*1. Объясните причину ошибок компиляции:*

*---cut---*

*typedef int \* int\_p;*

*const int \* x = 0;*

*const int\_p y = x; //Ошибка приведения типов. Если const int\_p y = (int\_p)x; то компилируется*

*#define long\_p long \**

*const long \* x = 0; // Повторное определение константы х*

*const long\_p y = x;*

*---cut---*

*2. Выберите POD-типы из перечисленных, аргументируя своё решение:*

*a) int - простой тип*

*b) unsigned long \* - простой тип*

*c) struct C {}; - структура (без пользовательских конструкторов и операторов = ) в стиле Си*

*d) void \* - указатель является POD типом*

*e) enum E {}; - перечисление является POD типом*

*f) enum F { f = 1 }; - перечисление является POD типом*

*g) C & - ссылка на обьект не является POD типом*

*h) class H {}; - класс (без пользовательских конструкторов и операторов = )*

*i) struct I { I(); }; - Пользовательский конструктор*

*j) int[ 13 ] - массив простых типов*

*k) struct K { ~K(); }; - Пользовательский деструктор*

*l) struct L { virtual ~L(); }; - Виртуальный член + пользовательский деструктор*

*m) K[ 13 ] – массив не POD объектов*

*n) union N { int i; float f; }; - объединение (без пользовательских конструкторов и операторов =)*

*o) L[ 13 ] – массив не POD объектов*

*p) struct P { static int x; } - структура в стиле Си*

*q) struct Q { int & i; } - ссылка*

*r) struct R { protected: int x; } – член объявлен как protected*

*s) struct S : K {}; - Наследование*

*t) struct T : I {}; - Наследование*

*u) struct U { static L x; } – статический не POD член*

*v) struct V : H {}; - Наследование*

*w) struct W { private: int x; }– член объявлен как private*

*x) L \* - ссылка на не POD член*

POD типами являются: a, b, c, d, e, f, h, j, n, p

*3. В приведённом коде создаются и удаляются объекты типа S, конструктор и деструктор которого выводят "create name" и "destroy name". В каком порядке будут вызваны соответствующие конструкторы и деструкторы этих объектов? Дайте развёрнутый ответ, используя понятия "область видимости", "статическое/автоматическое/динамическое хранение", "время жизни"...*

*---cut---*

*#include <string>*

*#include <iostream>*

*using namespace std;*

*struct S {*

*string name\_;*

*S( string const & name ) : name\_(name) { cout << "create " << name << endl; }*

*static S zero;*

*~S() { cout << "destroy " << name\_ << endl; }*

*};*

*S one( "one" );*

*void f() {*

*S two( "two" );*

*static S three( "three" );*

*}*

*S S::zero( "zero" );*

*void g() {*

*S four( "four" );*

*static S five( "five" );*

*}*

*S const six( "six" );*

*int main() {*

*f();*

*S \* seven = new S( "seven" );*

*( S ( "eight" ) );*

*new S( "nine" );*

*S ten( "ten" );*

*}*

*S eleven( "eleven" );*

*---cut---*

*Вывод программы:*

*create one*

*create zero*

*create six*

*create eleven*

*create two*

*create three*

*destroy two*

*create seven*

*create eight*

*destroy eight*

*create nine*

*create ten*

*destroy ten*

*destroy eleven*

*destroy six*

*destroy zero*

*destroy one*

Первыми создаются объекты в глобальной области видимости *(one, zero, six, eleven)* в порядке вызова их конструкторов в программе. Эти объекты «живут» на протяжении всей работы программы и уничтожаются последними в порядке, обратном их созданию *(eleven, six, zero, one)*.

В самом начале функции *main()* происходит вызов функции *f()*, в котором создаются объекты *two* и *three*. *two* объявлен внутри функции и имеет локальную область видимости (автоматическое время жизни), вследствие чего, уничтожается сразу при завершении *f()*. *Three* объявлен как ***static,*** его область видимости становиться глобальной (статическое время жизни), уничтожается объект перед завершении программы, до уничтожения 4х глобальных объектов.

Вызова функции *g()*не происходи, *four* и *five* не создаются.

Объект *seven* создаётся следующим, имеет динамическое время жизни и не уничтожается, так как для него не был вызван *delete*.

*eight* – временный объект. «Живёт» до оператора; создан и сразу же уничтожен.

*nine* – ситуация аналогичная с *seven* с той лишь разницей что ссылка на созданный объект никуда не передаётся.

*ten* – локальный объект с автоматическим временем жизни, создаётся последним вызовом перед завершением функции *main()*. После завершения *main()* сразу уничтожается.

*4. Чем отличаются понятия Linkage(связывание) и Binding(связывание)?*

Эти термины относятся к разным областям языка.

*Linkage* – связывания имён сущностей (переменных, методов, классов и т.д) в программе. Бывает внутренним: связывание внутри одной единицы трансляции(файлы .cpp, .cc + заголовочные), и внешним: по всей остальной программе.

*Binding* – связь объектов и вызовов функций между собой. Бывает ранним: во время компиляции и поздним: во время выполнения.