# Лабораторная работа № 2

**Тема**: Классы. Данные-члены функции-члены. Оператор расширения области видения. Спецификации public, private, protected. Конструкторы и деструкторы: общие положения. Параметризованные конструкторы. Конструкторы и инициализации класса.

**Цель**: Осознать понятие "Класс", "данные-члены" и "функции-члены". Научиться ими пользоваться. Изучить назначение конструкторов и деструкторов и научиться ими пользоваться

# теоретические сведения

**1 Объявление класса**

Классы С ++ предусматривают создание расширений системы предопределенных типов. Каждый тип класса составляет множество объектов и операций (правил), а также преобразований, используемых для образования, манипулирования и уничтожение таких объектов. Могут быть объявлены производные классы, наследующие компоненты одного или более базовых классов (классов, порождающих). В С++ структуры и объединения рассматриваются как классы с определенными умолчанию правил доступа. Упрощенный, в "первом приближении", синтаксис объявления класса следующий:

ключ\_класса имя\_класса

<: Базовый\_список> [<список\_компонентов>]

Ключ\_класу - это class, struct или union.

Дополнительный базовий\_список перечисляет базовый класс или классы, из которых имя\_класса берет (или наследует) объекты и правила. Если объявлены некоторые базовые классы, то класс "имя\_класса" называется производным классом. Базовый список содержит спецификаторы доступа по умолчанию и необязательно их переопределения, которые могут модифицировать права доступа производного класса к компонентам базовых классов.

Дополнительный список\_компонентов объявляет компоненты класса (данные и функции) для имени класса по умолчанию и переопределения спецификаторов доступа, которые могут влиять на то, какие функции в каких компонентов класса могут иметь доступ.

**1.1 Имена классов**

имя\_класса - это любой идентификатор, уникальный в пределах своего контекста. В классах структур и в объединениях имя\_класса может быть опущено.

**1.2 Типы классов**

Объявления создает уникальный тип, тип класса "имя\_класса". Это позволяет объявлять следующие объекты класса (или вхождения класса) предоставленного типа, а также объекты, являющиеся производными от этого типа (например, указатели, посылки, массивы объектов "iмья\_класу" и т. Д. ):

class X {....};

X x, & xr, \* xptr, xarray [10]; / \* четыре объекта: типа X, посылка на X, указатель на X и массив элементов типа X \* /

struct Y {....};

Y y, & yr, \* yptr, yarray [10];

// С позволяет иметь

// Struct Y y, & yr, \* yptr, yarray [10];

union Z {....};

Z z, & zr, \* zptr, zarray [10];

// С позволяет иметь

// Union Z z, & zr, \* zptr, zarray [10];

Определим разницу между объявлением структур и объединений в С и С ++: в С ключевые слова struct и union обязательны, но в С ++ они нужны только в том случае, когда имена классов Y и Z скрытые (см. Следующий раздел).

**1.3 Контекст имени класса**

Контекст имени класса является локальным, с некоторыми особенностями, характерными для классов. Контекст имени класса начинается с точки его объявления и заканчивается вместе с объемным блоком. Имя класса скрывает любой класс, объект, нумератор или функцию с тем же именем в объемном контексте. Если имя класса объявлен в контексте, содержащий объявления объекта, функции или нумератора с тем же именем, обращение к классу возможно только с помощью уточненного спецификатора типа. Это означает, что с именем класса надо использовать ключ класса, class, struct или union. К примеру:

struct S {....};

int S (struct S \* Sptr)

void func (void)

 {

  S t; // Недопустимо объявления: нет ключа класса и функции S в контексте

  struct S s; // Так можно: есть уточнения ключом класса

  S (& s); // Так нельзя: это вызов функции

}

С ++ позволяет также неполное объявление класса:

  class X; // Ещё нет компонентов!

  struct Y;

  union Z;

Неполные объявления позволяют некоторые ссылки к именам классов X, Y или Z (обычно ссылка на указатели объектов классов) до того, как классы будут полностью определены. Разумеется, прежде чем вы сможете объявить и использовать объекты классов, вы должны выполнить полное объявление классов со всеми их компонентами.

**2 Объекты классов**

Объекты классов могут быть присвоены (если не было запрещено копирования), переданные в качестве аргументов функции, возвращены функцией (за некоторыми исключениями) и т. Д. Другие операции с объектами и компонентами классов могут быть различными способами определенные пользователем, включая функции -компоненты и "друзей", а также переопределение стандартных функций и операций при работе с объектами конкретного класса. Переопределение функций и операций называется перегрузкой. Операции и функции, ограниченные объектами конкретного класса (или взаимосвязанной группы классов) называются функциями-компонентами данного класса. С ++ имеет механизм, позволяющий вызвать то же самое имя функции или операции для выполнения другой задачи, в зависимости от типа или числа аргументов или операндов.

**3 Список компонентов класса**

Дополнительный список компонентов составляет последовательность объявлений данных (любого типа, включая нумераторы, битовые поля и другие классы) и объявлений и определений функций, каждое из которых может иметь необязательные спецификаторы класса памяти и модификаторы доступа. Определенные таким образом объекты называются компонентами класса. Спецификаторы класса памяти auto, extern и register в данном случае недопустимы. Компоненты могут быть объявлены со спецификаторами класса памяти static.

**3.1 Функции-компоненты**

Функция, которая объявлена ​​без спецификатора friend, называется функцией-компонентом класса. Функция, объявленная с модификатором friend, называется "функцией-другом".

Одно и то же имя может использоваться для обозначения более чем одной функции, при условии, что они отличные по типам или числу аргументов.

**3.2 Ключевое слово this**

Нестатические функции-компоненты работают с объектами типа класса, с которыми они вызываются. Например, если x - это объект класса X, а f - это функция-компонент X, то вызов функции x.f () работает с x. Аналогичным образом, если xptr есть указатель объекта X, то вызов функции xptr -> f () работает с \* xptr. Откуда f может знать, с каким X работать? С ++ передает f указатель на X, называется this. This передается как скрытый аргумент при вызове нестатических функций-компонентов. Ключевое слово this составляет локальную переменную, доступную в теле нестатические функции-компонента. this не нуждается явлений, и на него редко встречаются явные посылки в определении функции. Однако, оно неявно используется в функции для посылки компонентов. Если, например, вызывается x.f (y), где y является компонентом X, то this устанавливается на & x, а y устанавливается на this-> y, что эквивалентно x.y.

**3.3 Функции встраиваемые (inline)**

Функция-компонента может быть объявлена ​​внутри своего класса, но определенная где в другом месте. И наоборот, функция-компонента может быть и объявлена, и определена в своем классе, и тогда она называется функцией, встраивается. В некоторых случаях C ++ может уменьшить затраты времени на вызов функции, подставляя вместо вызова функции непосредственно зкомпильований код тела функции. Этот процесс, называемый встроенным расширением тела функции, не влияет на контекст имени функции или ее аргументов. Встроенное распространения не всегда полезное и желаемое. Спецификатор inline представляет собой запрос (или требование) компилятор, в котором вы сообщаете, что встроенные распространение желательны. Как и в случае спецификатора класса памяти register, компилятор может либо удовлетворить, или проигнорировать ваше пожелание.

Явные или неявные запросы inline лучше всего резервировать для функций небольших по объему и функций, часто вызываются, как функции типа operator, реализующих перегружены операторы. Например, следующее объявление класса:

int i; // Global int

class X {

     char \* i;

    public:

     char \* func (void) {return i;} // inline по умолчанию char \* i;

 };

 эквивалентно

inline char \* X :: func (void) {return i;}

func определяется "вне" классом с явным спецификатором inline. Переменная i, которую возвращает func () является char \* i класса X.

**3.4 Статические компоненты**

Спецификатор класса памяти static может быть использован в объявлениях компонентов данных и функций-компонентов класса. Такие компоненты называются статическими и имеют свойства, отличные от свойств нестатических компонентов. В случае нестатических компонентов для каждого объекта класса "существует" отдельная копия; в случае же статических компонентов существует только одна копия, а доступ к ней выполняется без посылки на какой-либо конкретный объект класса. Если х - это статический компонент класса Х, то к нему можно обратиться как Х :: х (даже если объекты класса Х еще не созданы). Однако, можно выполнить доступ к х и с помощью обычных операций доступа к компонентам. Например, в виде y.x и yptr-> x, где y - это объект класса X, а yptr - это указатель объекта класса X, хотя выражения y и yptr еще не вычислены. В частности, статическая функция-компонент может быть вызвана как с использованием специального синтаксиса вызова функций-компонентов, так и без него.

class X {

int member\_int;

public:

static void func (int i, X \* ptr)

};

void g (void);

{

X obj;

func (1, & obj) // Ошибка, если где-нибудь еще не определена глобальная func ()

X :: func (1, & obj) // Вызов static func () в X допустим только для статических функций

obj.func (1, & obj) // то же самое (допустим как для статических, так и нестатических функций)

}

Поскольку статическая функция-компонент может вызываться без учета какого-либо конкретного объекта, она не имеет указателя this. Следствие по этому такой, что статическая функция-компонент не имеет доступа к нестатических компонентов без явного задания объекта с помощью "." Или "->". Например, на основе объявлений, сделанных в предыдущем примере, func () может быть определена следующим образом:

void X :: func (int i, X \* ptr)

{

member\_int = i; // На объект посилается member\_int? Ошибка!

ptr-> member\_int = i; // Так можно: теперь мы знаем!

}

Без учета встроенных функций, статические функции-компоненты глобальных классов имеют внешний тип компоновки. Статические функции-компоненты не могут быть виртуальными функциями. Недопустимо иметь статическую и нестатические функции-компоненты с одинаковыми именами и типами аргументов.

Объявления статического компонента данных в объявлении класса не является определением, так где-нибудь еще должно быть определение, отвечающее за распределение памяти и инициализацию. Определение статического компонента данных может быть опущена, если действует средство "инициализации нулями по умолчанию".

Статические компоненты класса, который объявлен локальным по отношению к некоторой функции, не имеют типа компоновки и не могут быть инициализированы. Статические компоненты глобального класса могут быть инициализированы подобно обычным глобальным объектам, но только в контексте файла. Статические компоненты подлежат обычным правилам доступа к компонентам класса, за исключением того, что они могут быть инициализированы.

class X {

...

static int x;

...

};

int X :: x = 1;

Главное использование статических компонентов состоит в том, чтобы отслеживать данные, общие для всех объектов класса, например, число созданных объектов, или ресурс, который использовался последним из пула, разделяемой всеми подобными объектами. Статические компоненты используются также для:

- Уменьшение числа видимых глобальных имен

- Того, чтобы сделать очевидным, какие именно статические объекты какому классу относятся

- Разрешения управления доступом к их имен.

**4 Контекст компонента**

Выражение X :: func () в примере, приведенном в разделе "Функции, встраиваемые", использует имя класса X с модификатором контекста доступа означает, что func (), хотя и определена "вне" классом, в действительности является функцией-компонентом Х и существует в контексте Х. Влияние Х :: распространяется на тело определения этой функции. Это объясняет, почему значения, возвращает функция, относится к X :: i, char \* i с Х, а не к глобальной переменной int i. Без модификатора Х :: функция func представляла бы обычную функцию, которая не относится к классу и возвращает глобальную переменную int i.

Итак, все функции-компоненты находятся в контексте своего класса, даже если они определены вне этого класса.

К компонентам данных класса Х можно обращаться за помощью операций выбора и -> (как и в структурах С). Функции-компоненты можно вызвать также с помощью операций выбора. К примеру,

class X {

public:

int i;

char name [20];

X \* ptr1;

X \* ptr2;

  void Xfunc (char \* data, X \* left, X \* right); // определение находится не здесь

};

void f (void)

{

X x1, x2, \* xptr = & x1;

x1.i = 0

x2.i = x1.i;

xptr-> i = 1;

x1.Xfunc ( "stan", & x2, xptr)

}

Если m является компонентом или базовым компонентом класса Х, то выражение Х :: m называется квалифицированным именем; оно имеет тот же тип, что и m, и является выражением, именует только в том случае, если выражением, именует является m. Ключевой момент здесь заключается в том, что даже если имя класса Х скрыто другим именем, квалифицированное имя Х :: m обеспечит доступ к нужному имени класса, m.

Компоненты класса не могут добавляться к нему в другом разделе вашей программы. Класс Х не может содержать объекты класса Х, но может содержать указатели или посылки на объекты класса Х (отметим аналогию с типами структур и объединений С).

**1 Основные свойства конструкторов и деструкторов**

Существует несколько специальных функций-элементов, которые определят, каким образом объекты класса создаются, инициализируются, копируются и разрушаются. Конструкторы и деструкторы являются наиболее важными из них. Они обладают большинством характеристик обычных функций-элементов: вы должны объявить и определить их в пределах класса или объявить их в классе, а определить вне его. Однако, они обладают и некоторыми уникальными свойствами.

1. Они не имеют объявлений значений возврата (даже void).
2. Они не могут быть унаследованы, хотя производный класс может вызвать конструкторы и деструкторы базового класса.
3. Конструкторы, как и большинство функций языка С ++, могут иметь аргументы по умолчанию или использовать списки инициализации элементов.
4. Деструкторы могут иметь атрибут virtual, но конструкторы не могут.
5. Вы не можете работать с их адресами:

main ()

 {

  ....

  void \* ptr = base :: base; // недопустимо

  ....

 }

1. Если конструкторы и деструкторы ни были заданы явно, то они могут генерироваться C ++. Часто они также могут запускаться при отсутствии явного вызова в вашей программе. Любой конструктор или деструктор, что создается компилятором, должен иметь атрибут public.
2. Вызвать конструктор таким же образом, как и обычную функцию, нельзя. Вызов деструктора допустим только с полностью уточненным именем.

{...

X \* p;

....

p-> x :: ~ x (); // Допустимый вызов деструктора

X :: X (); // Недопустим вызов конструктора

....

}

1. При определении и уничтожении объектов компилятор выполняет вызов конструкторов и деструкторов автоматически.
2. Конструкторы и деструкторы при необходимости распределения памяти объекту могут выполнять неявные вызовы операций new и delete.
3. Объект с конструктором или деструктором не может использоваться в виде элемента объединения.

Если класс Х имеет один или более конструкторов, то один из них запускается каждый раз при определении объекта х класса Х. Конструктор создает объект х и инициализирует его. Деструкторы выполняют обратный процесс, разрушая объекты класса, созданные конструкторами.

Конструкторы активизируются также при создании локальных или временных объектов данного класса. Деструкторы активизируются каждый раз, когда эти объекты выходят из области действия.

**2 Конструкторы**

Конструкторы отличаются от других функций-элементов тем, что имеют то же имя, что и класс, к которому они относятся. При создании или копировании объекта данного класса происходит неявный вызов соответствующего конструктора.

Конструкторы глобальных объектов вызываются до вызова функции main (). При использовании стартовой директивы pragma для установления некоторой функции к функции main () конструкторы глобальных объектов вызываются в функции начальной загрузки.

Локальные объекты создаются, как только становится активной область действия переменной. Конструктор также запускается при создании временного объекта данного класса.

  class X {

    public:

       X (); // Конструктор класса Х

   };

Конструктор класса Х не может воспринимать Х в качестве аргумента:

  class X {

    ....

    public

       X (X) // недопустимо

   }

Параметры конструктора могут быть любого типа, за исключением класса, элементом которого является данный конструктор. Конструктор может принимать в виде параметра посылку на свой собственный класс. В таком случае он называется конструктором копирования. Конструктор, не имеет параметров вообще, называется конструктором, используемый по умолчанию.

**3 Конструктор, используемый по умолчанию**

Конструктором, используемый по умолчанию для класса Х называется такой конструктор, не имеет никаких аргументов: Х :: Х (). Если для класса не существует конструкторов, определяемых пользователем, то C ++ генерирует конструктор по умолчанию. При таких объявлениях, как Х х, конструктор по умолчанию создает объект х.

Как и все функции, конструкторы могут иметь аргументы, используемые по умолчанию. Например, конструктор:

X :: X (int, int = 0);

может иметь один или два аргумента. Если данный конструктор будет представлен только с одним аргументом, второй аргумент, которого не хватает, будет принят как int 0 Аналогичным образом, конструктор:

X :: X (int = 5, int = 6)

может иметь два аргумента, один аргумент, или не иметь аргументов вообще, причем в каждом случае используются соответствующие умолчания. Однако, конструктор по умолчанию Х :: Х () не имеет аргументов вообще, и его не надо путать с конструктором, например, X :: X (int = 0), что может или иметь один аргумент, или не иметь аргументов.

При вызове конструкторов надо избегать неоднозначностей. В следующем примере возможно неоднозначное восприятие компилятором конструктора, используется по умолчанию и конструктора, воспринимает целочисленный параметр:

class X {

public:

X ();

X (int i = 0);

};

main ()

{

X one (10); // Допустим: используется X :: X (int)

X two; ....

 return 0;

 }

**4 Конструктор копирования**

Конструктор копирования для класса Х - это такой конструктор, который может вызываться с одним-единственным аргументом типа X: X :: X (const X &) или X: :( const X &, int = 0). В конструкторе копирования допустимыми также аргументы по умолчанию. Конструкторы копирования запускаются при копировании объекта данного класса, обычно в случае объявления с инициализацией объектом другого класса:

 X x = y;

 Если такой конструктор необходим, но в классе Х не определен, C ++ генерирует конструктор копирования для класса X автоматически.

**5 Переопределение конструкторов**

Конструкторы можно переопределить, что позволяет создавать объекты в зависимости от значений, которые использовались при инициализации.

class X {

     int integer\_part;

     double double\_part;

  public:

     X (int i) {integer\_part = i; }

     X (double d) {double\_part = d}

 };

main ()

 {

  X one (10); // Вызывает X :: X (int) и устанавливает integer\_part в значение 10

     X one (3 14); // Вызывает X :: X (double) и устанавливает double\_part в значение 3. 14

  ....

  return 0;

 }

При наличии конструктора класса объекты или инициализируются, или имеют конструктор по умолчанию. Последний используется в случае объектов без явной инициализации.

Объекты классов с конструкторами могут инициализироваться с помощью списков задаются в круглых скобках инициализаторов. Этот список используется как список аргументов, передаваемых конструктору. Альтернативным средством инициализации является использование знака равенства, за которым следует отдельное значение. Это отдельное значение может иметь тип первого аргумента, воспринимается конструктором данного класса. В этом случае дополнительных аргументов или не существует, или они имеют значения по умолчанию. Значение может также быть объектом данного типа класса. В первом случае для создания объекта вызывается соответствующий конструктор. В последнем случае для инициализации объекта вызывается конструктор копирования.

class X

 {

     int i;

  public:

     X (); // Тела функций для ясности опущены

     X (int x)

     X (const X &)

 };

main ()

 {

    X one; // Запуск конструктора по умолчанию

    X two (1); // Используется конструктор X :: X (int)

    X three = 1; // Вызывает X :: X (int)

    X four = one; // Запускает X :: X (const X &) для копирования

    X five (two) // Вызывает X :: X (cont X &)

    ....

 }

Конструктор может присваивать значение своим элементам следующим образом: он может принимать значения в качестве параметров и выполнять присвоение элементам переменным собственно в теле функции конструктора:

class X

 {

    int a, b;

  public:

     X (int i, int j) {a = i; b = j}

 };

**6 Деструкторы**

Деструктор класса вызывается для освобождения элементов объекта к уничтожению самого объекта. Деструктор - это функция-элемент, имя которой совпадает с именем класса, перед которым стоит символ тильды (~). Деструктор не может принимать никаких параметров, а также не объявляет типа или возвращаемого значения.

class X

 {

  public:

     ~ X (); // Деструктор класса X

 };

Если деструктор не объявлен для класса явно, компилятор генерирует его автоматически.

**7 Вызов деструкторов**

Вызов деструктора выполняется неявно, когда переменная выходит из своей объявленной области действия. Для локальных переменных деструкторы вызываются, когда перестает быть активным блок, в котором они объявлены. В случае глобальных переменных деструкторы вызываются как часть процедуры выхода после функции main ().

Когда указатели объектов выходят за пределы области действия, неявный вызов деструктора не происходит. Это означает, что для уничтожения такого объекта операция delete должна задаваться явно.

Деструкторы вызываются строго в обратной последовательности по последовательности вызова соответствующих им конструкторов.

**8 atexit, #pragma exit и деструкторы**

Все глобальные объекты остаются активными до тех пор, пока не будут выполнены коды во всех процедурах выхода. Локальные переменные, включая те, которые объявлены в функции main (), уничтожаются при их выходе из области действия. Последовательность выполнения в конце программы Borland C ++ в этом смысле такая:

• выполняются функции atexit в последовательности их вставки в программу;

• выполняются функции #pragma exit соответствии с кодами их приоритетов;

• вызываются деструкторы глобальных переменных.

**9 exit () и деструкторы**

При вызове exit () из программы деструкторы для любых локальных переменных в текущей области действия не вызываются. Глобальные переменные уничтожаются в обычной последовательности.

**10 abort () и деструкторы**

При вызове abort () где по программе деструкторы не вызываются, даже для переменных глобальной области действия.

Деструктор может также вызываться явно, одним из двух способов:

• косвенно, через вызов delete;

• непосредственно, путем задания полностью уточненного имени деструктора.

delete можно использовать для уничтожения тех объектов, для которых память распределялась с помощью new. Явный вызов деструктора необходим только в случае объектов, которым с помощью вызова new распределялась конкретный адрес памяти.

class X {

  ....

  ~ X ();

  ....

 };

void \* operator new (size\_t size, void \* ptr)

 {

  return ptr;

 }

char buffer [sizeof (x)];

main ()

 {

  X \* pointer = new X;

  X \* exact\_pointer;

  exect\_pointer = new (& buffer) X; // Указатель инициализируется адресом буфера

  ....

  delete pointer; // Delete служит для разрушения указателя

  exact\_pointer-> X :: ~ X (); } // Прямой вызов для отмены распределения памяти

Контрольные вопросы к Л.Р. 2:

1. Что называется классом?

2. Какие спецификаторы доступа вы знаете?

3. Какую роль играет спецификатор доступа?

4. нужно создать для того, чтобы использовать класс?

5. Что представляет собой объект класса?

6. С помощью каких операций можно обращаться к компонентам данных и функций класса?

7. Какие функции класса называют встроенными?

8. Для чего нужны неполные объявления класса?

9. Какие действия можно выполнять над объектами классов?

10. Расскажите о ключевое слово this

11. Какие компоненты называют статическими?

12. Что представляет собой конструктор?

13. Что представляет собой деструктор?

14. Перечислите уникальные свойства конструкторов и деструкторов

15. Сколько конструкторов можно сказать в любом классе?

16. Сколько деструкторов можно сказать в любом классе?

17. Как вызываются конструкторы и деструкторы?

18. Когда вызываются конструкторы глобальных объектов класса и это обеспечивает?

19. Когда вызываются деструкторы глобальных объектов класса?

20. Как вызываются конструкторы и деструкторы для автоматических объектов класса объявленных локальными в функции?

21. Какой конструктор класса вызывается для каждого объекта массива при создании массивов объектов класса?

22. Какие механизмы безопасного копирования и присвоение объектов класса, имеют члены-указатели?

Задание:

1. Определить классы Triangle и Circle и найти площади указанных фигур (треугольника и круга).

2. Определить класс треугольника где функция-член класса определяет меньшую сторону треугольника (по формуле определенния расстоянии между двумя точками на координатной плоскости).

3. С помощью класса Ttime напишите DT.Cpp, которая выводит текущие дату и время и заменяет дату в вашем объекте на завтрашнюю.

4. Разработайте класс кнопки, который можно использовать в моделировании, где необходимы переключатели включен / выключен (данные-члены класса) и функции-члены класса определяющих состояние включенной кнопки и выключенной кнопки.

5. Данная программа:

    #include <иоstream.h>

    main ()

   {

         cout << "Привет! \ N ";

return 0;

    }

    Модифицируйте ее так, чтобы она выводила на выходе:

              инициализация

              Здравствуйте!

              очистка

   Не изменяйте main () !!!.

6. Создайте класс, способный хранить строку в купе. Должна существовать возможность передачи строки объекта вашего класса и затем получения указателя на такою же строку. Должно быть также возможность изменения строки объекта класса. Используйте конструкторы и деструктор для введения всех автоматических инициализаций и очистки объекта.

7. Определите класс histogram, который подсчитывает числа в определенных интервалах, описанных в качестве аргументов конструктора histogram. Напишите функции для вывода гистограммы. Предусмотрите обработку значений, выходящих за пределы определенной области.

8. Есть класс с именем Tfruit и объект orange типа Tfruit. Используйте orange для инициализации нового объекта на имя grapefruit с помощью конструктора копирования.

Задания для самостоятельной работы:

1. Определить функцию, рисует линию, которая соединяет 2 фигуры (круги), отыскивая 2 ближайшие точки соприкосновения.

2.Створиты класс для работы с десятичными числами. Данные члены: цели, десятичные; данные-функции: установление целых и десятичных, вывода целых и десятичных, числа.

3. Разработайте класс траспорт. Данные члены: название транспорта, скорость в км / ч, затраты на топливо (л / км). Функции-члены: вывод времени затраченного на преодоление пути и расход топлива. Аргументы функции: название транспорта и расстояние.

4. Разработать класс со следующими членами. Данные члены: фамилия студента, массив [10] оценок. С помощью функции вывести «хорошая успеваемость», если кол-во 5 = кол-во 3, а остальные 4 или все 5, иначе вывести «плохая успеваемость».

5. Разработать класс для работы со строками. Функции-члены: преобразование первой буквы в большую, подсчет гласных букв, подсчет других символов.

6. Разработать класс для шифрования. Данные-член - ключ цифр. Данные-функции: шифратор, дешифратор. К каждой буквы предложения добавить ключ цифру (шифрование). Дешифровки - обратный процесс.

7. Разработать класс для работы с паролем. Данные члены: логин, пароль. Функции-члены: проверка пароль = логин, наличие букв и цифр в пароле, проверка длины пароля, если пароль! = Логин, длина пароля> = 6 и выполняется второе условие вывести «безопасный пароль», иначе «смените пароль».

8. Выполните задание 2 так, чтобы программа использовала ваш строчный объект для чтения и отображения текстового файла.

9. Используя созданный вами класс TTime, напишите программу с именем DAY.CPP, которая сообщает день недели для любой даты, передана в качестве параметра командной строки.

10. Определите класс для анализа, запоминание, вычисления и печати простых арифметических выражений, состоящих из целых констант и операторов +, -, \*, /. Общий интерфейс должен выглядеть так:

class expr {

public: expr (char \*)

             int eval ();

              void print ();

};

Аргумент для конструктора - выражение. Функция expr :: eval () возвращает значение выражения. Print () выводит выражение на экран.

К примеру:

expr x ( "123/4 + 123 \* 4 - 3")

cout << "x =" << x.eval () << "\ n";

 x.print ();

11. Разработать класс матрицы. Параметры: указатель на область, количество строк и столбцов. В классе предусмотреть 3 конструкторы:

• по умолчанию: создание матрицы 3х3 и заполнения случайными значениями,

• с одним параметром - создание квадратной матрицы и заполнения ее,

• с двумя - соответственно.

Функция-метод должна находить максимальное и минимальное число в строке.

5. Разработать классы Ford и Opel. Конструкторы с параметрами (модель, год выпуска авто). Конструктор инициализирует модель, год выпуска авто, в зависимости от марки (предусмотреть 5+) объем двигателя и другие параметры авто. Разработать метод, который выводит информацию об автомобиле. пример:

Марка: Ford

Модель: Fiesta

Год выпуска: 2012

Тип кузова ...

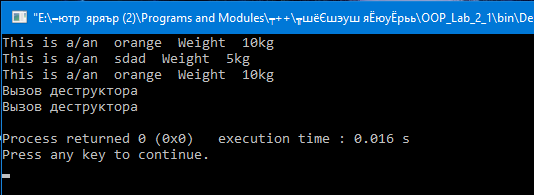
6. Создать класс, оптимизирует покупку посуды. Параметры конструктора класса: имя коллекции, цена предмета. Найти стоимость всех возможных комплектов из трех предметов и, как вариант, все предметы. Разработать функцию-член, принимает сумму денег и номер варианта, а возвращает количество выбранных комплектов, что можно купить на эту сумму.

12. Создать класс, который имеет конструктор с параметрами, необходимыми для вычисления таких значений для конуса, как: площадь основания конуса, площадь боковой поверхности, площадь поверхности конуса и объем. Для каждой искомой величины разработать методы.

13. Создать класс с конструктором, который получает все необхиднi аргументы для определения значения по формуле геометрической прогрессии. Использовать параметры по умолчанию.

14. Создать класс с конструктором, который получает все необхиднi аргументы для определения значения по формуле Бернулли. Все параметры должны быть по умолчанию.

15. Создать класс для авторизации пользователей. В конструкторi использовать параметры по умолчанию (user = "user", pass = "", dateOfBday, tel). Разработать метод проверки данных, передаваемых пользователем. Разработанный метод имеет выводить карточку пользователя с параметрами переданными конструктору, а если имя и пароль соответствуют администраторским, то вывести карточку администратора.

Задание №1

Есть класс с именем Tfruit и объект orange типа Tfruit. Используйте orange для инициализации нового объекта на имя grapefruit с помощью конструктора копирования.

//Листинг TFruit.cpp

#include "Tfruit.h"

#include <iostream>

#include <string.h>

using namespace std;

Tfruit::Tfruit(char \*s, int ww)

{

name = s;

w = ww;

}

Tfruit::Tfruit( Tfruit &copys)

{

w = copys.w;

if (copys.name)

name=strdup(copys.name);

else name=NULL;

}

Tfruit::~Tfruit()

{

cout << "Вызов деструктора "<< endl;

}

void Tfruit::show()

{

cout << "This is a/an " << name << " Weight " << w << "kg\n";

}

//Листинг Tfruit.h

#ifndef TFRUIT\_H

#define TFRUIT\_H

#include <string>

class Tfruit

{

public:

Tfruit();

Tfruit(char \*s, int ww);

Tfruit(Tfruit &copys);

~Tfruit();

void show();

private:

char \* name;

int w;

};

#endif // TFRUIT\_H

// Листинг main.cpp

#include <iostream>

#include <Tfruit.h>

#include <string>

using namespace std;

int main()

{

setlocale(LC\_ALL,"rus");

Tfruit orange("orange", 10);

orange.show();

Tfruit grapefruit("sdad", 5);

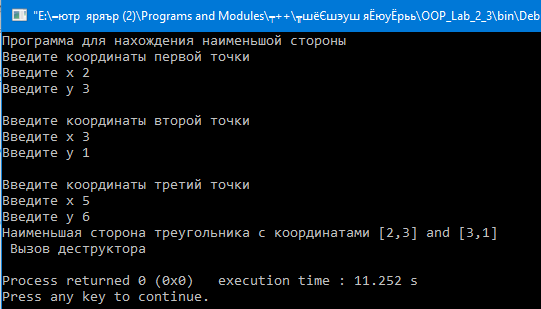
grapefruit.show();

grapefruit=orange;

grapefruit.show();

return 0;

}

Задание №2 . Определить класс треугольника где функция-член класса определяет меньшую сторону треугольника (по формуле определенния расстоянии между двумя точками на координатной плоскости).

//Листинг treugol.h

#ifndef TREUGOL\_H

#define TREUGOL\_H

struct point{

int x,y; };

struct side{

point one,two;};

class treugol

{

private :

side first, second, third;

public:

treugol();

treugol( side &p, side &t, side &w);

~treugol();

side func();

};

#endif // TREUGOL\_H

//Листинг treugol.cpp

#include "treugol.h"

#include <iostream>

#include <cmath>

using namespace std;

treugol::treugol( side &p, side &v, side & t)

{

first = p;

second = v;

third = t;

}

treugol::~treugol(){

cout << " Вызов деструктора "<< endl;

}

side treugol:: func(){

double res\_1, res\_2, res\_3;

res\_1 = sqrt(pow( abs((first.one.x-first.two.x)),2)+pow(abs ((first.one.y-first.two.y)),2));

res\_2 = sqrt(pow(abs ((second.one.x-second.two.x)),2)+pow( abs ((second.one.y-second.two.y)),2));

res\_3 = sqrt(pow( abs((third.one.x-third.two.x)),2)+pow(abs ((third.one.y-third.two.y)),2));

if (res\_1<res\_2)

{

if(res\_1<res\_3)

return first;

else return third;

} else if (res\_2< res\_3)

return second;

else return third;

}

//Листинг main.cpp

#include <iostream>

#include <treugol.h>

using namespace std;

int main()

{

point p\_1;

point p\_2;

point p\_3;

setlocale(LC\_ALL, "rus");

cout << "Программа для нахождения наименьшой стороны\n" ;

cout << "Введите координаты первой точки \n";

cout << "Введите x ";

cin >> p\_1.x;

cout << "Введите y ";

cin>> p\_1.y;

cout << "\nВведите координаты второй точки \n";

cout << "Введите x ";

cin >> p\_2.x;

cout << "Введите y ";

cin>> p\_2.y;

cout << "\nВведите координаты третий точки \n";

cout << "Введите x ";

cin >> p\_3.x;

cout << "Введите y ";

cin>> p\_3.y;

side side\_1 {p\_1, p\_2};

side side\_2 {p\_2, p\_3};

side side\_3 {p\_3, p\_1};

treugol object(side\_1, side\_2, side\_3);

side side\_result = object.func();

cout << "Наименьшая сторона треугольника с координатами "

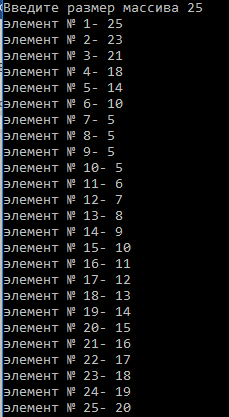
<< "[" <<side\_result.one.x<< "," << side\_result.one.y << "]" << " and "

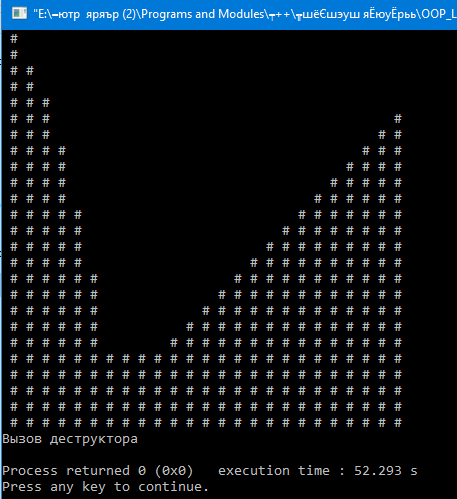
<< "[" <<side\_result.two.x<<"," <<side\_result.two.y << "]" << endl;

return 0;

}

Задание №3 Определите класс histogram, который подсчитывает числа в определенных интервалах, описанных в качестве аргументов конструктора histogram. Напишите функции для вывода гистограммы. Предусмотрите обработку значений, выходящих за пределы определенной области.





// Листинг histogram.h

#ifndef HISTAGRAM\_H

#define HISTAGRAM\_H

class histagram

{

private :

int \*c\_point;

int col\_elementov;

public:

histagram(int \* point, int counts);

~histagram();

void show\_histagram();

};

#endif // HISTAGRAM\_H

// Листинг histogram.cpp

#include <iostream>

#include "histagram.h"

histagram::histagram(int \*point, int counts)

{

c\_point = point;

col\_elementov = counts;

}

histagram::~histagram()

{

std::cout << "Вызов деструктора " << std::endl;

}

void histagram::show\_histagram()

{

int max=0;

for(int i = 0; i<col\_elementov; i++)

{ if (c\_point[i]>max)

max=c\_point[i]; }

for (int i = max; i>0 ; i--){

for(int j=0; j<col\_elementov; j++)

{

if (c\_point[j]>=i )

std::cout << " #" ;

else std::cout <<" " ;

}

std::cout <<std::endl;

}

}

// Листинг main.cpp

#include <iostream>

#include <histagram.h>

using namespace std;

int main()

{

setlocale (LC\_ALL, "rus");

int n;

cout << "Введите размер массива " ;

cin >> n;

int array[n];

for (int i = 0; i < n; i++)

{

cout << "элемент № " << i+1 << "- ";

cin >> array[i];

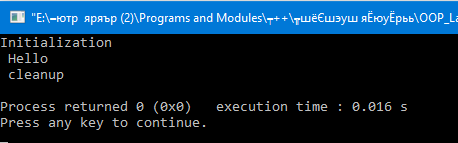
}

histagram object(array, n);

object.show\_histagram();

return 0;

}

Задание №4 Данная программа:

    #include <iоstream.h>

    main ()

   {

         cout << "Hello! \ N ";

return 0;

    }

 Модифицируйте ее так, чтобы она выводила на выходе:

              Инициализация,     Здравствуйте!             Очистка

   Не изменяйте main () !!!.

// Листинг base.h

#ifndef BASE\_H

#define BASE\_H

class base

{

public:

base();

~base();

};

#endif // BASE\_H

// Листинг base.cpp

#include "base.h"

#include <iostream>

base::base()

{

std::cout << "Initialization " << std::endl;

}

base::~base()

{

std::cout << " cleanup " << std::endl;

}

// Листинг main.cpp

#include <iostream>

#include <base.h>

base p;

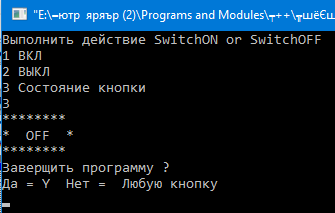
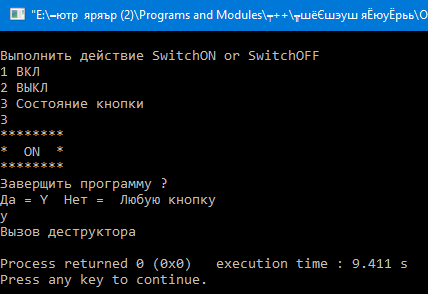
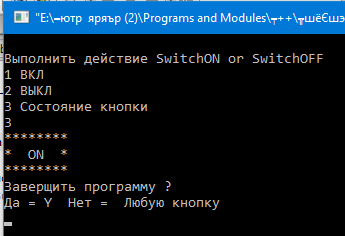
int main()

{

std::cout << " Hello " << std::endl;

}

Задание №5 Разработайте класс кнопки, который можно использовать в моделировании, где необходимы переключатели включен / выключен (данные-члены класса) и функции-члены класса определяющих состояние включенной кнопки

 и выключенной кнопки.

// Листинг Button.h

#ifndef BUTTON\_H

#define BUTTON\_H

class Button

{

public:

Button(bool s);

~Button();

void status();

void on();

void off();

private :

bool but\_on;

};

#endif // BUTTON\_H

// Листинг Button.cpp

#include "Button.h"

#include <iostream>

Button::Button(bool s)

{

but\_on = s;

}

Button::~Button()

{

std:: cout << "Вызов деструктора \n";

}

void Button::status()

{

if (but\_on) {

std::cout << "\*\*\*\*\*\*\*\*\n"

<< "\* ON \*\n"

<< "\*\*\*\*\*\*\*\*\n";

} else { std::cout << "\*\*\*\*\*\*\*\*\n"

<< "\* OFF \*\n"

<< "\*\*\*\*\*\*\*\*\n";

}

}

void Button:: on()

{

if(!but\_on)

but\_on = true;

}

void Button:: off()

{

if(but\_on)

but\_on = false;

}

// Листинг main.cpp

#include <iostream>

#include <Button.h>

#include <Windows.h>

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

Button object(true);

int ch, but, max\_=20;

do {

system("cls");

cin.clear();

cin.ignore(max\_, '\n');

cout << "Выполнить действие SwitchON or SwitchOFF\n"

<< "1 ВКЛ \n"

<< "2 ВЫКЛ \n"

<< "3 Состояние кнопки\n";

cin >> ch;

switch (ch)

{

case 1 : object.on();

break;

case 2 : object.off();

break ;

case 3 : object.status();

break;

default: cout << "Неправильный выбор повторите попытку\n";

}

cout << "Заверщить программу ?\n"

<< "Да = Y " << "Нет = Любую кнопку \n";

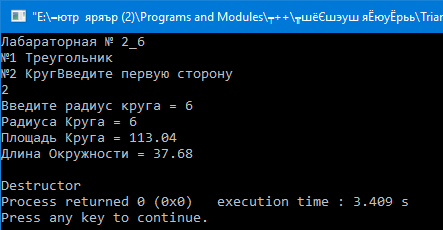
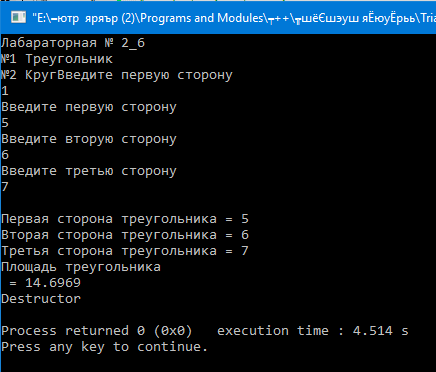
cin >> but;

} while (but != 'y');

return 0;

}

Задание №6 Определить классы Triangle и Circle и найти площади указанных фигур (треугол. и круга).



// Листинг triangel.h

#ifndef TRIANGEL\_H

#define TRIANGEL\_H

class Triangel

{

private:

double side\_1;

double side\_2;

double side\_3;

double square;

public:

Triangel(double a=1, double b=1, double c=1);

~Triangel();

void show();

};

#endif // TRIANGEL\_H

// Листинг triangel.cpp

#include "triangel.h"

#include <iostream>

#include <cmath>

Triangel::Triangel(double a, double b, double c)

{

side\_1 = a;

side\_2 = b;

side\_3 = c;

}

Triangel::~Triangel()

{

std::cout << "\nDestructor \n";

}

void Triangel::show()

{

double pol;

pol =(side\_1+side\_2+side\_3)/2;

square = sqrt(pol\*(pol-side\_1)\*(pol-side\_2)\*(pol-side\_3));

std::cout << "\nПервая сторона треугольника = " << side\_1;

std::cout << "\nВторая сторона треугольника = " << side\_2;

std::cout << "\nТретья сторона треугольника = " << side\_3;

std::cout << "\nПлощадь треугольника \n = " << square;

}

// Листинг Circle.h

#ifndef CIRCLE\_H

#define CIRCLE\_H

class Circle

{

private:

double radius;

double c\_square;

double c\_lenght;

public:

Circle();

Circle(double m\_radius=1);

~Circle();

void show();

};

#endif // CIRCLE\_H

// Листинг Circle.cpp

#include "Circle.h"

#include <iostream>

Circle::Circle(double m\_radius)

{

radius = m\_radius;

}

Circle::Circle()

{

}

Circle::~Circle()

{

std:: cout << "\nDestructor";

}

void Circle::show()

{

c\_lenght = (2\*3.14\*radius);

c\_square = (3.14\*radius\*radius);

std::cout << "Радиуса Круга = " << radius << std::endl;

std::cout << "Площадь Круга = " << c\_square << std:: endl;

std::cout << "Длина Окружности = " << c\_lenght << std:: endl;

}

// Листинг main.cpp

#include <iostream>

#include <triangel.h>

#include <Circle.h>

#include <clocale>

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

double side\_1\_g, side\_2\_g, side\_3\_g, radius\_g;

int choose\_value;

cout << "Лабараторная № 2\_6\n";

cout << "№1 Треугольник ";

cout << "\n№2 Круг";

cout << "Введите первую сторону \n";

cin>>choose\_value;

switch(choose\_value)

{

case 1 : {

cout << "Введите первую сторону \n";

cin>>side\_1\_g;

cout << "Введите вторую сторону \n";

cin>>side\_2\_g;

cout << "Введите третью сторону \n";

cin>>side\_3\_g;

Triangel object\_1(side\_1\_g, side\_2\_g, side\_3\_g);

object\_1.show();

break;}

case 2: {

cout << "Введите радиус круга = ";

cin >> radius\_g;

Circle object\_2(radius\_g);

object\_2.show();

break;}

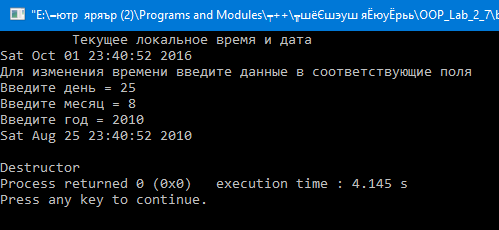
default : cout << "Error \n";

}

return 0;

}

Задание №7 С помощью класса Ttime напишите MAIN.Cpp, которая выводит текущие дату и время и заменяет дату в вашем объекте на завтрашнюю.



// Листинг Ttime.h

#ifndef TTIME\_H

#define TTIME\_H

#include <ctime>

class Ttime

{

private:

time\_t now;

struct tm \* timeinfo;

public:

Ttime();

~Ttime();

void show\_obj\_time();

void current\_time();

void change\_time(int a, int b, int c);

void show();

};

#endif // TTIME\_H

// Листинг Ttime.cpp

#include "Ttime.h"

#include <iostream>

#include <ctime>

Ttime::Ttime()

{

}

Ttime::~Ttime()

{

std::cout << "\nDestructor";

}

void Ttime::current\_time()

{

time( &now );

timeinfo = localtime( &now );

std::cout << "\t Текущее локальное время и дата \n" << asctime(timeinfo);

}

void Ttime:: change\_time(int a, int b, int c)

{

timeinfo->tm\_mday =a;

timeinfo->tm\_mon = b-1;

timeinfo->tm\_year =c-1900;

}

void Ttime::show()

{

//std::cout << "Дата : "<< timeinfo->tm\_mday <<"."<< timeinfo->tm\_mon << "." << timeinfo->tm\_year;

std:: cout<<asctime(timeinfo);

}

// Листинг main.cpp

#include <iostream>

#include <Ttime.h>

#include <ctime>

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

int g\_day, g\_year, g\_month;

Ttime object;

object.current\_time();

cout << "Для изменения времени введите данные в соответствующие поля\n";

cout << "Введите день = ";

cin >> g\_day;

cout << "Введите месяц = ";

cin >> g\_month;

cout << "Введите год = ";

cin >> g\_year;

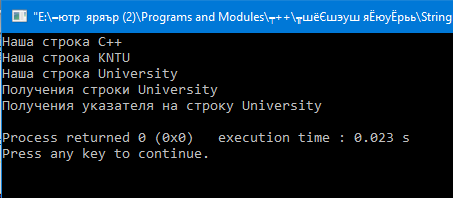
object.change\_time(g\_day, g\_month, g\_year);

object.show();

return 0;

}

Задание №8. Создайте класс, способный хранить строку в купе. Должна существовать возможность передачи строки объекта вашего класса и затем получения указателя на такою же строку. Должно быть также возможность изменения строки объекта класса. Используйте конструкторы и деструктор для введения всех автоматических инициализаций и очистки объекта.



// Листинг stringBad.h

#ifndef STRINGBAD\_H

#define STRINGBAD\_H

class stringBad

{

private:

char \*str;

int len;

public:

stringBad(const char \*s);

stringBad();

~stringBad();

stringBad(stringBad &copys);

void changed(const char\* s);

char\* returns(){return str;};

void show();

};

#endif // STRINGBAD\_H

// Листинг stringBad.cpp

#include "stringBad.h"

#include <iostream>

#include <cstring>

stringBad::stringBad()

{

len = 4;

str = new char[4];

std::strcpy(str, "C++");

}

stringBad::stringBad(const char\* s)

{

len = std::strlen(s);

str = new char[len+1];

std::strcpy(str, s);

}

stringBad::~stringBad()

{

delete[] str;

}

stringBad::stringBad(stringBad &copys)

{

if (copys.str)

str=strdup(copys.str);

else str=NULL;

}

void stringBad::changed(const char\* s)

{

delete []str;

len = std::strlen(s);

str = new char[len+1];

std::strcpy(str, s);

}

void stringBad::show ()

{

std::cout << "Наша строка "<< str << std::endl;

}

// Листинг main.cpp

#include <iostream>

#include <stringBad.h>

#include <cstring>

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

stringBad object;

object.show();

stringBad object\_2("KNTU ");

object\_2.show();

object\_2.changed("University");

object\_2.show();

cout << "Получения строки " << object\_2.returns() << endl;

char \*str\_1;

str\_1= strdup(object\_2.returns());

cout << "Получения указателя на строку " << str\_1 << endl;

return 0;

}