Ответы на вопросы экзамена по курсу «Языки программирования» 09.01.2016

В ответах курсивом выделены необязательные пояснения, которые можно опустить (особенно на экзамене)

Вариант 2

Задача 2-1

Объясните, что означает понятие «множественное наследование» в языке C++. Объясните, какие проблемы возникают при реализации виртуальных функций при множественном наследовании (не перечисляйте общих для множественного наследования проблем, а ограничьтесь только специфическими проблемами реализации виртуальных методов при множественном наследовании)?

Ответ

«Множественное наследование» в языке C++ означает одновременное наследование от нескольких базовых классов (среди которых не должно быть одинаковых).
Нужно привести хотя бы один пример записи хотя бы двух баз, а затем сразу приступить к объяснению проблем. Разговоры про конфликты имен, однозначность доступа, ромбовидное наследование и т. п. НЕ ИМЕЮТ ОТНОШЕНИЯ к поставленному вопросу — какие проблемы возникают при реализации виртуальных методов при множественном наследовании. На лекциях обсуждалась по-крайней мере, одна проблема: организация таблицы виртуальных методов при множественном наследовании.
Проблема состоит в том, что указатель this при множественном наследовании должен корректироваться в зависимости от того, какие именно функции (точнее, методы какой именно базы) вызываются.
Пусть имеется следующее наслевование:

```
class Base1 {… void f();...}; class Base2 {...void g();...};
class D: public Base1, public Base2 { … void h(); …};
Base1 b1; Base2 b2; D d;
```

Рассмотрим вызов `d.f()` – указатель this есть &d. Однако при вызове d.g() указатель this уже не может быть равен &d, потому что g – это метод класса Base2, а &d указывает не на Base2, а на Base1 (именно поэтому в первом случае &d проходит). Поэтому компилятор в случае вызова d.g() корректирует указатель this, передаваемый в метод g() , увеличивая &d на необходимое число байтов — размер Base1 с учетом выравнивания и т. п.
Это можно сделать во время трансляции, так как связывание невиртуальных методов — статическое. Однако допустим, что и f(), и g(), и h() - виртуальные методы, причем и f(), и g() - замещены в классе D.

```
class Base1 {… virtual void f();...};
class Base2 {...virtual void g();...};
class D: public Base1, public Base2 {
 … virtual void h();
   virtual void f();
   virtual void g();
};
Base1 * pb1; Base2 * pb2; D * pd;
Base1 b1; Base2 b2; D d;
```

Рассмотрим теперь вызов pb1→f(). Какая функция будет вызвана, и какой указатель this

должен быть передан?
Если бы pb1 был инициализирован как pb1=&b1, то была бы вызвана функция Base1::f().
Если же pb1 был инициализирован как pb1=&d, то была бы вызвана функция D::f().
Но в любом случае указатель this должен быть передан равным pb1.
Ситуация меняется в случае вызова pb2→g() (то есть для второй базы в списке наследования).
Если бы pb2 был инициализирован как pb2=&b2, то была бы вызвана функция Base2::g().
Если бы pb2 был инициализирован как pb2=&d, то была бы вызвана функция D::g().
И снова указатель this должен быть равен pb2. Однако представим, что функция g() НЕ ЗАМЕЩЕНА в классе D (то есть она объявлена в классе Base2, но заместителя в классе D нет). Тогда в ОБОИХ вариантах инициализации указатель this должен указывать на объект класса Base2. Поэтому в первом случае передается this=pb2. А во втором случае (pb2=&d), нужно скорректировать &d на величину Base1 с учетом выравнивания и т. п. так, чтобы this указывал на часть d от Base2. Как компилятор догадается, в каком случае нужно корректировать? В этом и состоит одна из проблем.
На этом можно и закончить — ведь проблема объяснена. Но можно (хотя и необязательно) добавить и про возможные решения проблемы. Стандартный подход состоит в том, что таблица виртуальных методов содержит в каждой строке дополнительное поле — смещение, которое нужно добавить к указателю this при вызове метода, соответствующего этой строке. Кроме этого, вместо одной таблицы виртуальных методов для производного класса D приходится делать ДВЕ таблицы: первая содержит ссылки на методы классов Base1 и D (две строки — для f() и h(), дополнительные смещения для всех нулевые), вторая — ссылки на методы Base2 и D (две строки — для g() и h(), причем дополнительное смещение для h() - нулевое, а для g() - нулевое, если метод замещен в D, и ненулевое, если метод НЕ замещен в D).

Задача 1-2

Объясните, что означают термины «семантика возобновления» и «семантика завершения». Приведите пример языка, в котором реализована семантика возобновления. В каких случаях семантика возобновления может оказаться предпочтительнее, чем семантика завершения?

Ответ

Семантика возобновления: после обработки исключения управление может вернуться непосредственно в точку, где возникло исключение (варианты: на следующий оператор или на любой оператор из того же блока). Пример языка: Visual Basic.
Семантика завершения: после возникновения исключения блок, в котором оно возникло, обязательно завершается. Обработка исключения происходит в блоках, вызвавших блок с исключением.
В средних и крупных проектах семантика завершения показала свое превосходство, однако, в небольших и простых проектах в ряде случаев семантика возобновления может быть предпочтительнее, чем завершение. *Примеров можно привести много. Например:* выделение ресурса по принципу ленивой инициализации, ввод информации от пользователя с последующей валидацией. В случае ошибки возможно повторить попытку ввода (вернуться в точку, где возникло исключение), задать значение по умолчанию (на следующий оператор), завершить исполнение или испробовать другие варианты ввода (на другой оператор в блоке).

Задача 1-3

Дайте определение языковой конструкции «интерфейс». Что означает термин «интеграция интерфейсов в язык программирования»? Приведите два различных примера интерфейсов, интегрированных с языком C#.

Ответ

Интерфейс можно рассматривать как абстрактный класс, доведенный до «абсолюта». Интерфейс состоит только из публичных абстрактных методов. Поскольку реализации методов нет (равно как нет и невиртуальных методов), то интерфейс не имеет нестатических членов (статические члены, например, константы допустимы). В объявлении интерфейса присутствуют только сигнатуры методов (а также свойств в языках, где есть это понятие), и, возможно, статических членов и вложенных интерфейсов. Интерфейс представляет собой «чистый» контракт. Производный класс, наследуя интерфейс, «подписывается» под контрактом. Производный класс, наследующий интерфейс и замещающий все его методы, называют реализацией интерфейса.
Интеграция интерфейсов в язык означает, что язык (компилятор) поддерживает ряд стандартных интерфейсов, позволяющих интегрировать семантику языковых конструкций и пользовательские классы, реализующие эти интерфейсы. Например, в языке C# стандартный интерфейс `IEnumerable` используется в цикле `foreach` для прохода по коллекциям. Если пользовательский класс реализует этот интерфейс, то он тоже может использоваться в цикле `foreach`, как это можно делать со стандартными массивами и коллекциями.
Другой пример интегрированного интерфейса в языке C# - `IDisposable` – интерфейс для работы с явно уничтожаемыми объектами:

```
interface IDisposable
{
          void Dispose();
}
```

Явно уничтожаемые объекты должны реализовать этот интерфейс. Процедура `Dispose()` должна освобождать ресурсы, занятые объектом (кроме памяти, поскольку это - забота сборщика мусора).

Задача 1-4

Перечислите модули, которые могут быть единицами компиляции (ЕК) в языке Ада. Какие ЕК называются первичными, а какие — вторичными? В чем состоит отличие первичных и вторичных ЕК в этом языке?

Ответ

ЕК в Аде — спецификации пакетов, в том числе и родовых - первичные модули, тела пакетов, тела подпрограмм (в том числе и родовых) - вторичные модули. Отличие первичных от вторичных состоит в том, что первичные модули связаны только односторонней связью — то есть клиенты первичных модулей их импортируют (предложение WITH список_имен_модулей), а сами первичные модули о клиентах ничего не знают. А вторичные модули могут (не всегда, но могут!) иметь двусторонние связи — в клиенте (например, объемлющем теле пакета P) вместо модуля-тела вложенного пакета PP вставляется заглушка (**PACKAGE BODY PP IS SEPARATE;**), а сам модуль связывается с клиентом с помощью спецификации **SEPARATE (P)** PACKAGE BODY PP IS ….. END PP;

Задача 1-5

Что будет напечатано в результате работы следующей программы на Javascript? Считаем, что метод `console.log` печатает значение аргумента на экран.

```
u = (function f() { var k = 0;  return function(x) { return x+k++;} }) ()
for (i = 0; i < 5; i++)  console.log(u(i))
for (i--; i >=0; i--)  console.log(u(i))
```

Ответ

Здесь при возврате из функции f() происходит замыкание локальной переменной k. Значение, запомненное в u – это функция от одной переменной и замыканием k. При каждом вызове этой функции r увеличивается на 1.
При первом цикле (5 итераций) увеличивается и k, и i, поэтому значения будут 0, 2, 4, 6, 8. После последней итерации k=4.
При втором цикле (тоже 5 итераций) k увеличивается, зато i уменьшается, поэтому значение будет все время 9 (при первой итерации второго цикла `i = 4, a k++ = 5`).
9 9 9 9 9

Задача 1-6

Переписать программу из задачи №5 на языке C++ с использованием механизма лямбда-функций так, чтобы она выдавала в стандартный вывод те же самые значения.

Ответ

```
int k = 0;

int main()
{
	auto u = [](){ return [](int i){return i+k++; };}();

	int i;
	for (i = 0; i < 5; i++)  std::cout << u(i) << " ";
	for (i--; i >= 0; i--)  std::cout << u(i) << " ";

	return 0;
}
```

Задача 1-7

Рассмотрим фрагмент программы на языке Java, содержащий определение класса C:

```
package test;
abstract class C { public abstract int f(); }
class D { public void foo ()
	{ C c = new C { public int f() { return 1; }};}
}
```

При трансляции этого файла выдается ошибка. Объясните, из-за чего она выдается и что нужно добавить в программу, не меняя ничего в классе C, чтобы данный файл транслировался без ошибок? Можно только добавлять конструкции в программу, но не исключать.

Ответ

В классе D создается объект некоторого локального анонимного класса — наследника класса C. Так как класс C — абстрактный, то при описании наследника нужно заместить абстрактный метод. Поскольку при создании объекта ВСЕГДА должен вызываться

конструктор (единственное исключение — клонирование объекта, но это здесь не при чем), то нужно указать, какой именно конструктор вызывается — у локального анонимного класса из примера есть только сгенерированный конструктор умолчания без параметров. Так что надо просто вставить пустые скобки, сигнализирующие вызов конструктора умолчания. Почему-то это надо делать сразу после имени базового класса до фигурных скобок:

```
C c = new C ()){ public int f() { return 1; }};
```

Задача 1-8

При ответе на эту задачу следует иметь ввиду то, что в Java приватные функции не замещаются в производных классах, так как они невидимы в них (см. также ответ на вопрос №8 из первого варианта). Поэтому приватная функция f4не будет замещаться в производных классах (неважно, замещается она публичной или приватной функцией — все равно не видна!). Все осталбные функции видны внутри пакета и впроизводном классе, поэтому они замещаеются.

Что будет напечатано в результате работы следующей программы на Java?

```
package P1;
public class T {
      public static void main (String [] args) { new CC().show(); }
}
public abstract class AC {
      public static final void print(String s) { System.out.println(s);}
      public void f1() { print("AC.f1");}
      void f2() { print("AC.f2");}
      protected void f3() { print("AC.f3");}
      private void f4() { print("AC.f4");}
      public final void show() { f1();f2();f3();f4();}
}
class CC extends AC {
      public void f1() { print("CC.f1");}
      public void f2() { print("CC.f2");}
      public void f3() { print("CC.f3");}
      public void f4() { print("CC.f4");}
}
```

Ответ

CC.f1

CC.f2

CC.f3

AC.f4