Вопросы к экзамену по прикладному программированию

1. **Понятие ООП. Инкапсуляция, полиморфизм, наследование.**

Структурное программирование означает то, что есть некие блоки, на которые делится программа.

ООП – технология, возникшая в кризис ПО.

Любая моделированная система состоит из объектов. Каждый объект характеризуется своим внутренним состоянием.

Инкапсуляция – свойство языка программирования, позволяющее пользователю не задумываться о сложности реализации используемого программного компонента, а взаимодействовать с ним посредством предоставляемого интерфейса, а также объединить и защитить данные.

Наследование – позволяет описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом.

Абстрактный класс – класс, содержащий хотя бы один абстрактный метод, он описан в программе, имеет поля, методы и не может использоваться для непосредственного создания объекта. От такого класса можно только наследовать. Объекты создаются только на основе производных классов, наследованных от абстрактного.

Множественное наследие: у класса может быть более одного предка. Класс наследует методы всех предков.

Полиморфизм – возможность объектов с одинаковой спецификацией иметь различную реализацию. (один интерфейс, множество реализаций)

Абстракция – придание объекту характеристик, которые отличают его от всех других объектов.

1. **Классы и объекты. Общий формат объявления классов. Доступ к членам классов.**

Класс объектов – абстрактное описание, которое объединяет набор объектов. В структурном программировании ключевым объектов является действие.

Классы – описание некоторой структуры программы, обладающей набором внутренних переменных (свойств) и функций, имеющих доступ к свойствам – методов.

Класс объединяет в себе как данные, так и методы обработки этих данных. Класс – это тип данных, созданный пользователем. Под класс память не выделяется. Класс – это всего лишь описание типа данных.

class имя\_класса

{закрытые поля и методы класса

Public: открытые поля и методы класса

} список объектов

К закрытым членам класса можно обращаться только внутри класса.

Создание объекта класса: SimpleClass MyObj

Полю объекта присваивается значение: MyObj.number = 5

1. **Открытые и закрытые члены класса. Объявление. Отличия.**

По умолчанию члены класса являются закрытыми. Закрытыми члены класса делают для ограничения несанкционированного внешнего доступа к этим членам. К закрытым членам класса можно обращаться только внутри класса.

class SimpleClass{

int m,n; - закрытые члены класса

public: - ключевое слово

void show();}; - открытые члены класса

Можно использовать ключевое слово private.

class SimpleClass{

private: - ключевое слово

int m,n; - закрытые

public: - ключевое слово

void show();}; - открытые

1. **Статистические члены класса. Объявление статических членов класса.**

Статистический член является общим для всех объектов класса. В отличии от обычного поля класса статистическое поле не исчезает при удалении объекта.

Когда член класса объявляется как статический, то тем самым компилятору дается указание, что должна существовать только одна копия этого члена, сколько бы объектов этого класса ни создавалось.

static int m;

static void msum(int k);

Пример программы:

#include <iostream.h>

class counter {

static int count;

public:

void setcount(int i) {count = i;};

void showcount () {cout << count << " "; }

};

int counter::count; // определение count

int main() {

counter a, b;

a.showcount (); // выводит 0

b.showcount (); // выводит 0

a.setcount (10); // установка статического count в 10

a.showcount (); // выводит 10

b.showcount (); // также выводит 10

return 0;}

1. **Дружественные функции. Объявление и использование.**

Внешние функции, которые имеют доступ к закрытым членам класса, называются дружественными функциями.

Чтобы задекларировать функцию как дружественную для класса, необходимо указать прототип этой функции в описании класса, предварив его ключевым словом friend.

friend void show(MyClass obj);

Дружественная функция имеет доступ к закрытым членам класса.

class B

class A{

double x;

public:

A(double z) {x = z;}

friend double summa (A a, B b);

}a(3.5);

class B{

double y;

public:

B(double z) {y = z;}

friend double summa (A a, B b);

}b(2.3);

double summa (A a, B b){

return a.x+b.y;}

int main(){

cout << “Total is” << summa(a,b) << endl;

return 0;}

Дружественными по отношению к классу могут быть отдельные методы другого класса или целый класс. В последнем случае все методы дружественного класса имеют доступ к закрытым полям и методам класса.

class B;

class A{

double x;

public:

A(double z) {x = z;}

double summa (B b);

} a(3.5);

class B{

double y;

public:

B(double z) {y = z;}

friend double A::summa(B b);

} b(2.3);

int main(){

cout << “Total is” << a.summa(b) << endl;

return 0;}

double A::summa(B b){

return x + b.y;}

Дружественные классы используются не очень часто – существуют более эффективные способы реализации доступа методов одного класса к членам другого, главным среди которых является механизм наследования.

1. **Создание конструктора в классе и отличие от создания метода в классе.**

Конструктор – метод, который автоматически вызывается при создании объекта. Деструктор – метод, вызываемый автоматически при выгрузке объекта из памяти.

Создание и перегрузка конструктора:  
1. Имя конструктора совпадает с именем класса.

2. Конструктор не возвращает результат и для него тип результата не указывается вообще.

Во всем остальном конструктор напоминает обычный метод, за исключением лишь той разницы, что метод необходимо вызывать в явном виде, а конструктор вызывается автоматически и только при создании объекта. Конструктор может иметь аргументы, а может и не иметь. Конструктор можно перегружать.

class MyClass{

public:

int m.n;

myClass(){

m = 0;

n = 0;}

void show(){

cout << “m=” << m << endl;

cout << “n=” << n << endl;}};

int main(){

MyClass obj;

obj.show();

return 0;}

Правила перегрузки конструктора такие же как и правила перегрузки методов и функций. Наличие перегруженного конструктора существенно повышает гибкость программного кода.

Конструктор для создания нового объекта на основе уже существующего объекта:

Аргументом конструктора является объект того же класса, в котором описан конструктор.

MyClass(MyClass obj){

m = obj.m;

n = obj.n;}

Конструкторы незаменимы с практической точки зрения, их применение позволяет легко и быстро решать сложные задачи.

1. **Наследование классов и типы наследования.**

Одним из фундаментальных механизмов, обеспечивающих уникальную гибкость и продуктивность программ, написанных на С++, является наследование. Наследование позволяет одним объектам получать свойства других объектов.

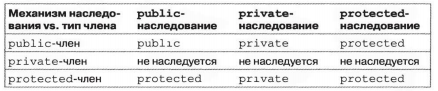
В С++ классы могут наследовать друг друга. Это значит, что класс-наследник(производный класс) получает свойства своего родительского(базового) класса. Кроме этого, производный класс может добавлять к свойствам, полученным от базового класса, свои собственные. Производный класса также может быть базовым по отношению к другому классу.

При наследовании классов производный класс фактически создается не на пустом месте, а на основе базового класса, получая в наследство от своего родителя поля и методы.

Какие поля и методы наследуются, а какие нет, определяются их доступностью. Они могут быть открытыми (public), закрытыми (private) или защищенными (protected). Защищенные члены класса фактически являются закрытыми, но от private они отличаются способом наследования.

Public-член базового класса наследуется производным классом при любом механизме наследования, но в производном классе унаследованный член может иметь разный уровень доступа. При public-наследовании он останется public-членом в производном классе.

При private-наследовании public-член базового класса становится private-членом производного класса.



Создание на основе базового класса производного класса:

сlass производный\_класс: тип\_наследования базовый класс{

//программный код производного класса

}список\_объектов;

Если производный класс B создается на основе базового класса А через открытое наследование (public-наследование):

сlass A{

//Программный код класса А

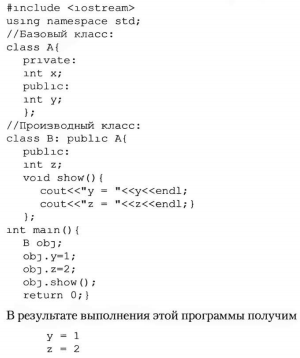
}список\_объектов;

//…

сlass B: public A{

//программный код класса B

}список\_объектов;



1. **Работа с файлами в С++. Описание файловой переменной. Способ доступа к файлам и их содержимому.**

Для работы с файлами необходимо подключить заголовочный файл <fstream>. В <fstream> определены несколько классов и подключены заголовочные файлы <ifstream> — файловый ввод и <ofstream> — файловый вывод.

Файловый ввод/вывод аналогичен стандартному вводу/выводу, единственное отличие – это то, что ввод/вывод выполнятся не на экран, а в файл. Если ввод/вывод на стандартные устройства выполняется с помощью объектов cin и cout, то для организации файлового ввода/вывода достаточно создать собственные объекты, которые можно использовать аналогично операторам cin и cout.

//запись  
ofstream fout;  
fout.open("cppstudio.txt");  
fout << "Работа с файлами в С++"; // запись строки в файл  
fout.close(); // закрываем файл  
//чтение  
char buff[50];  
ifstream fin("cppstudio.txt");   
if (!fin.is\_open()) // если файл не открыт  
cout << "Файл не может быть открыт!\n"; // сообщить об этом  
else  
{  
fin >> buff;  
cout << buff << endl;  
}  
ofstream fout("cppstudio.txt", ios\_base::app)  
ios\_base::in открыть файл для чтения  
ios\_base::out открыть файл для записи  
ios\_base::ate при открытии переместить указатель в конец файла  
ios\_base::app открыть файл для записи в конец файла  
ios\_base::trunc удалить содержимое файла, если он существует  
ios\_base::binary открытие файла в двоичном режиме

1. **Механизм ввода\вывода в С++. Потоки в С++. Связь потока с файлами. Чтение и запись текстовых файлов.**

В С++ используется механизм потокового ввода/вывода. Поток - механизм преобразования значений различного типа в последовательность символов (вывод) и наоборот (ввод), в значение переменной.

cout - стандартный поток вывода (экран)  
cerr - стандартный поток вывода ошибок (экран)  
cin - стандартный поток ввода (клавиатура)  
В поток ошибок и вывода можно писать, используя оператор <<  
Из потока ввода можно читать, используя оператор >>  
Так же, существуют различные манипуляторы. Используются они так:  
cout<<setw(4)<<setfill('#')<<left<<12;  
showbase   
noshowbase инициирует отображение основания системы счисления  
showpos   
noshowpos инициирует явное отображение знака (+) для положительных значений  
uppercase   
nouppercase инициирует преобразование выводимых символов в верхний регистр  
showpoint  
noshowpoint инициирует отображение десятичной точки при выводе вещественных чисел  
skipws  
noskipws инициирует пропуск пробельных символов при вводе  
left инициирует левое выравнивание, заполнение справа  
right инициирует правое выравнивание, заполнение слева  
internal инициирует внутреннее заполнение (между значением и знаком или основанием системы счисления)  
scientific инициирует вывод вещественных значений в научном формате (d.ddddddddE+dd)  
fixed инициирует вывод вещественных значений в фиксированном формате (d.dddddd)  
setbase (int base) изменяет систему счисления (10, 8, 16)  
dec инициирует вывод целочисленных значений в десятичной системе счисления  
oct инициирует вывод целочисленных значений в восьмеричной системе счисления  
hex инициирует вывод целочисленных значений в шестнадцатеричной системе счисления  
setfill (char n) задает заполняющий символ  
setprecision (int n) изменяет точность выводимых значений  
setw (int n) задает ширину поля вывода

Но самые важные -   
endl; // заканчивается строка, т.е. вставляется символ '\n'  
ends; // заканчивается строка записью ' '  
flush; // очистка буфера потока

Так как синтаксис работы с потоками ввода/вывода идентичен файловому, тот же код можно использовать и при работе с файлами

#include <fstream>  
using namespace std;  
void main()  
{  
ifstream inputfile("z:\data.txt"); // создается поток ввода из файла  
inputfile>>x; // все то же самое что и для cin  
ofstream outputfile("z:\res.txt"); // создается поток вывода в файл  
outputfile<<x; // все то же самое что и для cout  
}

1. **Абстрактные классы.**

Иногда возникает необходимость определить класс, который не предполагает создания конкретных объектов. Например, класс фигуры. В реальности есть конкретные фигуры: квадрат, прямоугольник и тд.   
Абстрактные классы - это классы, которые содержат или наследуют без переопределения хотя бы одну чистую виртуальную функцию. Абстрактный класс определяет интерфейс для переопределения производными классами. 

class Figure   
{   
public:   
virtual double getSquare() = 0;   
virtual double getPerimeter() = 0;   
virtual void showFigureType() = 0;   
};   
class Rectangle : public Figure   
{   
private:   
double width;   
double height;   
public:   
Rectangle(double w, double h) : width(w), height(h)   
{   
}   
double getSquare() override   
{   
return width \* height;   
}   
double getPerimeter() override   
{   
return width \* 2 + height \* 2;   
}   
void showFigureType()   
{   
std::cout << "Rectangle" << std::endl;   
}   
};

1. **Переопределение методов и виртуальные функции.**

Полиморфизм времени исполнения обеспечивается за счет использования производных классов и виртуальных функций. Виртуальная функция — это функция, объявленная с ключевым словом virtual в базовом классе и переопределенная в одном или в нескольких производных классах. Виртуальные функции являются особыми функциями, потому что при вызове объекта производ­ного класса с помощью указателя или ссылки на него С++ определяет во время исполнения про­граммы, какую функцию вызвать, основываясь на типе объекта. Для разных объектов вызываются разные версии одной и той же виртуальной функции. Класс, содержащий одну или более вир­туальных функций, называется полиморфным классом

Переопределение означает, что вы создали иерархию классов, у которой в базовом классе есть виртуальная функция и вы можете переопределить её в производном классе

class Base {  
public:  
virtual void prepare() { cout << "Base";}  
};  
class Derived : public Base {  
public:  
 virtual void prepare() { cout << "Derived";}  
};  
int main(int argc, char \*argv[])  
{  
 QCoreApplication a(argc, argv);  
 Base \*b=new Derived();  
 b->prepare();  
 return a.exec();  
}

Переопределение относится именно к виртуальным функциям.

1. **Создание и перегрузка конструктора.**

Конструктор - это метод класса, выполняемый при создании объекта данного класса. Он не имеет типа возвращаемого значения и должен называться так же, как и сам класс.

Конструктор должен иметь спецификатор доступа public.

class Test{  
public:  
 Test(){}  
};

Конструктор нельзя вызвать.

Можно переопределить конструктор параметрами, так же как и функцию

1. **Динамическое выделение памяти под объекты и высвобождение её.**

Для динамического выделения памяти под объекты класса используют оператор new.

Сначала объявляется указатель на объект соответствующего класса, после чего в формате указатель = new класс соответствующей командой под объект выделяется место в памяти и адрес передается указателю. Если при создании объекта конструктору необходимо передать аргументы, они указываются в круглых скобках после имени класса. Для удаления объекта из памяти используется команда delete указатель, где указатель является указателем на объект.

Конструктор – метод, вызываемый при создании объекта, деструктор вызывается при удалении объекта из памяти. Динамические массивы объектов создаются аналогично динамическим массивам базовых типов.

class MyClass {

public:

double x;

void show(){

cout << ”x=” << x << endl;}

MyClass (double z){

x = z;

cout << “Object with x = “ << x << “has been created!\n”;}

MyClass(){

x = 0;

cout << “Object with x = “ << x << “has been created!\n”;}

~MyClass(){

cout << “Object with x = “ << x << “has been deleted!\n”;}};

int main(){

MyClass \*p;

p = new MyClass;

p -> show();

delete p;

p = new MyClass(1);

p -> show();

delete p;

return 0;}

Для элементарных типов данных:

int \*p\_darr = new int[num];

Для объектов:

Test \*\*objPtr = new Test\*[COUNT\_OBJ];

for (int i = 0; i < COUNT\_OBJ; ++i) objPtr[i] = new Test(i + 5);

Для освобождения используется оператор delete

delete p\_darr;

Матрица:

int \*\*a = new int\* [n]; // Создаем массив указателей

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

{

a[i] = new int [n]; // Создаем элементы

}