15 популярных вопросов с IT-собеседований по языку C++

Источник: https://proglib.io/p/tricky-challenges-cpp/

1 Что получим на выходе?

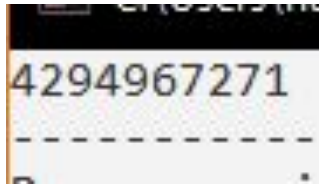
```
#include<iostream>
using namespace std;
int f();
int x = 9;

int main()
{    f();
    cout << x;
}
 int f()
{
   ::x = 8;
}
```

Используя операцию «::», получаем доступ к переменной x и меняем ее значение x = 8;

2 Что на выходе и почему?

```
int main(int argc, char **argv)
{
    std::cout << 25u - 50;
    return 0;
}
```



Ответ равен не -25, а равен 4294967271, почему?

Существует иерархия преобразования типов: long double, double, float, unsigned long int, long int, unsigned int, int.

Все операнды в выражении преобразуются к верхнему типу.

В выражении «25u (unsigned int) - 50 (int)» 50 будет преобразовано в безнаковое целое (в число 50u).

И полученный результат -25 тоже будет типа unsigned int, а именно будет равен 4294967271 (для 32-х битной системы).

3 Разберем код:

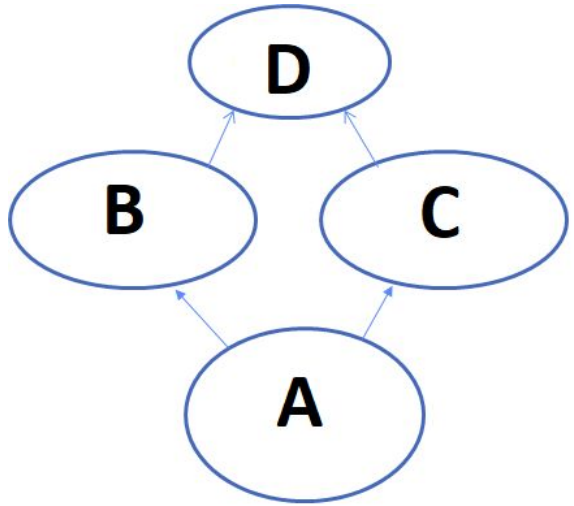
```
#include<iostream>
 class D  {
   public:
    void foo() { std::cout << "Fooooooo" << std::endl;   }
};

class C:  public D {};

class B:  public D {};

class A: public B, public C {};

int main() {
    A a;
    a.foo();
}
```



Имеем «ромбовидное» наследование.

Класс A получает две копии класса D: один из класса B и один из класса C.

Чтобы исправить, нужно изменить объявления классов C и B (virtual).

```
[Error] request for member 'foo' is ambiguous
```

«Использование foo неоднозначно»

3 Когда используется виртуальное наследование?

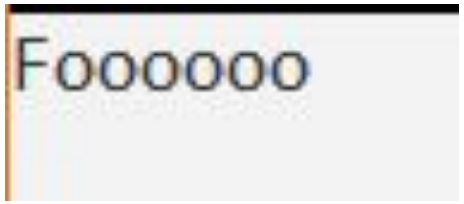
Исправим:

```
#include<iostream>
 class D {
public:
   void foo() { std::cout << "Fooooooo" << std::endl;   }
};

class C:  virtual public D {};

class B:  virtual public D {};
class A: public B, public C {};

int main() {
   A a;
   a.foo();
}
```



4 Что означает модификатор *virtual*?

В C++ виртуальные функции позволяют поддерживать полиморфизм – одну из ключевых составляющих ООП. С его помощью в классах-потомках можно переопределять функции класса-родителя.

Без виртуальной функции мы получаем «раннее связывание», а с ней – «позднее связывание». То есть, какая реализация метода используется, определяется непосредственно во время выполнения программы и основывается на типе объекта с указателем на объект, из которого он построен.

5 Пример использования виртуальной функции

```
class Animal
{    public:
        void eat() { std::cout << "I'm eating generic food."; }
};

class Cat : public Animal
{    public:
        void eat() { std::cout << "I'm eating a rat."; }
};
main()
{  Animal *animal = new Animal;
    Cat *cat = new Cat;
    animal->eat();      // Outputs: "I'm eating generic food."
    cat->eat();         // Outputs: "I'm eating a rat.
}
```

5 Пример использования виртуальной функции

Добавим функцию:

void func(Animal *xyz) { xyz->eat(); }

```
class Animal
{    public:
       void eat() { std::cout << "I'm eating generic food."; }
};
 class Cat : public Animal
{    public:
       void eat() { std::cout << "I'm eating a rat."; }
};
main()
{  Animal *animal = new Animal;
    Cat *cat = new Cat;
// вызовем eat() с использованием fun
 func(animal);  // Outputs: "I'm eating generic food."
 func(cat);       // Outputs: "I'm eating generic food."
}
```

```
I'm eating generic food.I'm eating generic food.
```

5 Пример использования виртуальной функции

Исправим:

```
void func(Animal *xyz) { xyz->eat();
class Animal
{    public:
       void virtual eat() { std::cout << "I'm eating generic food."; }
};
 class Cat : public Animal
{    public:
       void eat() { std::cout << "I'm eating a rat."; }
};
main()
{  Animal *animal = new Animal;
    Cat *cat = new Cat;
// вызовем с использованием fun
 func(animal);
 func(cat);
}
```

```
I'm eating generic food.I'm eating a rat.
--------------------------------
```

6 Существует ли различие между классом и структурой?

Единственное различие между классом и структурой – это модификаторы доступа.

Элементы структуры являются общедоступными по умолчанию – **public**, а элементы класса – **private**.

Рекомендуется использовать классы, когда вам нужен объект с методами, а иначе (простой объект) – структуры.

7 Что не так с кодом?

```
class A
{  public:
    A() {}
    ~A(){  cout<<"destruct A"<<endl;}
};
 class B: public A
{ public:
   B():A(){}
   ~B(){   cout<<"destruct B"<<endl;  }
};
 int main(void)
{   A* a = new B();
    delete a;
}
```

```
destructor A
```

Исправим:

```
class A
{  public:
    A() {}
 virtual ~A(){cout<<"destructor A"<<endl;}
};
 class B: public A
{ public:
   B():A(){}
   ~B(){cout<<"destructor B"<<endl;}
};
 int main(void)
{  A* a = new B();
   delete a;
}
```

```
destructor B
destructor A
```

7 Что не так с кодом?

Удаление объекта порожденного класса через указатель на базовый класс без виртуального деструктора (**virtual ~**) является неопределенным поведением (**undefined behavior**) согласно стандарту C++11 §5.3.5/3.

Вызов деструктора порожденного класса может работать, а может и нет, и нет никаких гарантий -> избегать такого.

8 Что такое класс хранения?

Класс, который определяет срок существования, компоновку и расположение переменных/функций в памяти.

В C ++ поддерживаются такие классы хранения:

- auto,
- static,
- register,
- extern,
- mutable.

Обратите внимание, что register устарел для C++11. Для C++17 он был удален и зарезервирован для будущего использования.

9 Как вызвать функцию C в программе на C++?

```
//C code
void func(int i)
{
    //code
}

void print(int i)
{
    //code
}
```

```
//C++ code
extern "C"{
void func(int i);
void print(int i);
}

void myfunc(int i)
{   func(i);
     print(i);
}
```

Соглашение о вызове:

в C и C++ параметры передаются по разному (используя регистры и стек, но каждый по-своему).

10 Что делает ключевое слово *const*?

Задает константность объекта, указателя, а также указывает, что данный метод сохраняет состояние объекта (не модифицирует члены класса).

Пример с неизменяемыми членами класса:

```
 class Foo
{
  private:
    int i;
  public:
    void func() const
    { i = 1;     // error C3490: 'i' cannot be modified because it is being
    }            //accessed through a const object
};
```

11 Виртуальный деструктор: что он собой представляет?

Во-первых, он объявляется как ***virtual***. Он нужен, чтобы с удалением указателя на какой-нибудь объект был вызван деструктор данного объекта.

Например, у нас есть 2 класса:

```
class base
{ public:
virtual
            ~base()
   { cout<<"destructor base\n";
   }
 };

class derived: public base
{ public:
  ~derived()
  { cout << "destructor derived\n";
  }
};
```

```
int main(){
base *p;  //указатель на base
p=new derived();
delete p;
return 0;
}
```

```
destructor base
```

```
destructor derived
destructor base
```

11 Виртуальный деструктор: что он собой представляет?

Без виртуального деструктора будет выполняться только вызов деструктора базового класса, а вызов производного деструктора - не будет. Получаем «утечку памяти».

Поэтому необходимо деструктор базового класса сделать виртуальным (добавить **virtual**), тогда освобождение памяти будет корректным.

12 Виртуальный конструктор: что он собой представляет?

Каверзный вопрос с IT-собеседований, который чаще всего задают именно после виртуальных деструкторов, дабы сбить кандидата с толку.

Конструктор ***не*** может быть виртуальным, поскольку в нем нет никакого смысла: при создании объектов нет такой неоднозначности, как при их удалении.

13. Сколько раз будет выполняться этот цикл?

```
unsigned char half_limit = 150;

for (unsigned char i = 0; i < 2 * half_limit; ++i)
  {   //что-то происходит;
  }
```

Еще один вопрос с подвохом с IT-собеседований.

Ответ был бы равен 300, если бы i был объявлен как int.

Но поскольку i объявлен как unsigned char, правильный ответ – зацикливание (бесконечный цикл).

Выражение 2 * half_limit будет повышаться до int (на основе правил преобразования C++) и получит значение 300. Но так как i – это unsigned char, он пересматривается по 8-битному значению, которое после достижения 255 будет переполняться, поэтому вернется к 0, и цикл будет продолжаться вечно.

14. Каков результат следующего кода?

```
#include<iostream>
class Base {
   virtual void method() {std::cout << "from Base" << std::endl;}
public:
   virtual ~Base() {method();}
   void baseMethod() {method();}
};

class A : public Base {
   void method() {std::cout << "from A" << std::endl;}
public:
   ~A() {method();}
};

int main(void) {
   Base* base = new A;
   base->baseMethod();
   delete base;
   return 0;
}
```

```
from A
from A
from Base
```

from A
from A
from Base

Здесь важно отметить порядок уничтожения классов и то, как метод класса Base возвращается к своей реализации после удаления A.

15. Что мы получим на выходе?

```
#include <iostream>

int main() {
    int a[] = {1, 2, 3, 4, 5, 6};
    std::cout << (1 + 3)[a] - a[0] + (a + 1)[2];
}
```

Ответ равен 8

- (1 + 3)[a] – то же, что и a[1 + 3] == 5
- a[0] == 1
- (a + 1)[2] – то же, что и a[3] == 4

Суть вопроса заключается в проверке арифметических знаний и понимании всей магии, которая происходит за квадратными скобками.