
    Matrix *b=new Matrix(*a);
    cout << "Матрица a\n";
    a->printall();
    cout << "Изменение размеров\n";
    a->setvsize(6);
    a->sethsize(6);
    cout << "Матрица a\n";
    a->printall();
    cout << "Изменение размеров\n";
    a->setvsize(3);
    a->sethsize(3);
    cout << "Матрица a\n";
    a->printall();
    cout << "Матрица b\n";
    b->printall();
    return 0;
}

```

Вектор

MyVector.h

```

#pragma once
#define NIL          0x00000000
#define NULL         0

class CMyVector
{
private:
    int m_size;
    int* m_vector;
    int m_lastError;
public:

```

```

enum {no_error, memory_limit, exit_array};
CMyVector(void);
CMyVector(int size, ...);
~CMyVector(void);
int getLastError(void);
int getSize(void);
int operator[] (int);
void print(void);
int sum(void);
int product(void);
};

```

MyVector.cpp

```

#include "MyVector.h"
#include <iostream>

CMyVector::CMyVector(void)
{
    m_vector = NIL;
    m_size = 0;
    m_lastError = no_error;
}

CMyVector::CMyVector(int size, ...)
{
    m_vector = new int[size];
    m_size = size;
    int* marker = &size;
    marker++;
    while (size != 0)
    {
        m_vector[m_size-size] = *marker;
        marker++;
        size--;
    }
    m_lastError = no_error;
}

CMyVector::~CMyVector(void)
{
    delete [] m_vector;
}

int CMyVector::getLastError(void)
{
    int vp = m_lastError;
    m_lastError = no_error;
    return vp;
}

int CMyVector::getSize(void)
{
    return m_size;
}

```

```

int CMyVector::operator[](int index)
{
    if (index<0 || index>m_size-1)
    {
        m_lastError = exit_array;
        return NULL;
    }
    return m_vector[index];
}
void CMyVector::print(void)
{
    for (int i = 0; i < m_size; i++) std::cout << m_vector[i]
<< std::ends;
    std::cout << std::endl;
}

int CMyVector::sum(void)
{
    int s = 0;
    for (int i = 0; i < m_size; i++) s += m_vector[i];
    return s;
}

int CMyVector::product(void)
{
    int p = 1;
    for (int i = 0; i < m_size; i++) p *= m_vector[i];
    return p;
}

```

Main_mod.cpp

```

#include "MyVector.h"

int main()
{
    CMyVector v1(5, 1, 2, 3, 4, 5);
    CMyVector v2(3, 1, 2, 3);
    int x = v1[7];
    int error = v1.getLastError();
    x = v1[2];
    x = v1.getSize();
    x = v2.sum();
    x = v2.product();
    v2.print();
    return 0;
}

```

Студент

Student.h

```

#pragma once

```

```

#include "string.h"
class Student
{
    char name[30];
    int age;
public:
    Student();
    Student(char*, int);
    void setAge(int);
    void setName(char*);
    ~Student(void);
    void print();
    char* getName(void);
    int getAge(void);
};

```

Student.cpp

```

Student::Student()
{
}
Student::Student(char* NAME, int AGE)
{
    strcpy(name, NAME);
    age = AGE;
}
void Student::setAge(int AGE)
{
    age = AGE;
}
void Student::setName(char *NAME)
{
    strcpy(name, NAME);
}
Student::~~Student(void)
{
}
void Student::print()
{
    cout << "Name - " << Student::name << endl;
    cout << "Age - " << Student::age << endl;
}
char* Student::getName(void)
{
    return name;
}
int Student::getAge(void)
{
    return age;
}

```

Main_mod.cpp

```
#include "stdafx.h"
#include "student.h"
#include <iostream>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    Student a1, a2;
    a1.setName("Oleg");
    a1.setAge(17);
    a2.setName("Olya");
    a2.setAge(18);
    char* q = a1.getName();
    int w = a1.getAge();
    a1.print();
    a2.print();
    return 0;
}
```

Теория

Объект - структурированная переменная, содержащая всю информацию о некотором физическом предмете или реализуемом в программе понятии.

Класс - описание множества таких объектов и выполняемых над ними действий.

Классы

Классы C++ предусматривают создание расширенной системы предопределенных типов. Каждый тип класса представляет уникальное множество объектов, операций над ними, а также операций, используемых для создания, манипулирования и уничтожения таких объектов. С классами связан ряд новых понятий.

Наследование. Новый, или производный класс может быть определен на основе уже имеющегося, или базового. При этом новый класс сохраняет все свойства старого: данные объекта базового класса включаются в данные объекта производного, а методы базового класса могут быть вызваны для объекта производного класса.

Полиморфизм. Он основывается на возможности включения в данные объекта также и информации о методах их обработки (в виде указателей на функции). Будучи доступным в некоторой точке программы, даже при отсутствии полной информации о типе объекта, он всегда может корректно вызвать свойственные ему методы. *Полиморфной* называется функция, определенная в нескольких производных классах и имеющая в них общее имя.

Для определения класса используются три ключевых слова: `struct`, `union` и `class`. Каждый класс включает в себя функции - называемые методами, компонентными функциями или функциями членами (`member function`), и данные - элементы данных (`class members`)

Имя класса становится идентификатором нового типа данных и может использоваться для объявления объектов.

С точки зрения синтаксиса, класс в C++ - это структурированный тип, образованный на основе уже существующих типов. Класс можно определить с помощью формы:

```
тип_класса имя_класса <:базовый_класс>
    {список_членов_класса или список компонентов};
```

где `тип_класса` - одно из служебных слов `class`, `struct`, `union`;

`имя_класса` - идентификатор;

необязательный параметр `базовый_класс` - содержит класс или классы, из которого `имя_класса` заимствует элементы и методы, при необходимости спецификаторы доступа;

`список_членов_класса` - определения и описания типизированных данных и принадлежащих классу функций.

Пример:

```
class STUDENT    {
public:
    char name[25];           // имя
    int age;                 // возраст
    float grade;             // рейтинг
    char * getName() ;       //метод - получить имя
    int getAge() const;       //метод - получить возраст
    float getGrade() const;   //метод - получить рейтинг
    void setName(char*);      //метод - установить имя
    void setAge(int);         //метод - установить возраст
    void show(); };          //метод - печать
```

Тело элемента-функции может быть определено в самом классе или в определении класса дается только прототип функции (заголовок с перечислением типов формальных параметров), а определение самой функции дается отдельно, при этом полное имя функции имеет вид:

`имя_класса::имя_функции`

Для структурированной переменной вызов функции - элемента этой структуры имеет вид:

имя_переменной.имя_функции (список_параметров);

Для описания объекта класса (экземпляра класса) используется форма:

имя_класса имя_объекта;

Пример:

```
STUDENT my_friend, me;  
STUDENT *point = &me;           // указатель на объект STUDENT  
STUDENT group [30];             // массив объектов  
STUDENT &girl = my_friend;      // ссылка на объект
```

Обращаться к данным объекта или компонентам класса и вызывать функции для объекта можно двумя способами. Первый с помощью «квалифицированных» имен:

**имя_объекта. имя_данного
имя_объекта. имя_функции**

Пример:

```
STUDENT x1, x2;  
x1.age = 20;  
x1.grade = 2.3;  
x2.set("Петрова", 18, 25.5);  
x1.show ();
```

Второй способ доступа использует указатель на объект:

указатель_на_объект->имя_компонента

Пример:

```
STUDENT *point = &x1;  // или point = new STUDENT;  
point ->age = 22;  
point ->grade = 7.3;  
point ->Show();
```

Доступность компонентов класса

В рассмотренных примерах компоненты классов являются общедоступными. В любом месте программы, где «видно» определение класса, можно получить доступ к компонентам объекта класса. Тем самым не выполняется основной принцип абстракции данных - инкапсуляция (сокрытие) данных

внутри объекта. Для изменения видимости компонент в определении класса можно использовать спецификаторы доступа: `public`, `private`, `protected`.

Общедоступные, `public` компоненты доступны в любой части программы. Они могут использоваться любой функцией как внутри данного класса, так и вне его. Доступ извне осуществляется через имя объекта:

имя_объекта.имя_члена_класса

ссылка_на_объект.имя_члена_класса

указатель_на_объект->имя_члена_класса

Собственные, `private` компоненты локализованы в классе и не доступны извне. Они могут использоваться функциями - членами данного класса и функциями-друзьями того класса, в котором они описаны.

Защищенные, `protected` компоненты доступны внутри класса и в производных классах.

По умолчанию для `class` все элементы являются собственными или частными, т.е. `private`.

Пример:

```
class STUDENT{
    char name[25];           // private по умолчанию
    int age;
    float grade;
    public:
    void setName(char*);
    void setAge(int);};
```

Спецификатор доступа действителен пока не встретится другой спецификатор доступа, они могут пересекаться и группироваться:

```
class STUDENT{
    char name[25];
    int age;
    float grade;
    private
    protected:
    void setName(char*);
    public:
    void setAge(int);
}Me;
Me.age=18;           // ошибка
Me->setAge(18);       // правильно
```

Создание и уничтожение объектов

Недостатком рассмотренных ранее классов является отсутствие автоматической инициализации создаваемых объектов. Для каждого вновь создаваемого объекта необходимо вызвать функцию типа `Set` либо явным образом присваивать значения данным объекта.

Элементы-функции, неявно вызываемые при создании и уничтожении объектов класса называются *конструкторами* и *деструкторами*. Они определяются как элементы-функции с именами, совпадающими с именем класса. Конструкторов для данного класса может быть сколь угодно много, если они отличаются формальными параметрами, деструктор же всегда один и имеет имя, предваренное символом "~".

Конструктор

Формат определения конструктора:

имя_класса(список_форм_параметров){операторы_тела_конструктора}

```
class STUDENT{
    char name[25];
    int age;
    float grade;
public:
    STUDENT();                // конструктор без параметров
    STUDENT(char*,int,float); // конструктор с параметрами
    STUDENT(const STUDENT&);  // конструктор копирования
    ...};
    STUDENT::STUDENT(char*NAME,int AGE,float GRADE)
    {
        strcpy(name,NAME); age=AGE; grade=GRADE;
        cout<< "\nКонструктор с параметрами вызван для объекта
        <<this<<endl;
    }
```

Момент вызова конструктора и деструктора определяется временем создания и уничтожения объектов. Конструкторы обладают некоторыми уникальными свойствами:

- конструктор имеет то же имя, что и сам класс;
- для конструктора не определяется тип возвращаемого значения (даже тип `void` не допустим);
- конструкторы не наследуются, хотя производный класс может вызывать конструкторы базового класса;
- конструкторы могут иметь аргументы по умолчанию или использовать списки инициализации элементов;

- конструкторы не могут быть описаны с ключевыми словами `virtual`, `static`, `const`, `mutable`, `volatile`;
- нельзя работать с их адресами;
- если он не был задан явно, то генерируется компилятором;
- конструктор нельзя вызывать как обычную функцию;
- конструктор автоматически вызывается при определении или размещении в памяти с помощью неявного вызова оператора `new` каждого объекта класса и `delete`;
- конструктор выделяет память для объекта и инициализирует данные-члены класса.

По умолчанию создается общедоступный (`public`) конструктор без параметров и конструктор копирования. Если конструктор описан явно, то конструктор по умолчанию не создается. Параметром конструктора не может быть его собственный класс, но может быть ссылка на него (`T&`). Без явного указания программиста конструктор всегда автоматически вызывается при определении (создании) объекта. Для явного вызова конструктора используются две формы:

имя_класса имя_объекта (фактические_параметры);
имя_класса (фактические_параметры);

Первая форма допускается только при не пустом списке фактических параметров. Вторая форма вызова приводит к созданию объекта без имени. Существуют два способа инициализации данных объекта с помощью конструктора: передача значений параметров в тело конструктора; применение списка инициализаторов данного класса. Каждый инициализатор списка относится к конкретному компоненту и имеет вид:

имя_данного (выражение)

При определении объекта будет запущен тот конструктор, с которым совпадает заданный список аргументов.

Конструктор копирования (вида `T::T(const T&)`, где `T` – имя класса) вызывается всякий раз, когда выполняется копирование объектов, принадлежащих классу. Он вызывается:

а) когда объект передается функции по значению:

```
void view(STUDENT a){a.show;}
```

б) при построении временного объекта как возвращаемого значения функции:

```
STUDENT noName(STUDENT & student)
{STUDENT temp(student);
temp.setName("NoName");
return temp;}
```

```
    }  
STUDENT c=noName(a);
```

в) при использовании объекта для инициализации другого объекта:
STUDENT a("Иванов", 19, 50), b=a;

Если класс не содержит явным образом определенного конструктора копирования, то при возникновении одной из этих трех ситуаций производится побитовое копирование объекта (не во всех случаях является адекватным).

Деструктор

Динамическое выделение памяти для объекта создает необходимость освобождения этой памяти при уничтожении объекта. Такую возможность обеспечивает специальный компонент класса – *деструктор*. Его формат:
~имя_класса(){операторы_тела_деструктора}

Свойства деструктора:

- имя деструктора совпадает с именем его класса, но предваряется символом "~" (тильда);
- деструктор не имеет параметров и возвращаемого значения;
- вызов деструктора выполняется не явно (автоматически), как только объект класса уничтожается.

Если в классе деструктор не определен явно, то компилятор генерирует деструктор по умолчанию, который просто освобождает память, занятую данными объекта. Так же, как и для конструктора, не может быть определен указатель на деструктор.

Указатели на компоненты-функции

Можно определить указатель на компоненты-функции:

```
тип_возвр_значения(имя_класса::*имя_указателя_на_функцию)  
    (специф_параметров_функции);
```

Пример:

```
void (STUDENT::*pf)();  
pf=&STUDENT::show;  
(p[1].*pf)();
```

Указатель this

Когда функция-член класса вызывается для обработки данных конкретного объекта, этой функции автоматически и неявно передается указатель на тот

объект, для которого функция вызвана. Этот указатель имеет имя `this` и неявно определен в каждой функции класса следующим образом:

имя_класса *const this = адрес_объекта

Указатель `this` является дополнительным скрытым параметром каждой нестатической компонентной функции. При входе в тело принадлежащей классу функции `this` инициализируется значением адреса того объекта, для которого вызвана функция. В результате этого объект становится доступным внутри этой функции.

В большинстве случаев использование `this` является неявным. В частности, каждое обращение к нестатической функции-члену класса неявно использует `this` для доступа к члену соответствующего объекта. Примером широко распространенного явного использования `this` являются операции со связанными списками.

Вставляемые (inline) функции

Если функция (обычная или функция-элемент класса) объявлена `inline`-функциями, то при вызове таких функций транслятор выполняет подстановку по тексту программы тела функции с соответствующей заменой формальных параметров на фактические. Функция-элемент также считается `inline` по умолчанию, если ее тело находится непосредственно в определении класса, например:

```
struct dat
{int      day,month,year;
void      setDat(char *p)      // Функция inline по умолчанию
    { ... }                  // тело функции
};
```

Функцию-член можно описать как `inline` вне описания класса. Например:

```
char Char_stack {
    int size;
    char* top;
    char* s;
public:
    char pop();
    // ... };
inline char Char_stack::pop()
{    return *--top; }
```

Друзья классов

Иногда требуются исключения из правил доступа, когда некоторой функции или классу требуется разрешить доступ к личной части объекта класса. Тогда в определении класса, к объектам которого разрешается такой доступ, должно быть объявление функции или другого класса как «дружественных».

Объявление дружественной функции представляет собой прототип функции, объявление переопределяемой операции или имя класса, которым разрешается доступ, с ключевым словом `friend` впереди. Общая схема объявления такова:

```
class A
{
    int x; // Личная часть класса
    ...
friend class B; // Функции класса B дружественны A
friend void C::fun(A&);
    // Элемент-функция fun класса C имеет доступ к приватной части A
friend void xxx(A&,int); // Функция xxx дружественна классу A
friend void C::operator+(A&); // Переопределяемая в классе C операция
}; // <объект C>+<объект A> дружественна классу A
class B // Необходим доступ к личной части A
{public: int fun1(A&);
void fun2(A&); };
class C
{ public: void fun(A&);
void operator+(A&); };
```

К средствам контроля доступа относятся также объявления функций-элементов постоянными (`const`). В этом случае они не имеют права изменять значение текущего объекта, с которым вызываются. Заголовок такой функции имеет вид:

```
void Dat::put() const { ... }
```

По аналогии с функциями и методами можно объявить дружественными весь класс. При этом все методы такого дружественного класса смогут обращаться ко всем компонентам класса:

```
class MyClass
{... friend class MyFriend; ...}
```

Дружба классов не транзитивна и односторонняя.

Статические элементы класса

Иногда требуется определить данные, которые относятся ко всем объектам класса. Например, контроль общего количества объектов класса или одновременный доступ ко всем объектам или части их, разделение объектами общих ресурсов. Тогда в определение класса могут быть введены статические элементы-переменные. Такой элемент сам в объекты класса не входит, зато при обращении к нему формируется обращение к общей статической переменной с именем. Доступность ее определяется стандартным образом в зависимости от размещения в личной или общей

части класса. Сама переменная должна быть явно определена в программе модификатором `static` и инициализирована.

Статическими могут быть объявлены методы класса. Их вызов не связан с конкретным объектом и может быть выполнен по полному имени. Соответственно в них не используются неявный указатель на текущий объект `this`. Статические методы не могут быть константными (`const`) и виртуальными (`virtual`). Например:

```
class List
{ ...
static void show();           // статическая функция просмотра
                               // списка объектов
static void List::show()
{List *p;
for (p=fst; p !=NULL; p=p->next)
    { ...вывод информации об объекте... }}
void main()
{ ... List::show(); }         // вызов функции по полному имени
```

Приложение . Стандарт оформления кода

При выполнении задания ознакомьтесь с правилами написания кода (Code convention). Указанные здесь требования не являются единственно верными для написания кода. Но практически всегда в любой команде разработки существует свой стиль, которого придерживаются все члены группы.

Указанные здесь требования являются производными от широко распространённого стандарта, с которым вам стоит ознакомиться в оригинале:

Стандарт кодирования GNU http://www.opennet.ru/docs/RUS/coding_standard/

Google C++ Style Guide <https://google-styleguide.googlecode.com/svn/trunk/cppguide.html>

C++ Programming Style Guidelines <http://geosoft.no/development/cppstyle.html>

а также почитать на эту тему:

Стандарт оформления кода <https://ru.wikipedia.org/wiki>

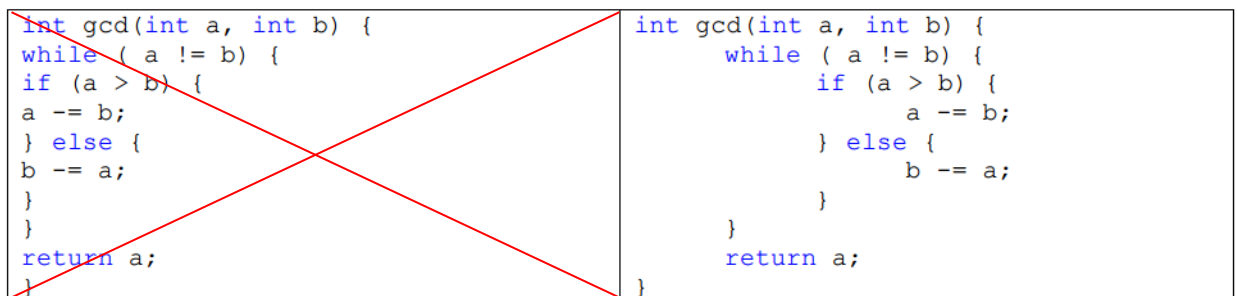
<http://habrahabr.ru/post/194752/> и др.

Стиль программирования — это набор правил и соглашений, используемых при написании исходного кода на некотором языке программирования. Наличие общего стиля программирования облегчает понимание и поддержание исходного кода, написанного больше, чем одним программистом, а также облегчает сотрудничество нескольких человек в развитии одного программного обеспечения. Следование правилам хорошего стиля программирования значительно уменьшает вероятность появления ошибок на этапе набора текста, делает программу легко читаемой, что, в свою очередь, облегчает процессы отладки и внесения изменений.

Отступы

Для всего проекта должен применяться единый стиль отступов. Всегда будьте последовательны в использовании отступов и применяйте их для повышения читабельности кода.

Отступы ставятся везде, где есть фигурные скобки. То есть в теле функций, циклах (do, while, for), операторах if и switch.



Фигурные скобки

Всегда используйте фигурные скобки, даже если их можно опустить. Сегодня имеется четыре наиболее распространенных стиля расстановки фигурных скобок, используемых в C-сообществе. Можно использовать любой из них

1. Стилль K&R или 1TBS (One True Bracing Style) Назван в честь Brian W. Kernighan и Dennis M. Ritchie.

```
if (condition) {
    content;
}
```

2. Стилль Алмена Используется по умолчанию в Microsoft Visual Studio (и более ранних продуктах).

```
if (condition)
{
    content;
}
```

3. Стилль GNU Используется во всех проектах GNU. Отступы всегда 4 символа на уровень, скобки находятся на половине отступа.

```
if (condition)
{
    content;
}
```

Каждый блок примитива управления потоком («if», «else», «while», «for», «do», «switch») должен заключаться в скобки, даже если он содержит только одну строку, или не содержит ничего вообще.

Сокращённая запись часто приводит к ошибкам, которые тяжело вычислить, т.к. если такой блок кода потребуется расширить, то наличие скобок уже становится обязательным, о чём не всегда вспоминает программист, дописавший код.

В булевских выражениях («if», «for», «while», «do» и первом операнде тернарного оператора "?") всегда записывайте равенство и неравенство в явном виде.

Избегайте условных выражений, которые всегда имеют один и тот же результат (за исключением платформозависимого кода, когда результат на другой платформе может все-таки быть иным).

Длина строки

Длина строки не должна превышать 80 символов. Выражения длиннее 80 символов разделяются на части, при этом последующие части должны быть короче первой и сдвинуты вправо (чтобы схожие логические объекты находились на одной вертикально прямой). Те же правила применяются к функциям с длинным списком аргументов и текстовым строкам.

```
void fun(int a, int b, int c) {
    if (condition) {
        printf("Warning this is a long "
               "printf with 3 parameters a: %u b: %u "
               "c: %u \n", a, b, c);
    } else {
        next_statement;
    }
}
```

Когда Вы пишете конструкцию if-else, которая вложена в другую конструкцию if, всегда следует помещать скобки вокруг if-else.

Комментарии

Комментарии нужны для облегчения чтения кода, но при неправильном использовании этого средства можно только усложнить читабельность кода. Например, комментируя те моменты, которые и так всем понятны. Не пытайтесь в комментариях объяснить, как работает программа, код должен сам говорить за себя. Т.е. он должен быть как можно более понятным и прозрачным. Комментарии должны пояснять, что делает программа, а не как она это делает. Имена переменных и функций Существует несколько стилей названия переменных

Сокращения

Стоит пользоваться упрощёнными конструкциями:

Инкремент

Конструкция вида: b[j] = <выражение>; j++;	Может быть упрощена до: b[j++] = <выражение>;
--	--

лишняя переменная

Конструкция вида: A a = <выражение>; return a;	Может быть упрощена до: return <выражение>;
--	--

лишняя else-ветка

Конструкция вида: if (<условие>) { return true; } else { return false; }	Может быть упрощена до: return <условие>;
---	--

лишнее использование условного оператора

Конструкция вида: if (<условие>) { return true; } return false;	Может быть упрощена до: return <условие>;
---	--

формирование переменной внутри условного оператора

Конструкция вида: A a; if (<условие>) {	Может быть упрощена до: if (<условие>) { return <выражение1>;
---	---

<pre> a = <выражение1>; } else { a = <выражение2>; } return a; </pre>	<pre> } return <выражение2>; </pre>
---	---

Имена переменных, типов и функций

Существует несколько стилей названия переменных

var_bell – стиль C: нижний регистр, знак подчёркивания.

VarBell – стиль Pascal: каждая подстрока в названии начинается с большой буквы.

varBell – стиль Java: первая строка начинается с маленькой буквы, все последующие с большой.

Не имеет значения, какой стиль будет вами выбран — главное придерживаться в коде одного стиля

Константы традиционно записываются в верхнем регистре (например, **YANDEX_BOT**).

При выборе имени переменной не так важна длина имени, как понятность. Длинные имена могут назначаться глобальным переменным, которые редко используются, но индексы массивов, появляющиеся в каждой строке цикла, не должны быть значительно сложнее, чем **i**. Использование **index** или **elementnumber** не только усложняет набор, но и может сделать менее понятными детали вычислений.

Примеры плохих и хороших (понятных) имен.

<pre> num1 do_this() g() hxq </pre>	<pre> numberOfCharacters number_of_chars num_chars get_number_of_chars() is_char_limit() character_max() charMax() i (loop value) </pre>
-------------------------------------	--

Называйте переменные, функции и файлы со строчной буквы, а свои типы (в т.ч. классы) с заглавной.

```

struct ListNode
{
    int value;
    ListNode* next;
};

int getNextValue();

void printoutList();

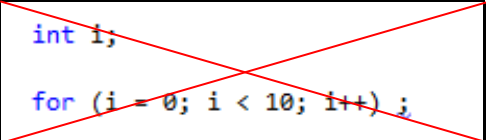
```

Приватные переменные к классу используют в качестве суффикса нижнее подчеркивание:

```
class SomeClass
{
    private:
    int length_;
}
```

Числа

Не используйте "магические константы" (опять же, не следует увлекаться).

 <pre>int i; for (i = 0; i < 10; i++) ;</pre>	<pre>int const dimension = 10; for (i = 0; i < dimension; i++)</pre>
---	---

Функции

Функции-предикаты (булевы функции) следует именовать следующим образом:

[глагол][предикат]

где "глагол" - глагол бытия (to be) или обладания (to have) в соответствующей форме, а "предикат" - проверяемое свойство. Например:

<pre>bool isPrime(int n) ; bool hasColor();</pre>

При вызове функций-предикатов запрещается сравнивать результат с логическими константами.

if (isPrime(n) == true) –плохой код.

Сравнивать логические переменные с логическими константами столь же бессмысленно.

Не используйте транслит, пишите все либо по-русски в читабельной в компьютерном классе кодировке, либо по-английски. Например, `vstavka()` - неудачное название для функции, реализующей алгоритм сортировки вставками.

Все методы, которые не изменяют видимое извне состояние объекта, должны быть определены константными.

<pre>string getName () const { return name; }</pre>

Вывод

Каждый вывод данных на экран должен сопровождаться сообщением с пояснением того, что именно выводится.

Классы

Классы следует называть с большой буквы, слова в названии не должны иметь разделителей и должны состоять только из латинских букв и, при необходимости, цифр. Если название содержит несколько слов, то каждое следующее слово должно начинаться с большой буквы.

```
class PrimeNumbers {///  
};
```

Члены класса следует определять в порядке, соответствующем уровню доступа: сначала «public», потом «protected», последними «private».

```
class Student  
{  
public:  
  
    Student (string a, int b, int c) { ... }  
    Student () { ... }  
protected: int number;  
  
private: string name;  
         int score;  
         int scriptedlesson;  
};
```

Экземпляры класса стоит определять напрямую конструктором, а не копирующим конструктором через присвоение.

Вопросы

1. Какие ключевые слова можно использовать при определении класса?
2. В чем отличие между объектом и классом?
3. Что такое конструктор?
4. Что такое деструктор (destructor) ?
5. Что такое дружественная функция (friend function)?
6. Что такое копирующий конструктор (copy constructor)?
7. Продолжите фразу «... является специальной функцией-членом, используемой для задания начальных значений данным, членам класса».
8. Продолжите фразу «по умолчанию доступ к членам класса ...».
9. Продолжите фразу «говорят, что реализация класса скрыта от его клиентов или ...».
10. Продолжите фразу «набор открытых функций-членов класса рассматривается как ... класса».
11. Продолжите фразу «в определении класса члены класса с ключевым словом `private` доступны ...».
12. Напишите определение класса `SS`, включающего одно закрытое поле типа `int` с именем `bar` и одним открытым методом с прототипом `void Print()`.
13. Истинно ли следующее утверждение: поля класса должны быть закрытыми.
14. Продолжите фразу «операция точки (операция доступа к члену класса) объединяет следующие два элемента (слева направо)».
15. Конструктор вызывается автоматически в момент ... объекта.
16. Верно или неверно следующее утверждение: класс может иметь более одного конструктора с одним и тем же именем.
17. Найдите ошибку и объясните, как ее исправить. Допустим, что в классе `Time` объявлен следующий прототип:

```
void ~Time( int )
```
18. Функция, не являющаяся членом, которая должна иметь доступ к закрытым данным-членам класса, должна быть объявлена как ... этого класса.
19. Определите класс `СВОХ`, объявите массив объектов `СВОХ`.
20. Определите класс `СВОХ`, объявите статический член класса `int`. Выполните его инициализацию. Сколько экземпляров статического данного будет существовать, если число объектов класса равно 5.
21. Пусть есть класс.

```
class List  
{ // ...
```

```

static void show();
// Статическая функция просмотра списка объектов
}
static void List::show()
{
List *p;
for (p=fst; p !=NULL; p=p->next)
    // { ...вывод информации об объекте... }
}

```

Создайте объект класса и вызовите функцию show.

22. Пусть есть класс DATE. Запишите возможные способы создания объектов класса. Когда будет вызываться конструктор копирования?

23. Что такое вложенные классы? Приведите пример.

24. В чем разница между X x; и X x(); ?

25. Что будет выведено на экран?

```

#include <iostream>
class A
{
public:
    A(void){this->_num=0;}
    int A(int num){this->_num=num;}
    ~A(void){std::cout << this->_num;}
private:
    int _num;
};
int main(void)
{
    A val(100);
    return 0;
}

```

26. Скомпилируется ли следующий код:

```

class Clazz
{
};

```

27. Каким будет результат выполнения следующего кода:

```

class Counter {
public:
    void count() { printf("%d", 1); }
};

int main() {
    Counter obj;
    obj.count();
    return 0;
}

```

```
}
```

```
void Counter::count() { printf("%d", 2); }
```