**лекция 4**

**Тема лекции:** Программирование классов. Спецификаторы доступа к членам класса. Использование функций-членов. Создание и уничтожение объектов класса с помощью конструкторов и деструкторов. Глобальные и локальные объекты класса. Массивы объектов класса. Почленно инициализация. Конструктор копии.

классы

Класс - это вид структуры, объединяющей данные и функции. К примеру:

class TTime

{

public:

int year;

int month;

int day;

int hour;

int minute;

void Display (void);

};

Объявление класса начинается с ключевого слова class. В скобках заключены 6 членов класса, которым предшествует слово public. Это спецификатор доступа. Он открывает доступ ко всем членам, идущие за ним для всех пользователей класса. Такие члены называются открытыми.

**Функции-члены** класса объявлены прототипами функций. Они выполняют некоторые операции над данными членами класса. Тело функции размещается в другом месте программы. Имени функции-члена предшествует имя класса и оператор расширения области видения. К примеру:

void TTime :: Display (void);

Здесь заголовок функции указывает компилятору, что Display () - член класса TTime. В программе могут быть другие функции и функции-члены других классов с именем Display (), не вызывают конфликтов имен. Внутри функции-члена операторы имеют прямой доступ к членам класса.

Класс - это лишь шаблон, схема, описывающая формат членов класса. Чтобы использовать класс, необходимо создать объект типа класса. Для доступа к членам объекта класса используется оператор ". ". Например: используем класс TTime, что был описан:

main ()

{

TTime appointment; // Объект типа TTime

appointment.month = 7;

appointment.day = 14;

appointment.year = 2013;

appointment.hour = 8;

appointment.minute = 30;

cout << "Appontment ==";

аppointment.Display (); // Вызов функции-члена класса

cout << '\ n';

return 0;

}

void TTime :: Display (void)

{

char s [32];

sprintf (s, "Date:% 02d /% 02d /% 04dTime:% 02d:% 02d \ n", month, day, year, hour,

minute)

cout << s;

}

В программе могут быть другие объекты класса TTime:

TTime today;

TTime tomorrow;

TTime yesterday;

Этим объектам можно присвоить значение и затем отобразить их с помощью таких операторов:

today.Display ();

tomorrow.Display ();

yesterday.Display ();

С ++ позволяет структурам и объединениям иметь функции-члены подобно классам.

Структура - это класс, все члены которого открыты. На практике функции-члены в структурах и объединениях используются очень редко.

В рассматриваемом примере при изменении данных членов класса необходимо будет менять операторы присваивания. Создадим новый класс, облегчит изменение структуры данных членов класса:

#include <iostream.h>

#include <stdio.h>

class TTime

{

private:

int year, int month, int day, int hour, int minute;

public:

void Display (void);

void GetTime (int & m, int & d, int & y, int & hr, int & min)

void SetTime (int m, int d, int y, int hr, int min)

};

main ()

{

TTime appointment;

int month, day, year, hour, minute;

appointment.SetTime (7,14,1996,8,30)

cout << "Appointment ==";

appointment.Display ();

appointment.GetTime (month, day, year, hour, minute)

appointment.SetTime (month, day, year, ++ hour, minute)

cout << "Next hour ==";

appointment.Display ();

return 0;

}

void TTime :: Display (void)

{...... ....}

void TTime :: GetTime (int & m, int & d, int & y, int & hr, int & min)

{

m = month;

d = day;

y = year;

hr = hour;

min = minute;

}

void TTime :: SetTime (int m, int d, int y, int hr, int min)

{

month = m;

day = d;

year = y;

hour = hr;

minute = min;

}

В данной программе 3 дополнения:

Спецификатор доступа private делает данные-члены закрытыми. Теперь только функции-члены класса могут непосредственно ссылаться на эти данные-члены. То есть закрытые члены невидимые вне класса. Члены класса закрыты по умолчанию. В классе могут присутствовать открытые и закрытые члены, располагаются в произвольном порядке, функции-члены могут быть объявлены в закрытой части, но обычно они открыты.

Добавлены функции-члены GetTime () и SetTime (). Из функции GetTime () значения компонентов объекта класса передаются обратно, к аргументам оператора вызова, с помощью параметров-посылок. В SetTime () значения параметров присваиваются данным-членам класса. Поскольку данные-члены класса TTime закрыты, поэтому невозможно присваивать значения непосредственно объекта класса. Для использования закрытых данных-членов класса существует единственный способ - вызов функций-членов данного класса.

Большинство классов лучше объявлять в заголовочный файлах. Однако, это не является обязательным средством. Пример: объявление класса TTime в заголовочный файл time1.h:

class TTime

{

 private:

 int year;

 int month;

 int day;

 int hour;

 int minute;

 public:

 void Display (void);

 void GetTime (int & m, int & d, int & y, int & hr, int & min)

 void SetTime (int m, int d, int y, int hr, int min)

 };

Объявление класса в заголовочный файл позволяет использовать этот класс в других модулях. Тела функций-членов приводятся в отдельном файле, что заканчивается на .CPP. Включается создан заголовочный файл в программу так:

#include "time1.h"

В большинстве случаев вызовы функций-членов вместо прямого доступа к данным членов сказываются на быстродействии программы. Для достижения наибольшей эффективности программ, С ++ позволяет объявлять классы с функциями-членами, встраиваемые. Хотя встроенные функции-члены используются так же, как и другие функции, в скомпилированном коде они не вызываются. Они непосредственно встраиваются в скомпилированную программу. К примеру:

class TTime

{

public: ...... ....

char \* GetSTime (void)

{

char \* cp = strdup (ctime (& dt))

return cp;

}

void Display (void)

{

cout << ctime (& dt)

}

};

Тела функций-членов, встраиваемые, вложенные в фигурные скобки сразу после заголовка функции в объявлении класса. В данном случае функции-члены встраиваемых объявлялись непосредственно в классе. Но можно их объявить и в модуле

реализации класса с помощью ключевого слова inline. Поставив inline перед именем функции, можно и не добиться желаемого результата, поскольку реализация функций-членов встраиваемых должно быть доступна компилятору до того, как эта функция будет вызвана.

Функции-члены встраиваемые лучше распознаются, когда они объявлены непосредственно в классе. В объявлении класса можно создать несколько функций с одинаковыми именами, отличающиеся хотя бы одним параметром. Их тела реализуются отдельно. Параметры функций-членов по умолчанию используются в объявлении классов и не приводятся при вызове функции-члена.

Пример объявления функции-члена в классе:

void SetTime (int m = 1, int d = 1, int y = -1) .....

Реализация функции-члена вне класса:

void TTime :: SetTime (int m, int d, int y)

{

// Тело функции

}

**Конструкторы и деструкторы**

 В классах может быть объявлено несколько конструкторов для автоматической инициализации объекта класса при его создании. Деструктор, предназначенный для очистки при выходе объекта класса из области видение, может быть только один. С ++ автоматически вызывает конструкторы и деструкторы. Конструкторы объявляются как функции-члены, которые не возвращают никаких значений, с произвольным числом параметров любого типа. Конструкторы чаще объявляются в открытой секции, но могут определяться в любом месте класса. Они имеют то же имя, что и класс, для которого объявляются. К примеру:

class TTime

{

private:

long dt;

char \* dts;

void DeleteDts (void);

public:

TTime (); // Конструктор

TTime (int m, int d = 1, int y = -1) // Конструктор

~ TTime (); // Деструктор

           ...

};

Конструктор объявлен без параметров, вызывается по умолчанию. Здесь он объявлен первым. Вторым объявляется перегружен конструктор с тремя действительными параметрами, два из которых передают значения по умолчанию. Тела конструкторов должны находиться в модуле реализации класса:

TTime :: TTime ()

{

dts = NULL; // Обнуление текущей строки

SetTime (1, 1, -1);

}

TTime :: TTime (int m, int d, int y)

{

dts = NULL;

SetTime (m, d, y)

}

TTime :: ~ TTime ()

{

delete dts; // Удаление строки сохраняется в объекте

dts = NULL; // Обнуление указателя

}

Программирование внутренних операторов конструкторов ничем не отличается от программирования других функций-членов. Но обязанности конструктора ограничены присвоением начальных значений данным-членам класса, выделением памяти, используемой объектом класса и т.д.

В данном случае символьный указатель dts устанавливается равным NULL. Это означает, что указатель еще не ссылается на строку. С ++ автоматически вызывает конструктор для инициализации объекта класса TTime, поэтому все такие объекты гарантированно имеют инициализирован указатель dts. В любом классе можно объявить столько конструкторов, сколько нужно. Они должны отличаться хотя бы одним параметром.

Пример: использование конструкторов класса TTime:

main ()

{

TTime t1;

TTime t2 (8);

TTime t3 (8,1)

TTime t4 (8,1,2013)

t1.Display ();

t2.Display ();

t3.Display ();

t4.Display ();

return 0;

}

Здесь объявляются четыре объекта класса TTime. С ++ автоматически вызывает конструктор класса при создании объекта. Здесь для t1 вызывается конструктор для инициализации объекта. Далее вызывается конструктор с параметрами.

Деструкторы обычно объявляются в открытой секции класса, но могут и не быть объявленными совсем. Класс может иметь не более одного деструктора с именем класса, в котором он объявлен. Перед объявлением деструктора всегда ставится знак "~" (тильда). Если объект класса выходит из области видение, то С ++ автоматически вызывает деструктор объекта класса. Это дает объекта удобную возможность для самоочищения. Обычно явно деструктор не вызывается.

**Глобальные и локальные объекты класса. Указатели на объекты**

  Каждая функция-член класса получает скрытый параметр с именем this, содержащий адрес объекта, для которого была вызвана эта функция.

Объекты класса могут быть глобальными или локальными в функции. Как и переменные обычных типов, объекты класса могут также адресоваться указателями и посылками.

Природа объекта влияет на то, когда будут вызываться конструкторы и деструкторы этого объекта класса. Конструкторы глобальных объектов класса вызываются до начала выполнения функции main (). Это обеспечивает инициализацию всех глобальных объектов формального начала выполнения программы. Например, глобальное объявления

TTime today;

создает глобальный объект today класса TTime. До вызова main () С ++ вызывает для today конструктор TTime по умолчанию. объявление:

TTime SomeTime (9,4,2013)

инициализирует глобальный объект SomeTime класса TTime конкретной датой. Наличие списка параметров заставляет С ++ заняться поиском в классе TTime конструктора, способного принять указанные аргументы. Деструкторы глобальных объектов класса вызываются в качестве части кода завершения программы. Автоматические объекты класса, объявленные локальными в функции, создаются вместе с вызовом функции и уничтожаются после ее завершения. Данные члены автоматического объекта класса хранятся в куче. В функции

void AnyFunction (void)

{

TTime now;

   ...

}

по умолчанию вызывается конструктор TTime () для инициализации объекта класса now каждый раз, когда вызывается функция. Когда функция завершит работу, С ++ вызывает деструктор TTime для объекта now. Если вызывается функция exit () для завершения программы, деструкторы глобальных объектов класса вызываются как обычно, а деструкторы любых существующих автоматических объектов, локальных в функции, не вызываются.

Указатели могут ссылаться на динамические объекты класса, пространство для которых обычно выделяется в куче с помощью new. Объявим указатель pToday на объект класса типа TTime следующим образом:

TTime \* pToday;

Это объявление может быть как глобальным, так и локальным в функции. Как и все указатели, pToday необходимо инициализировать до его использования:

pToday = new TTime;

Здесь с помощью оператора new выделяется пространство в куче для объекта типа TTime. Адрес первого байта этого объекта присваивается pToday. С ++ в этом примере вызывает конструктор объекта по умолчанию. Для использования другого конструктора:

pToday = new TTime (9,4,2013)

Указатели на объекты класса используются так же, как и указатели на объекты других типов. К примеру:

sp = (\* pToday) .GetSTime ();

Происходит привласнення переменной sp результата функции члена GetSTime () для объекта класса, на который ссылается pToday. оператор:

sp = pToday -> GetSTime ();

выполняет те же действия.

Деструктор динамического объекта класса вызывается при его уничтожении. оператор

delete pToday;

уничтожает объект, на который ссылается pToday, освобождая память этого объекта. Непосредственно перед разрушением объекта С ++ вызывает деструктор класса. Например, в случае с классами TTime деструктор может освободить память, на которую ссылается указатель dts.

Объекты класса могут объявляться как посылки.

пример:

TTime today; // Глобальный объект today

TTime & rToday = today; // Посылка на today

Посылка rToday может использоваться вместо today. К примеру:

rToday.Display ();

today.Display ();

Эти записи эквивалентны. Поскольку посылки ссылаются на реально существующие объекты, то при входе в область действия посылки конструктор класса не вызывается. Соответственно не вызывается и деструктор при выходе из области действия посылки. Эти действия выполняются только при создании и удалении самого объекта. Можно передавать объекты класса, указатели на объекты класса и посылки на них в качестве аргументов функций. пример:

void AnyFunction (TTime t)

{

t.Display ();

}

Функция AnyFunction () имеет параметр t типа TTime. В функции вызывается функция-член Display () для объекта t.

Программа может объявить объект класса TTime:

TTime today;

и передать его по значению функции AnyFunction ():

AnyFunction (today)

Функции также использует в качестве параметров указатели на класс. Передача функции крупных объектов класса по значению приводит к выделению пространства нежелательного размера. В таких случаях лучше использовать параметры-указатели:

void AnyFunction (TTime \* tp)

{

tp -> Display ();

}

Для вызова такой функции оператор вызова передает адрес объекта класса TTime вместо самого объекта:

AnyFunction (& today)

Параметры функции могут быть объявлены как посылки на объекты класса. Результаты будут аналогичны предыдущему случаю, только внутри функции не будут нужны операции доступа к указателю:

void AnyFunction (TTime & tr)

{

tr.Display ();

}

В другом месте программы можно передать объект класса TTime непосредственно в качестве параметра посылки AnyFunction ():

AnyFunction (today)

Но такой оператор похож на передачу аргумента today по значению. Это - недостаток параметров-посылок. Функции могут возвращать объекты классов непосредственно, в качестве указателей или как посылки. Реже используется функция, возвращает непосредственно объект класса.

Пример: функция, выделяет в памяти новый объект класса TTime и возвращает его адрес:

TTime \* newTime (void)

{

TTime \* p = new TTime;

return p;

}

Внутри функции указателе p на TTime присваивается адрес объекта класса

TTime, созданного с помощью new. Указатель существует только внутри функции, но объект, на который ссылается этот указатель - глобальный. Можно использовать функции, на которые ссылаются, для посылок на существующие объекты:

TTime & newTime (void)

{

return today;

}

В другом месте можно объявить посылку на TTime и присвоить ей результат newTime ():

TTime & tr = newTime ();

Теперь посылка tr - псевдоним для today и может использоваться в операторе вида:

newTime (). Display ();

**Массивы объектов класса**

Массивы объектов класса объявляются так же, как и массивы данных обычных типов. К примеру:

TTime tenTimes [10];

Но существует правило: объекты класса, будут содержать массив, должны иметь конструкторы по умолчанию. При создании массивов объектов класса вызывается конструктор класса по умолчанию для каждого объекта массива. Не существует простого средства инициализировать одни объекты класса из массива с помощью конструктора по умолчанию, а другие - с помощью альтернативного конструктора. В таких случаях объявляется м ассив, а затем инициализируются заново значение нужных объектов заданным образом. Массивы объектов класса могут также размещаться в куче и адресоваться с помощью указателей. пример:

main ()

{

TTime \* tarray;

tarray = new TTime [6];

for (int i = 0; i <6; i ++)

tarray [i] .Display ();

delete [] tarray;

return 0;

}

Здесь объявляется указатель tarray на объект класса TTime. Для инициализации указателя вызывается new, что выделяет в куче массив из 6 объектов и присваивается адрес первого объекта массива. В связи с обращением к new 6 раз автоматически вызывается конструктор по умолчанию класса TTime для инициализации каждого объекта в массиве. После завершения использования массива объектов, его нужно удалить. Для указания компилятору, что tarray ссылается на массив в операторе delete используются пустые скобки. Такая команда обеспечивает вызов деструктора для каждого объекта в массиве.

**Копирование объектов классов. конструктор копии**

Когда один объект класса копируется в другой объект совместного типа, могут образоваться неожиданные результаты. Особенно это касается классов, в которых объявлены члены-указатели на данные. Возникают такие опасные ситуации:

• Один объект используется при объявлении нового объекта этого же класса для его инициализации.

• Объект передается функции параметром типа класса по значению.

• Функция возвращает объект класса (но не посылку или указатель на объект).

• В операторе один объект присваивается другому.

В первых трех случаях происходит создание и инициализация новой копии объекта существующим объектом класса. В четвертом случае один существующий объект присваивается другому, ранее объявленном объекта. Существует разница между копированием объекта и присвоением одного объекта другому. В первых трех случаях объекты создаются с помощью вызова конструктора, в четвертом - присвоение одного объекта другому создает нового объекта и не приводит к вызову конструктора.

Копирование объектов может привести к появлению указателей, ссылаются на то же самое место в памяти. Удаление одного из таких указателей приведет к тому, что другие указатели будут ссылаться на освобожденную участок памяти. А удаление той же участки памяти более одного раза в деструкторе класса может привести к разрушению вместе. В С ++ существует два механизма для безопасного копирования и присвоение объектов класса, имеют члены-указатели: Почленно инициализация и Почленное присвоения.

Когда один объект используется для инициализации другого объекта класса, С ++ копирует каждый член, содержащий данные, из существующего объекта к новому. Например, рассмотрим сначала класс, не имеет членов-указателей:

class TAnyClass

{

private:

int i;

double r;

char \* s;

public:

TAnyClass ()

{

i = 0;

r = 0;

}

TAnyClass (int ii, double rr)

{

i = ii;

r = rr;

}

};

В классе объявлены два конструктора: конструктор по умолчанию и параметризованный конструктор, инициализирует объект класса явными значениями. Создадим объекты этого класса:

TAnyClass V1;

TAnyClass V2 (100,3.14)

Инициализирует новый объект этого класса:

TAnyClass V3 = V2;

Это приведет к копированию объекта V2 в V3 с Почленно инициализации данных-членов объекта V3. То есть, данные члены объекта класса V2 копируются по порядку в члены V3, равнозначно такой записи:

TАnyClass V3;

V3.i = V2.i; // Для демонстрации

V3.r = V2.r; // Оператор не скомпилюються, так как i и r - закрыты.

Почленно инициализация нового класса выполняется и тогда, когда функции передается объект по значению и когда функция возвращает объект класса. Теперь введем в класс закрыт член-указатель:

char \* s;

Добавим также конструктор копии для создания объектов, копируемых с других объектов. Конструктор копии может выполнять любые действия, необходимые для создания безопасной копии, в частности, не дублировать указатели. В случае необходимости С ++ создает конструкторы копии для классов, в которых они не определяются. Конструктор копии объявляется как class (const class &). Как и все конструкторы, он не может быть виртуальным и успадкуватися производным классом. Итак, новая версия TAnyClass:

class TAnyClass

{

private:

int i;

double r;

char \* s;

public:

TAnyClass ()

{

i = 0;

r = 0;

s = NULL;

}

TAnyClass (int ii, double rr, const char \* ss)

{

i = ii;

r = rr;

s = strdup (ss)

}

~ TanyClass ()

{

delete s;

}

const char \* GetStr (void)

{

return s;

}

TAnyClass (TAnyClass & copy)

};

TAnyClass :: TAnyClass (TAnyClass & copy)

{

cout << "конструктор копии \ n";

i = copy.i;

r = copy.r;

if (copy.s) s = strdup (copy.s)

else s = NULL;

}

Конструктор копии TAnyClass (TAnyClass & copy) имеет одинаковое имя с классом, в котором он определен. В нем объявлена ​​посылка & copy на объект класса, данные с которого копируются во вновь созданного объекта. Здесь копируются члены i и r с copy к новому объекту. Чтобы можно было преобразовать строку, на который ссылается указатель s, с помощью функции strdup () выделяется еще один участок памяти в куче для копии исходной строки. Или же, если в указателе s объекта copy содержится 0, указателе нового объекта присваивается NULL. Для скопированных объектов класса TAnyClass нет опасности, что несколько указателей s будут случайно ссылаться на одну и ту же участок памяти. В операторе:

TAnyClass V3 = V2;

вызывается конструктор копии.

ИТОГИ:

• C ++ распознает все ключевые слова, операторы и другие элементы языка ANSI С. В C ++ также есть несколько новых ключевых слов и операторов в дополнение к ANSI С.

• В C ++ необходимо указывать прототипы всех функций Прототип функции должно предшествовать ее вызова.

• Язык C ++ имеет следующие особенности: проверка типов осуществляется более строго, очень ослаблена потребность в typedef-объявлениях, определение char означает выделение 8-битного байта. В C ++ такие литеральни символы, как "Х", имеют тип char. В ANSI С литеральни константы имеют тип int.

• Символ комментария // следует использовать в C ++ для того, чтобы начать комментарий, который продолжается до конца строки. Комментарии C ++ и ANSI С (скобки / \* и \* /) могут использоваться в одной программе.

• В C ++ объявления могут появляться в любом месте любого блока операторов. Например, Вы можете объявить переменную внутри скобок цикла for. Область видение переменной ограничена операторным блоком, в котором она определена. Разумеется, Вы должны объявлять переменные до их использования.

• Если имя локальной переменной совпадает с именем глобальной, локальное имя скрывает имя глобальной переменной. Вы можете использовать оператор расширения области видение для того, чтобы получить доступ к глобальной переменной. Если локальная переменная названа count, то выражение :: count ссылается на глобальную переменную.

• В классах может объявляться конструктор копии, который вызывается, когда один объект используется для инициализации объекта того же класса, если объект передается функции по значению и в случае возвращения функцией объекта класса.

ТЕСТЫ К РАЗДЕЛА 1 для самоконтроля:

1) Класс - это:

a) определенный тип объектов

b) объективная реальность

c) предметная область

d) сообщение с текущим объекта

e) шаблонный тип

2) Объект - это

a) виртуальная функция

b) качество абстракции

c) экземпляр класса

d) новый метод

e) шаблонный тип

3) Вывести данные в С ++ можно с помощью операторов:

a) out object

b) cout <<

c) cin >>

d) print

e) write

4) Выделить память в С ++ можно с помощью операторов:

a) append

b) overload

c) new

d) template

e) delete

5) При создании объекта в С ++ вызывается

a) деструктор

b) виртуальные функции

c) конструктор

d) шаблон

e) встроенная функция

6) По умолчанию все члены класса в С ++:

a) закрыты

b) открыты

c) защищены

d) абстрактные

e) неопределенные

7) Скрытый параметр функции-члену класса this содержит адрес:

a) текущего объекта

b) виртуального объекта

c) функции, параметром которой он

d) исключительной ситуации

e) локального объекта

8) Сколько деструкторов может иметь класс?

a) по запросу пользователя

b) сколько угодно

c) столько же, сколько создается объектов

d) по определению пользователя

e) один

9) В протокол описания класса относятся:

a) данные-члены и функции-члены класса

b) глобальные данные и функции

c) функции препроцессора

d) классы-потомки

e) дружественные функции

10) Какой оператор в С ++ определяет область видение?

a) ==

b)) \*

c) ::

d) &&

e) new

11) В каком месте программы на С ++ объявляются значения аргументов по умолчанию?

a) в прототипе функции

b) в реализации функции

c) в операторных скобках

d) в конструкторе

e) в деструкторе

12) Для чего в С ++ предназначены конструкторы?

a) для реализации функций

b) для объявления класса

c) для инициализации объектов класса

d) для защиты данных класса от копирования

e) для определения предметной области класса

13) Значение какого типа возвращают конструкторы?

a) типа long

b) типа char \*

c) типа float

d) любого типа

e) ничего не возвращают

14) Какой скрытый параметр содержит каждая функция-член класса?

a) указатель на текущий объект

b) указатель на кучу

c) начальные значения данных членов

d) нет такого параметра

e) указатель на объект производного класса

15) Где можно разместить объявление в С ++?

a) в любой точке программы

b) только в начале функции

c) только глобально

d) только внутри класса

e) только в конце функции

16) Когда функции будут перегружены?

a) когда удаляется объект производного класса

b) когда необходимо удалить объекты базового класса

c) при множественной преемственности

d) если в классе есть конструктор копии

e) когда их имена одинаковые, а выполняемая ими работа разная

17) Чем отличаются аргументы, которые передаются по значению от аргументов, передаваемых по посылками?

a) ничем

b) они имеют ключевое слово

c) они не указываются в качестве параметров

d) они используются только в производных классах

e) их нельзя использовать в конструкторах

18) Какой оператор используется для доступа к членам объекта класса?

a) оператор «=»

b) оператор "\*"

c) оператор «&»

d) оператор ». »

e) оператор new

19) Конструктор не может быть:

a) встроенным

b) параметризованным

c) определенным по умолчанию

d) виртуальным

e) определенным вне класса

20) Какой адрес содержит параметр this?

a) объекта производного класса

b) объекта базового класса

c) объекта шаблонного к лакомую

d) объекта абстрактного класса

e) текущего объекта класса

21) Деструкторы объектам вызываются при аварийном завершении программы?

a) локальных

b) глобальных

c) никаких

d) всех без исключения

e) созданных по умолчанию

22) В каких классах нужно определить конструктор копии?

a) в тех, где есть члены-указатели

b) в закрытых

c) в защищенных

d) в абстрактных

e) во всех

23) Есть ли в каждом классе должно быть описан конструктор?

a) только в дружеских

b) только в закрытых

c) только в защищенных

d) только в абстрактных

e) в описании класса конструктор может отсутствовать

24) Какие конструкторы должны иметь объекты класса, содержащих массив?

a) локальные

b) глобальные

c) никаких

d) всех без исключения

e) по умолчанию

25) В какой ситуации не вызывается конструктор копии?

a) объект передается функции параметром типа класса по значению

b) один объект используется при объявлении нового объекта этого же класса для его инициализации

c) функция возвращает объект класса (но не посылку или указатель на объект)

d) вызывается во всех ситуациях без исключения

e) в операторе один объект присваивается другому уже существующему объекту

26) В каком месте программы объявляются глобальные объекты?

a) в любой точке программы

b) только в начале функции

c) до начала работы функции main ()

d) только внутри класса

e) только в конце функции

27) В каком месте программы объявляются локальные объекты?

a) в любой точке программы

b) только в начале функции

c) до начала работы функции main ()

d) только внутри класса

e) только внутри какой-либо функции

28) Как сказывается комментарий в программе, написанной на языке С ++?

a) ==

b)) \*

c) ::

d) //

e) \*\*

29) спецификатор доступа private делает данные-члены:

a) открытыми

b) закрытыми

c) защищенными

d) свободными

e) доступными

30) Можно ли реализовать функцию-член класса вне класса?

a) только по запросу пользователя

b) можно

c) нельзя

d) только если она реализована в классе тоже

e) только если она является виртуальной