百题魔鬼训练营详解（笔试篇）

讲师介绍：易飞云 qq 1069619619

1 下列代码的运行结果（）

```
short i = 65537;

int j = i + 1;

printf(“i = % d, j = % d\n”, i, j);
```

A i=65537,j=65538

B i=1,j=2.

C i=-1,j=0

D i=1,j=65538

补充知识：负数是以补码的形式在计算机里面存储的

原码：一个整数，按照绝对值大小转换成的二进制数，称为原码。

如：00000000 00000000 00000000 00000101 是 5 的原码。

反码：将二进制数按位取反，所得的新二进制数称为原二进制数的反码。

取反操作指：原为 1，得 0；原为 0，得 1。（1 变 0; 0 变 1）

比如将 00000000 00000000 00000000 00000101 每一位取反得：11111111 11111111 11111111 11111010

称 11111111 11111111 11111111 11111010 是 00000000 00000000 00000000 00000101 的反码

反码是相互的。

补码：反码加 1 称为补码。

比如：00000000 00000000 00000000 00000101 的反码是：11111111 11111111 11111111 11111010。

那么，补码为：

11111111 11111111 11111111 11111010 + 1 = 11111111 11111111 11111111 11111011

所以-5 在计算机中的表达为：11111111 11111111 11111111 11111011

解析：

short 型为 2 个字节，16bit

65537 二进制 1 0000 0000 0000 0001 需要 17bit 表示，最高位溢出，因此 i=1

j=i+1=2

//为什么会不相等

```
int main()
{
	long long m = -65537;
```

```
        unsigned int i = 0XFFFEFFFF;
        if (m == i)
        {
                printf("yes\n");
        }
        else
        {
                printf("no\n");
        }
        return 0;
}

/*
1 万能方法： 转为二进制

补码 = 反码 + 1  补码的补码就是原码
原码： 一个整数 按照绝对值的大小转化为二进制数；
  0000 0000 0000 0001 0000 0000 0000 0001  原码
  1111 1111 1111 1110 1111 1111 1111 1110  反码
  1111 1111 1111 1110 1111 1111 1111 1111  补码  0XFFFE FFFF
  i = 0xffff      1  111 1111 1111 1111
  000 0000 0000 0000 反码
  000 0000 0000 0001 补码

  2 周期函数法；
*/
```

原则 1：数据存储与类型无关；先转化为二进制再说；

原则 2：看符号位；

原则 3：补码的补码就是原码

第一种方法叫做万能方法；

第二种方法叫做函数的周期性；

2 头文件已经正常包含，以下代码在 VS2019 上编译和运行结果是（）

```
#include <stdio.h>

class A {

public:

        void test() { printf("test A"); }

};

int main() {

        A* pA = NULL;

        pA->test();

        return 0;

}
```

A 编译出错　　B 程序运行奔溃　　C 输出"test A"　　D 输出乱码

因为对于非虚成员函数， C ++这门语言是静态绑定的。这也是 C ++语言和其它语言 Java, Python 的一个显著区别。以此下面的语句为例：

这语句的意图是：调用对象 pA 的 test 成员函数。如果这句话在 Java 或 Python 等动态绑定的语言之中，编译器生成的代码大概是：

找到 pA 的 test 成员函数，调用它。（注意，这里的找到是程序运行的时候才找的，这也是所谓动态绑定的含义：运行时才绑定这个函数名与其对应的实际代码。有些地方也称这种机制为迟绑定，晚绑定。）

但是对于 C++。为了保证程序的运行时效率， C ++的设计者认为凡是编译时能确定的事情，就不要拖到运行时再查找了。所以 C++的编译器看到这句话会这么干：

1：查找 pA 的类型，发现它有一个非虚的成员函数叫 test 。（编译器干的）

2：找到了，在这里生成一个函数调用，直接调 A:: test (pA)。

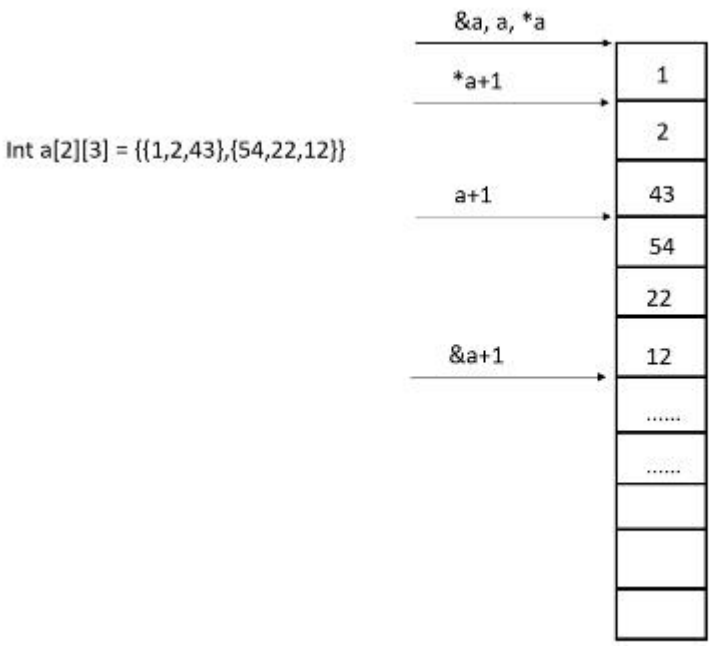
所以到了运行时，由于 test ()函数里面并没有任何需要解引用 pA 指针的代码，所以真实情况下也不会引发 segment fault。这里对成员函数的解析，和查找其对应的代码的工作都是在编译阶段完成而非运行时完成的，这就是所谓的静态绑定，也叫早绑定。

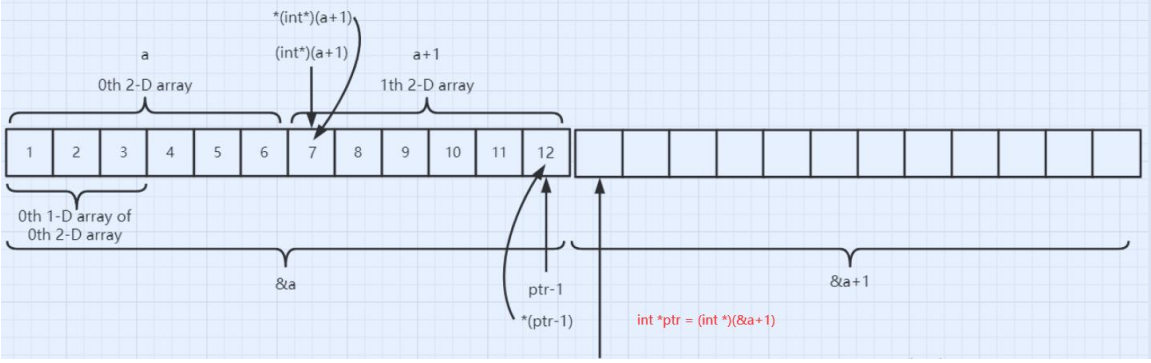
正确理解 C++的静态绑定可以理解一些特殊情况下 C++的行为。

3 求输出结果 ()

```
int main() {

        int a[2][2][3] = { {{1,2,3},{4,5,6}},{{7,8,9},{10,11,12}} };

        int* ptr = (int*)(&a + 1);

        printf(" %d %d", *(int*)(a + 1), *(ptr - 1));

    printf(" %d %d", *(int*)(*a + 1), *(int*)(*a + 2));


        return 0;

}
```

A 7 12 B 1 6 C 1 3 D 7 9





3.1 求输出结果 ()

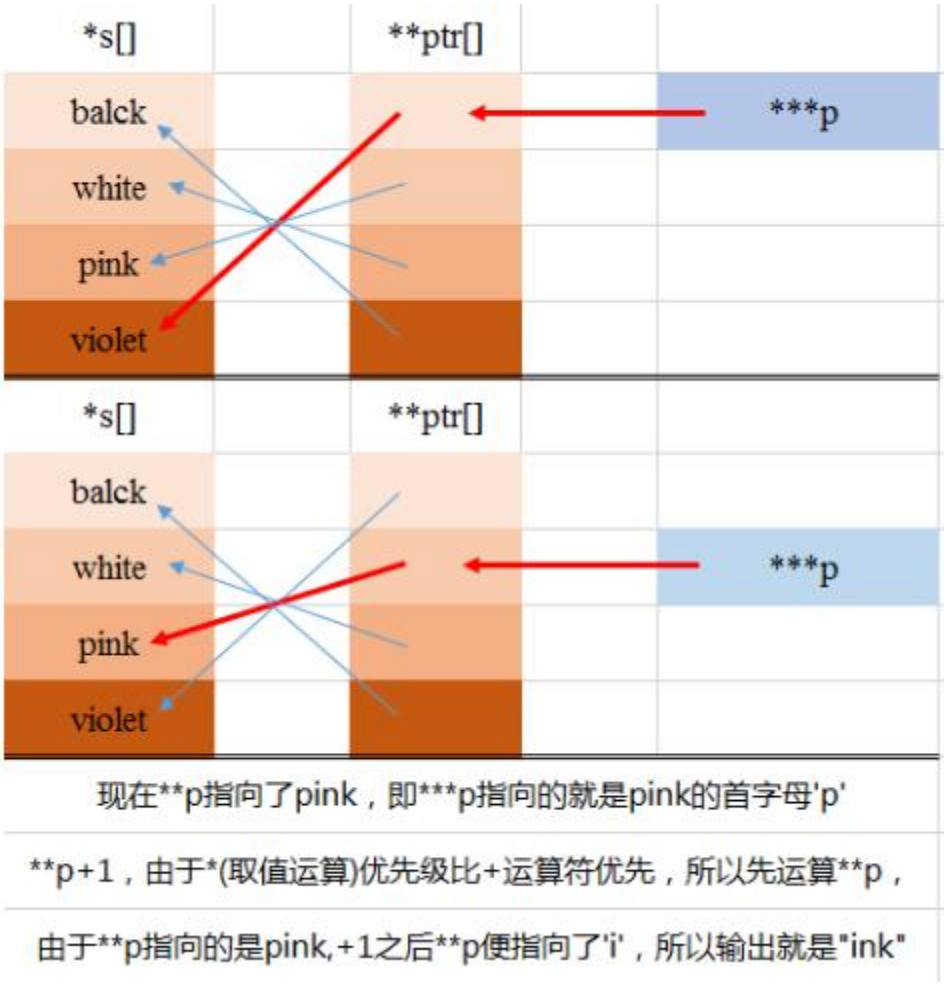
```
#include <stdio.h>

int main()

{

        int a[3][4] = { {1, 2, 3, 4},

                                {2, 3, 4, 5},

                                {3, 4, 5, 6} };


        int b[3][4] = { {10, 11, 12, 13},

                                {11, 12, 13, 14},

                                {12, 13, 14, 15} };


        int(*array[2])[4] = { a, b };//array[0]:   a        array[1]:   b


        int* p1[3] = { a[0], a[1], a[2] };

        int* p2[3] = { b[0], b[1], b[2] };

        int** pp[2] = { p1, p2 };

        int*** p = pp;


        printf("%d\n", (*(*p + 2))[1]);

        printf("%ld\n", array[1][2] - array[1][0]);

        return 0;
```

```
}
```

4 下述程序的输出是_____。

```
int main()
{
        static char* s[] = { "black", "white", "pink", "violet" };
        char** ptr[] = { s + 3, s + 2, s + 1, s }, *** p;
        p = ptr;
        ++p;
        printf("%s", **p + 1);
        return 0;
}
```

A ink B pink C white D hite



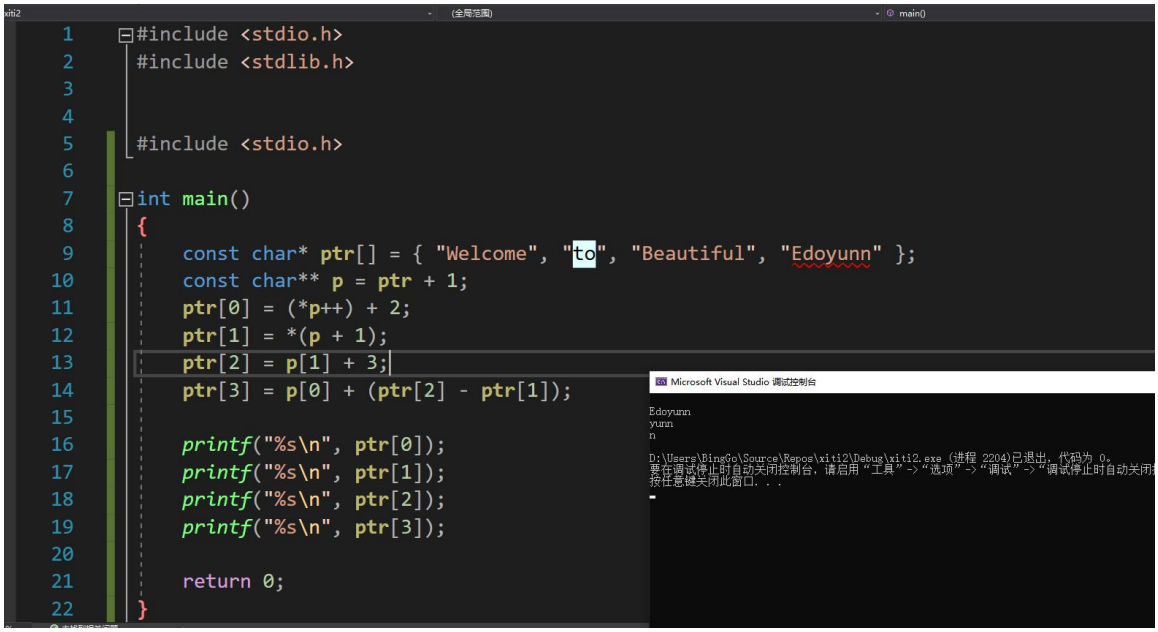
4.1 下述程序的输出是_____。

```
#include <stdio.h>


int main()

{

        const char* ptr[] = { "Welcome", "to", "Beautiful", "Edoyun" };

        const char** p = ptr + 1;

        ptr[0] = (*p++) + 2;

        ptr[1] = *(p + 1);

        ptr[2] = p[1] + 3;

        ptr[3] = p[0] + (ptr[2] - ptr[1]);
```

```
        printf("%s\n", ptr[0]);
        printf("%s\n", ptr[1]);
        printf("%s\n", ptr[2]);
        printf("%s\n", ptr[3]);

        return 0;
}
```

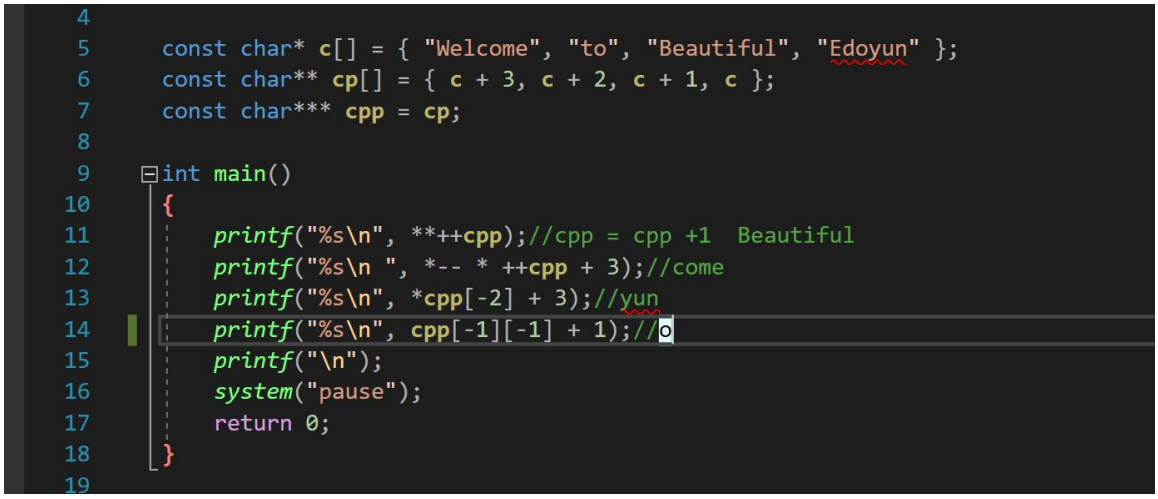
以下程序的输出是：

```
#include <stdio.h>

const char* c[] = { "Welcome", "to", "Beautiful", "Edoyun" };
const char** cp[] = { c + 3, c + 2, c + 1, c };
const char*** cpp = cp;

int main()
{
        printf("%s\n", **++cpp);
        printf("%s\n ", *-- * ++cpp + 3);
        printf("%s\n", *cpp[-2] + 3);
        printf("%s\n", cpp[-1][-1] + 1);
        printf("\n");

        return 0;
}
```



```c
#include <stdio.h>
#include <stdlib.h>


#include <stdio.h>

int main()
{
    const char* ptr[] = { "Welcome", "to", "Beautiful", "Edoyunn" };
    const char** p = ptr + 1;
    ptr[0] = (*p++) + 2;
    ptr[1] = *(p + 1);
    ptr[2] = p[1] + 3;
    ptr[3] = p[0] + (ptr[2] - ptr[1]);

    printf("%s\n", ptr[0]);
    printf("%s\n", ptr[1]);
    printf("%s\n", ptr[2]);
    printf("%s\n", ptr[3]);

    return 0;
}
```

```
Microsoft Visual Studio 调试控制台

Edoyunn
yunn
n

D:\Users\BingGo\Source\Repos\xiti2\Debug\xiti2.exe (进程 2204)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭
按任意键关闭此窗口. . .
```



```c

    const char* c[] = { "Welcome", "to", "Beautiful", "Edoyun" };
    const char** cp[] = { c + 3, c + 2, c + 1, c };
    const char*** cpp = cp;

int main()
{
    printf("%s\n", **++cpp);//cpp = cpp +1  Beautiful
    printf("%s\n ", *-- * ++cpp + 3);//come
    printf("%s\n", *cpp[-2] + 3);//yun
    printf("%s\n", cpp[-1][-1] + 1);//o
    printf("\n");
    system("pause");
    return 0;
}

```

5 下列关于对象初始化的叙述中，正确的是:（）

A 定义对象的时候不能对对象进行初始化

B 定义对象之后可以显式地调用构造函数进行初始化

C 定义对象时将自动调用构造函数进行初始化

D 在一个类中必须显式地定义构造函数实现初始化

我的答案是 B

A：String s = "hahaha";定义时初始化

B:String s = new String("haha");显式地调用构造函数进行初始化

C:定义对象不会调用构造函数 实例化时调用构造；

D：不用必须显示的定义构造函数

6 下面程序的输出结果是？

```
#include <iostream>
using namespace std;
class animal
{
protected:
	int age;
public:
	virtual void print_age(void) = 0;
};
class dog : public animal
{
public:
	dog() { this->age = 2; }
	~dog() {}
	virtual void print_age(void)
	{
		cout << "wang. my age=" << this->age << endl;
	}
};
```

```
class cat :public animal
{
public:
        cat() { this->age = 1; }
        ~cat() {}
        virtual void print_age(void) { cout << " Miao,my age= " << this->age << endl; }
};
int main(void)
{
        cat kitty;
        dog jd;
        animal* pa;
        int* p = (int*)(&kitty);
        int* q = (int*)(&jd);
        p[1] = q[1];
        pa = &kitty;
        pa->print_age();
        system("pause");
        return 0;
}
```

A wang .my age =2 B wang. my age =1

C Miao, my age =2 D Miao, my age=1

选择 C

int*p = (int*)(&kitty);

int*q = (int*)(&jd);

p 和 q 是分别指向 kitty 和 jd 两个对象的首地址，因为 Cat 和 Dog 都包含虚函数，所以 kitty 和 jd 两个对象均包含一个虚函数表，并通过一个指针指向它，p[0]和 q[0]就是该虚函数表指针，而 p[1]和 q[1]则为该对象的数据成员即 age 的值，p[1]=1,q[1]=2

p[1] = q[1]=2;

kitty 的 age 被修改为 2,

animal *pa;

pa = &kitty;

pa->print_age();

pa 指针声明时的类型为 animal，即基类，它指向派生类 kitty 对象，典型的多态特性，则 pa 的静态类型为 animal，动态类型为 cat。

而 print_age()是虚函数，因此是动态绑定，动态绑定指向是动态类型的成员，因此调用的是 kitty 的成员函数 print_age(),

cout << " Miao,my age= "<< this->age << endl;

此时 age=2.

题目不严谨

p[0]和 q[0]都是各自虚函数表的指针

32 位指针占 4 字节 64 位指针占 8 字节

而不论 32 位还是 64 位 int 都是 4 字节

因此 32 位时 p[1]和 q[1]指向的都是各自的 age p[1]被重新赋值

输出是 Miao,my age= 2

64 位时 指针占 8 字节 相当于两个 int 大小 p[1]和 q[1]指向的不是 age p[2]和 q[2]才是 age

输出是 Miao,my age= 1

7 int Func(int，int);不可与下列哪个函数构成重载（）

```
A int Func(int, int, int);

B double Func(int, int);

C double Func(double, double);

D double Func(int, double);
```

8 下面程序的输出结果是()

```
#include <iostream>

using namespace std;

int i = 1;

int fun(int n)

{

        static int a = 2;

        a++;

        return(a * n);

}



int main()

{

        int k = 5;

        {

                int i = 2;

                k += fun(i);

        }

        k += fun(i);

        cout << k;

        return(0);

}
```

作用域 答案是 15

9 分析一下这段程序的输出（微软）

```
#include<iostream>

using namespace std;

class B

{

public:

        B()

        {

                cout << "default constructor" << " ";

        }


        ~B()

        {

                cout << "destructed" << " ";

        }

        B(int i) : data(i)

        {

                cout << "constructed by parameter" << data << " ";

        }

private: int data;

};

B Play(B b)

{

        return b;

}

int main(int argc, char* argv[])
```

```
{
        B temp = Play(5);
        return 0;
}
```

```
A constructed by parameter5  destructed  destructed
B constructed by parameter5  destructed
C default constructor"  constructed by parameter5 destructed
D default constructor"  constructed by parameter5 destructed
destructed
```

- 调用 Play 函数需要将 5 隐式类型转换为 Play 函数中的形参 b，会调用 B 的 B(int i): data(i)，打印“constructed by parameter5”。
- Play 函数返回时需要调用 B 的复制构造函数给对象 temp 初始化。

可以加拷贝构造函数：

```
B(const B & s) {//拷贝构造函数，这里与编译器生成的一致
        cout << "copy constructor" << " ";
};
```

- Play 函数返回后需要调用 b 的析构函数，将 Play 函数的形参释放，打印“destructed”。
- main 函数返回后需要释放 temp，打印“destructed”。

10 指出下面程序哪里可能有问题？(多选题)（微软）

```
#include <stdio.h>
#include <string.h>
class CBuffer
{
        char* m_pBuffer;
        int m_size;
```

```
public:
	CBuffer()
	{
		m_pBuffer = NULL;
	}
	~CBuffer()
	{
		Free();
	}
	void Allocte(int size)  {   // (1)
		m_size = size;
		m_pBuffer = new char[size];
	}
private:
	void Free()
	{
		if (m_pBuffer != NULL)// (2)
		{
			delete[] m_pBuffer;
			m_pBuffer = NULL;
		}
	}
public:
	void SaveString(const char* pText) const //(3)
	{
		strcpy(m_pBuffer, pText);// (4)
	}
	char* GetBuffer() const
	{
```

```
        return m_pBuffer;

    }

};

void main(int argc, char* argv[])
{
    CBuffer buffer1;
    buffer1.SaveString("Microsoft");
    printf(buffer1.GetBuffer());
}
```

A 1　　B 2　　C 3　　D 4

链接：
https://www.nowcoder.com/questionTerminal/09ec766d373a43769603963664e231e7

来源：牛客网

正确答案应该为：A C D

理由：(1) 分配内存时，未检测 m_pBuffer 是否为空，容易造成内存泄露；

构造函数不是已经让他为 NULL 吗。但是，如果我们紧接着进行两次 Alllocate，那么第一次 Allocate 申请的内存，会在第二次 Allocate 后被悬挂起来，没有任何指针指向他，那么内存泄露。new 不会返回空， 如果申请内存失败，会抛异常。int *p = new(nothrow) int[3];

(3) 常成员函数不应该对数据成员做出修改，虽然可以修改指针数据成员指向的数据，但原则上不应该这么做；

```
void
        SaveString(const char* pText) const
    {
        strcpy(m_pBuffer,
            pText);
    }
```

```
void
        SaveString(const char* pText) (1)
{
        Allocte(strlen(pText) + 1); (2)
                strcpy(m_pBuffer,
                        pText);
}
```

(4) 字符串拷贝时，未检测是否有足够空间，可能造成程序崩溃。

11 有下列程序

```
using namespace std;
class SC
{
public:
        SC(int r) { R = _____; }
        int Get() { return *R; };
private:
        int* R;
};
int main()
{
        SC C(10);
        cout << C.Get() << endl;
        return 0;
}
```

请将构造函数补充完整，使得程序的运行结果是 10（ B ）

A new int R B R = new int(r); C &r D *r

{ R = new int(r); }；如果直接传 r 的地址进去是个临时变量

12 32 位机器上定义如下结构体：

```
struct xx
        {
                long long _x1;
                char _x2;
                int _x3;
                char _x4[2];
                static int _x5;
        };
        int xx::_x5;


#include <stdio.h>
class xx
{

        long long _x1;
        char _x2;
        int _x3;
        double _x4[2];
        virtual int fun() {};
        int fun1() {};
        static int _x5;
};
int main()
{
        printf("%d\n",sizeof(xx));
        return 0;
}
```

```
#include <stdio.h>
class xx
{
        //virtual int fun2() {};

        char _x2;
        long long _x1;
        int _x3;
        double _x4[2];
        virtual int fun() {};
        int fun1() {};
        static int _x5;
};
int main()
{
        printf("%d\n",sizeof(xx));
        return 0;
}
```

请问 sizeof(xx)的大小是()

A 19 B 20 C 15 D 24

原则 1：数据成员的对齐规则(以最大的类型字节为单位)。

第一个数据成员放在偏移为 0 的地方，后面的数据成员存放在偏移为该数据成员大小的整数倍的地方（比如 int 在 32 位机为 4 字节，则要从 4 的整数倍地址开始存储）

原则 2：结构体作为成员的对齐规则。

如果一个结构体 B 里嵌套另一个结构体 A，则结构体 A 应从 offset 为 A 内部最大成员的整数倍的地方开始存储。（struct B 里存有 struct A，A 里有 char，int，double 等成员，那 A 应该从 8 的整数倍开始存储。），结构体 A 中的成员的对齐规则仍满足原则 1、原则 2。

注意：

1. 结构体 A 所占的大小为该结构体成员内部最大元素的整数倍，不足补齐。

2. 不是直接将结构体 A 的成员直接移动到结构体 B 中

原则 3：收尾工作

结构体的总大小，也就是 sizeof 的结果，必须是其内部最大成员的整数倍，不足的要补齐。

原则 4：对于虚函数:

原则 5：静态函数　静态变量的位置

原则 6 ：对于指针，要注意 X64 和 X86

13 在 C++中，为了让某个类只能通过 new 来创建（即如果直接创建对象，编译器将报错），应该（）

A 将构造函数设为私有

B 将析构函数设为私有

C 将构造函数和析构函数均设为私有

D 没有办法能做到

将析构函数设为私有）。

编译器在为类对象分配栈空间时，会先检查类的析构函数的访问属性，其实不光是析构函数，只要是非静态的函数，编译器就会进行检查，如果类的析构函数是私有的，则编译器不会在栈上为类对象分配内存，因此，将析构函数设为私有，类对象就无法建立在栈（静态）上了，只能在堆上（动态 new）分配类对象。

```
/*
请设计一个类，只能在堆上创建对象

        实现方式：
        将类的构造函数私有，拷贝构造声明成私有。防止别人调用拷贝在栈上生成对象。
        提供一个静态的成员函数，在该静态成员函数中完成堆对象的创建
*/
#include <iostream>
```

```
using namespace std;

class test

{

public:

        static test* GetObj()

        {

                return new test();  //堆上申请并创建一个对象

        }

private:

        //构造函数私有

        test() { cout << "调用了构造函数" << endl; }

        //拷贝构造私有化，无法实例化对象

        test(const test& obj) {};

};

int main ()

{

        test* p = test::GetObj();  //通过调用静态函数获取对象的指针

        //test p1;

        return 0;

}
```

14 下面有关值类型和引用类型描述正确的是（）？

A 值类型的变量赋值只是进行数据复制，创建一个同值的新对象，而引用类型变量赋值，仅仅是把对象的引用的指针赋值给变量，使它们共用一个内存地址。

B 值类型数据是在栈上分配内存空间，它的变量直接包含变量的实例，使用效率相对较高。而引用类型数据是分配在堆上，引用类型的变量通常包含一个指向实例的指针，变量通过指针来引用实例。

C 引用类型一般都具有继承性，但是值类型一般都是封装的，因此值类型不能作为其他任何类型的基类。

D 值类型变量的作用域主要是在栈上分配内存空间内，而引用类型变量作用域主要在分配的堆上。

引申：指针和引用的区别

1 指针是一个变量，存储的是一个地址，引用跟原来的变量实质上是同一个东西，是原变量的别名 指针可以有多级，引用只有一级

2 指针可以为空，引用不能为 NULL 且在定义时必须初始化

指针在初始化后可以改变指向，而引用在初始化之后不可再改变

3sizeof 指针得到的是本指针的大小，sizeof 引用得到的是引用所指向变量的大小

4 当把指针作为参数进行传递时，也是将实参的一个拷贝传递给形参，两者指向的地址相同，但不是 同一个变量，在函数中改变这个变量的指向不影响实参，而引用却可以。

链接：
https://www.nowcoder.com/questionTerminal/053e885b9ded45e8bf2c34093d8157ca

来源：牛客网

B 错在，值类型变量不包含实例，实例是针对于对象的概念，当类实例化为对象的时候，这个时候可以称为是类的一个实例。同时，效率比较高这个概念比较模糊。

C 错在，封装的概念也是针对类而言的，值类型数据不存在封装概念。

D 错在，值类型变量可以作为成员变量存储在堆里，例如一个 Class A 中包含一个 int value，那么 value 是作为成员变量存储在堆中的。D 选项表述有漏洞。

15 下列说法正确的是（）

A 内联函数在运行时是将该函数的目标代码插入每个调用该函数的地方

B 内联函数在编译时是将该函数的目标代码插入每个调用该函数的地方 B

C 类的内联函数必须在类体内定义

D 类的内联函数必须在类体外通过加关键字 inline 定义

1）类内定义的函数都是内联函数，不管是否有 inline 修饰符 2）函数声明在类内，但定义在类外的看是否有 inline 修饰符，如果有就是内联函数，否则不是

16 下面程序的输出是什么？

```
#include <iostream>

using namespace std;

class parent

{

        int i;

protected:

        int x;

public:

        parent() { x = 0; i = 0; }

        void change() { x++; i++; }

        void display();

};

class son :public parent

{

public:

        void modify();

};

void parent::display() { cout << "x=" << x << endl; }

void son::modify() { x++; }

int main()

{

        son A; parent B;

        A.display();

        A.change();
```

```
        A.modify();
        A.display();
        B.change();
        B.display();
}
```

```
A x=1 x=0 x=2
B x=2 x=0 x=1
C x=0 x=2 x=1
D x=0 x=1 x=2
```

链接：
https://www.nowcoder.com/questionTerminal/fc38c01cdf814315a8085c826302a383

来源：牛客网

类的成员变量未声明访问权限则默认 private，这是它和结构体的唯一区别。--基类的 private 成员不能被子类访问，但其 protected 成员可以被子类访问，两者均不可在类外访问，故文中子类可以访问 x,不能访问 i

17 下面程序运行时的输出结果是？

```
#include<iostream>
using namespace std;
class MyClass
{
public:
        MyClass(int i = 0)
        {
                cout << i;
        }
```

```
        MyClass(const MyClass& x)
        {
                cout << 2;
        }
        MyClass& operator=(const MyClass& x)
        {
                cout << 3;
                return *this;
        }
        ~MyClass()
        {
                cout << 4;
        }
};
int main()
{
        MyClass obj1(1), obj2(2);
        MyClass obj3 = obj1;
        return 0;
}
```

A 11214444

B 11314444

C 122444

D 123444

链接：
https://www.nowcoder.com/questionTerminal/cf1a3145d1b946c1861c9d10b8629665

来源：牛客网

解析

C D 辨析：

关键是区分 浅/深拷贝操作 和 赋值操作:

没有重载=之前：

A a ;

A b;

a = b;

这里是赋值操作。

A a;

A b = a;

这里是浅拷贝操作。

重载 = 之后：

A a ;

A b;

a = b;

这里是深拷贝操作（当然这道题直接返回了，通常我们重载赋值运算符进行深拷贝操作）。

A a;

A b = a;

这里还是浅拷贝操作。

所以 MyClass obj3 = obj1; 调用的是拷贝构造函数。

如果写成 MyClass obj3; obj3 = obj1; 输出的结果就是 1203444

18 假定有类 AB，有相应的构造函数定义，能正确执行

AB a(4),b(5),c[3],*p[2]={&a,&b};语句，请问执行完此语句后共调用该类的构造函数次数为——

A 5　B 4　C 3　D 9

链接：
https://www.nowcoder.com/questionTerminal/bf70aadeb78949c2a61b1b561a0ee784

来源：牛客网

A 只有给对象分配内存才调用构造函数 AB a(4) 定义对象 a，调用了带一个参数的构造 AB b(5)跟上面的性质类似，调用了带一个参数的构造 AB c[3] 跟上面的性质类似，定义对象数组,调用无参构造 3 次 AB *p 这至是一个指针,没有指向任何空间,更么有分配内存,不会调构造

19 以下程序片段输出什么内容:

```
class Demo {
public:
        Demo() :count(0) {}
        ~Demo() {}

        void say(const std::string& msg) {
                fprintf(stderr, "%s\n", msg.c_str());
        }
private:
        int count;
```

```
};


int main(int argc, char** argv) {
        Demo* v = NULL;
        v->say("hello world");
}
```

A 无有效输出，程序 coredump

B 输出"hello world"

C 随机输出"hello world"的字串

D 不确定答案

首先可以确定的是 say 方法不是虚函数且不涉及类成员。C++ 非虚的成员函数是静态绑定，只要不涉及类成员空指针调用是没有问题

20 下列说法正确的是

A 使用 typedef 定义新类型名后，新类型名与原类型名实际上是等价的

B 结构体类型中的各个成分均不能是数组或指针

C 结构体类型的变量，不能在声明结构体类型组成时一起定义

D 元素为结构体类型的数组，只能在声明过结构体类型之后，单独进行定义

[答案]A

[解析]本题考查 typedef 的使用方法，typedef 对已存在的类型使用一个新的名字,结构体类型中的成分可以是数组和指针,所以 B 选项错误,结构体类型的变量可以在声明结构体的时候一起定义,C 选项错误,D 选项中可以一起定义。

21 请问刚进入 func 函数时，参数在栈中的形式可能为（左侧为地址，右侧为数据—）

int i=0x22222222；

char szTest[]=”aaaa”; //a 的 ascii 码为 0x61

func(I, szTest);　　//函数原型为 void func(int a,char *sz);

A 0x0013FCF0　0x61616161

0x0013FCF4　　0x22222222

0x0013FCF8　　0x00000000

B 0x0013FCF0　0x22222222

0x0013FCF4　　0x0013FCF8

0x0013FCF8　　0x61616161

C 0x0013FCF0　0x22222222

0x0013FCF4　　0x61616161

0x0013FCF8　　0x00000000

D 0x0013FCF0　0x0013FCF8

0x0013FCF4　　0x22222222

0x0013FCF8　　0x61616161

链接：
https://www.nowcoder.com/questionTerminal/66d21ec011e844e9977425825a046a0f

来源：牛客网

函数调用，入栈是右边参数、左边参数，然后函数返回地址。

栈是由高地址向地地址增长。

PS：排版真是差

选 D

22 有关虚函数和纯虚函数说法错误的是（）

A 被 virtual 关键字修饰的成员函数，就是虚函数

B 在基类中实现纯虚函数的方法是在函数原型后加“=0” virtual void funtion1()=0

C 同时含有纯虚拟函数的类称为抽象类，它可以被实例化，但是对象不可以调用纯虚函数

D 使用纯虚函数的意义是在很多情况下，基类本身生成对象是不合情理的

抽象类的定义：在 C++中，含有纯虚函数的类称为抽象类，它不能生成对象。抽象类是不完整的，它只能用作基类。

23 下列语句错误的是（）

A int * p=new int(10);

B int*p=new int[10];

C int*p=new int;

D int*p=new int[40](0);

答案：D　　不能为数组指定新的初始化值设定项

24 在 64 位系统中，有如下类：

```
class C
    {
    public:
        char a;
        static char b;
        void* p;
        static int* c;
```

```
        virtual void func1();
        virtual void func2();
    };
```

那么 sizeof（C）的数值是（）

A 9

B 17

C 32

D 24

链接：
https://www.nowcoder.com/questionTerminal/8262288d505d4fc599ebd9c8e7fd86ce

来源：牛客网

解析

应该是 D

sizeof（类）计算的是类中存在栈中的变量的大小，而类中的 b 和*c 都是 static 静态变量，存在全局区中，因此不在计算范围之内，于是只剩下 char a，void *p 和两个 virtual 虚函数，a 是 char 类型，占用一个字节，p 是指针，在 64 位系统的指针占用 8 个字节，而两个虚函数只需要一个虚函数表指针，也是八个字节，加上类中的对齐方式（char a 对齐时后面补上 7 个字节），故答案为 24.

25 在 Windows 32 位操作系统中，假设字节对齐为 4，对于一个空的类 A，sizeof(A)的值为（）？

A 0　　B 1　　C 2　　D 4

解析

肯定不是 A.0　C++对空类或者空结构体 ,对其 sizeof 操作时候,默认都是 1 个字节.所以答案 选 B

26 设有以下定义：

a[4][3]={1,2,3,4,5,6,7,8,9,10,11,12};

int (* prt)[3]=a, * p=a[0];

则以下能够正确表示数组元素 a[1][2]的表达式是哪些？

A * ((* prt+1)[2])

B * (* (p+5))

C (* prt+1)+2

D * (* (a+1)+2)

链接：
https://www.nowcoder.com/questionTerminal/dbcdd264ab2249249b588d976c48c9d2

来源：牛客网

1. 拆分二维数组

int a[4][3] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };

拆分成：

int b[3] = {1, 2, 3 };

int c[3] = {4, 5, 6 };

int d[3] = {7, 8, 9 };

int e[3] = {10, 11, 12 };

2. 为何拆分？

以“b[3] = {1, 2, 3 }”为例：

b 是数组第一个元素的地址，这里 b 相当于整型指针！上述 b，c，d，e 都是整型指针。

那么就有：a[4] = { b, c, d, e };

这是一个一维数组，其中的**元素都是整型指针**。

a 是什么？是数组 a 中第一个元素 b 的地址！

根据上述这种理解，发现可以很方便的解出这道题。

3. 例题分析：AC 选项

先看“int (* prt)[3]=a”,相当于：

int b[3];

int *prt = &b;

即定义了一个指向“数组第一个元素的地址”的指针 prt；

而从 1,2 分析来看，a 表示的正是 b 的地址。所以，这里等价于：prt = a。

我们看 AC 选项，先把 ptr 都换成 a。

A：* ((* prt+1)[2])

*a 即 a[0]，也就是 b；

(b+1) 表示 元素 2 的地址，也就是 a[0][1]的地址；

(b+1)[2] → *((b+1) + 2) = *(b+3) = b[3]，越界了！其实就是 c[0]，VS 上验证过，输出也是 4.

而答案提供的相当于*(b[3])，连数组元素都算不上！

注：下标和指针转化公式：(a+n) = a[n]*

C：(* prt+1)+2

(* a+1)+2 等价于(b+1) + 2 = b+3，是 4 的地址，也就是 c[0]的地址；同样错误。

不过可以验证*((* prt+1)+2)，输出为 4.

5. B 选项分析：*（*(p+5)）

int *p = a[0]，相当于 int *p = b，遇到 p 直接用 b 替换就行了！

(p+5)等价于 b[5]，也就是 c[2]，元素 6，前面还多个，所以这个错的也很明显。

6. D 选项

下标和指针转化公式：*(a+n) = a[n]，这个正反都可以使用，而且很好用。

27 以下代码实现了从表中删除重复项的功能，请选择其中空白行应填入的正确代码（）

```
template<typename T>
void removeDuplicates(list<T>& aList)
{
	T curValue;
	list<T>::iterator cur, p;
	cur = aList.begin();
	while (cur != aList.end())
	{
		curValue = *cur;
		//空白行
		while (p != aList.end())
		{
			if (*p == curValue)
			{
				//空白行
			}
			else
			{
```

```
                    p++;
                }
            }
        }
}
A p=curr+1;aList.erase(p++);

B p=++curr;aList.erase(p++);

C p=curr+1;aList.erase(p);

D p=++curr;aList.erase(p);
```

答案 B　　错选 D

当使用一个容器的 insert 或者 erase 函数通过迭代器插入或删除元素"可能"会导致迭代器失效 iterator 失效主要有两种情况：

1、iterator 变量已经变成了“野指针”，对它进行*,++,--都会引起程序内存操作异常；

2、iterator 所指向的变量已经不是你所以为的那个变量了。

不同的容器，他们 erase()的返回值的内容是不同的，有的会返回被删除元素的下一个的 iterator，有的则会返回删除元素的个数。

28 在 32 位操作系统中，我们定义如下变量 int (*n)[10]; 请问调用函数 sizeof(n),返回值为（）

A 4　　B 40　　C 8　　D 80

解析

【正确答案】A

【解析】n 是指针变量，不论指针变量的类型是什么，在同一个平台下大小都一样。在 32 位操作系统中占 4 个字节，在 64 位操作系统中占 8 个字节。

29 在 32 位系统下运行以下程序，可能的输出结果为（　　）

```
int main()
{
	int i, a[5];
	for (i = 0; i <= 30; i++)
	{
		a[i] = 0;
		printf("%d:hello\n", i);
	}
	printf("%d:hello world", i);
	return 0;
}
```

A 三十行的 i:hello (i∈[0,30]) 和一行 30:hello world

B 三十行的 i:hello (i∈[0,30]) 和一行 31:hello world

C 多行的 i:hello (i∈[0,30])

D 多行的 i:hello (i∈[0,31])

链接：
https://www.nowcoder.com/questionTerminal/cc49fd222f8948b795349660a2cfa34a

来源：牛客网

正确答案：C

本题是考察数组越界会导致死循环。经过实验得知，当循环体中改为 i<=7 后，开始 0-6 的死循环，i<=6 及之前的，都可以退出循环。

栈中是从高地址指向低地址的，如下：

高地址 | i | a[4] | a[3] | a[2] | a[1] | a[0] | 低地址

所以 i 在高地址，而数组是连续存储的，而又由于有些编译器做了优化，使数组和 i 之间留有内存间隙，如开头所述，我用的 VS2010 留了 2 个间隙，但是，如果 i 越界严重，比如不小心给了 50，还是会导致死循环。知 a[6]与 i 占据一块空间，当执行到 i=6，a[6]=0，将 i 的值又变成了 0,又进入循环段执行下去，i 永远的不大于 30，造成死循环

30 对于纯虚函数描述正确的是(　)

A 含有纯虚函数的类不能被声明对像，这些类被称为抽象类

B 继承抽象类的派生类可以被声明对像，但要在派生类中完全实现基类中所有的纯虚函数

C 继承抽象类的派生类可以被声明对像，不需要实现基类中全部纯虚函数，只需要实现在派生类中用到的纯虚函数

D 虚函数和纯虚函数是一样的，没什么区别

正确答案：A B

链接：
https://www.nowcoder.com/questionTerminal/05e7f66cf3504a26aeafc98035b3fd9a

来源：牛客网

虚函数可以被直接使用，也可以被子类(sub class)重载以后以多态的形式调用，而纯虚函数必须在子类(sub class)中实现该函数才可以使用，因为纯虚函数在基类(base class)只有声明而没有定义。

31 程序的输入为：

9

12215666

则输出为( )

```
#include<bits/stdc++.h>
```

```
using namespace std;
vector<int>g[10];
int ans = 0;
void dfs(int x) {
	if (g[x].size() == 0) {
		ans++;
		return;
	}
	for (int i = 0; i < g[x].size(); ++i) {
		dfs(g[x][i]);
	}
}
int main() {
	int n, x;
	scanf("%d", &n);
	for (int i = 2; i <= n; ++i) {
		scanf("%d", &x);
		g[x].push_back(i);
	}
	dfs(1);
	cout << ans << endl;
	return 0;
}
```

A 4　B 5　C 6　D 7

答案 B

32 C++中的拷贝构造函数在下面哪些情况下会被调用（）

A 对象创建的时候

B 使用一个类的对象去初始化该类的一个新对象

C 被调用函数的形参是类的对象

D 当函数的返回值是类的对象时，函数执行完成返回调用者

正确答案：B C D

链接：
https://www.nowcoder.com/questionTerminal/3cc51103f25e44cbb00ea329685994a4

来源：牛客网

拷贝构造函数从来不显示调用，而是由编译器隐式地调用：

（1） 用类的一个对象去初始化另一个对象时；

（2） 当函数的形参是类的对象时（也就是值传递时），如果是引用传递则不会调用；

（3） 当函数的返回值是类的对象或引用时。

33 以下代码输出的值是：

```
int x = 4;

void incre()

{

        static int x = 1;

        x *= x + 1;

        printf(“ % d”, x);

}

int _tmain(int argc, _TCHAR* argv[])

{

        int I;
```

```
        for (i = 1; i < x; i++) {
                incre();
        }
        return 0;
}
```

A 42　　B 2　　C 4　　D 5

正确答案：A

static 初始化一次，使得函数结束时局部变量 x 不被释放，下一次使用函数时 x 使用上次的值

此外局部变量会覆盖全局变量

34 在哪种派生方式中，派生类可以访问基类中的 protected 成员（　　）

A public 和 private

B public 和 protected

C protected 和 private

D public、protected、private 均可

正确答案：D

链接：
https://www.nowcoder.com/questionTerminal/156f95a4eb23498ba03b5c4b1d340ad0

来源：牛客网

不论哪种派生方式，派生类都可以访问基类中的 protected 成员

protect 保护成员在　派生类　中会变成 私有成员，派生类的成员函数是可以访问的。

private 私有成员在派生类中是不可访问成员，必须通过基类的成员函数访问

35 下设 A 为基类，AX 为派生类

```
class A {
        public:
                A();
                ~A();

                Data* data;
        };

        class AX : public A {
        public:
                AX();
                ~AX();

                AXData* ax_data;
        };
```

以下不会内存泄漏的写法吗？（多选）

A `AX* p = new AX(); delete p;`

B `A* p = new AX(); delete p;`

C `std::shared_ptr<AX> p{new AX()};`

D `std::shared_ptr<A> p{new AX()};`

正确答案：A C D

题目解析：

基类 A 析构函数不是虚函数，B 选项会引起内存泄漏。shared_ptr 可以保留原始指针类型，所以可以正确析构。

36 下列关于 C++类的说法中错误的有哪些？

A 一个空类默认会生成构造函数,拷贝构造函数,赋值操作符,析构函数

B 一个类可以有多个析构函数

C 类中析构函数可以为 virtual,可以被重载

D 类的构造函数如果都不是 public 访问属性,则类的实例无法创建

正确答案：B C D

链接：
https://www.nowcoder.com/questionTerminal/ef421bf0f5304868b27e5c0ac41116e5

来源：牛客网

一个空类在以下情况下会生成默认构造函数

- 含有类对象数据成员，该类对象类型有默认构造函数，生成的默认构造函数调用类成员的默认构造函数对其进行初始化
- 当前类是派生类，其基类具有默认构造函数
- 具有虚函数，默认构造函数用于配置虚表指针 vptr
- 带有虚基类的类

37 以下 32 位 C++程序，请计算 sizeof 的值_____

void Func (char str[100]) { sizeof(str) = ? }

void*p = malloc(100); sizeof(p) = ?;

A 100,4 B 4, 100 C 4,4 D 100, 100

答案 C

38 对以下数据结构中 data 的处理方式描述正确的是()

```
struct Node
{
   int size;
```

```
    char data[0];
};
```

A data 将会被编译成一个 char *类型指针

B 全部描述都不正确

C 编译器会认为这就是一个长度为 0 的数组,而且会支持对于数组 data 的越界访问

D 编译器会默认将数组 data 的长度设置为 1

答案 C

柔性数组(Flexible Array)也叫伸缩性数组，也就是**变长数组**，反映了 C 语言对精炼代码的极致追求。

这种代码结构产生于对动态结构体的需求， 比如我们需要在结构体中存放一个动态长度的字符串时，就可以用柔性数组。

C99 使用不完整类型来实现柔性数组，标准形式如下：

```
struct MyStruct

 {

 int a;

 double b;

 char c[]; // or char c[0]; 也可以用其他数据类型;

 };
```

c 不占用 MyStruct 的空间，只是作为一个符号地址存在，而且必须是结构体的最后一个成员。

https://blog.csdn.net/gatieme/article/details/64131322

39 下列代码的结果是（2，5）

```
#include<stdio.h>

    main() {

        int a[5] = { 1, 2, 3, 4, 5 };

        int* ptr = (int*)(&a + 1);
```

```
        printf("%d,%d", *(a + 1), *(ptr - 1));
    }
```

A 3,5

B 2,4

C 2,5

D 3,4

C，a 代表了数组的整体，&a+1 就超出了数组的范围

40 下列代码的结果是（36）

```
#define f(x) x*x

#include<stdio.h>

        main() {

                int a = 6, b = 2, c;

                c = f(a) / f(b);

                printf("%d", c);

          }
```

A．9 B．6 C．36 D．18

答案解析

C

41 在 32 位机器上用 gcc 编译以上代码，求 sizeof(A)，sizeof(B)分别是（）

```
class A {
    int a;
    short b;
    int c;
    char d;
};
```

```
class B {
    double a;
    short b;
    int c;
    char d;
};
A 12 16   B 12 12   C 16 24    D 16 20
```

答案 D

GCC 默认 4 字节对齐

42 当一个类的某个函数被说明为 virtual，则在该类的所有派生类中的同原型函数_____?

A 只有 被重新说明时才识虚函数

B 只有被重新说明为 virtual 时才是虚函数

C 都不是虚函数

D 都是虚函数

答案解析 D

父类的所有子类中，同名函数都是虚函数。

43 哪个操作符不能作为类成员函数被重载?

A ?:

B ++

C []

D ==

E *

解析

A.应该说的是 三目运算符 ?: 吧

不可重载运算符包括 ?:　　::　　.　　.*　这四个

44 以下程序在 32 位机器上运行输出是

```
#include<iostream>
	using namespace std;
	class animal
	{
	protected:
		int age;
	public:
		virtual void print_age(void) = 0;
	};
	class dog : public animal
	{
	public:
		dog() { this->age = 2; }
		~dog() { }
		virtual void print_age(void)
		{
			cout << "Wang, my age = " << this->age << endl;
		}
	};
```

```
class cat : public animal
{
public:
        cat() { this->age = 1; }
        ~cat() { }
        virtual void print_age(void)
        {
                cout << "Miao, my age = " << this->age << endl;
        }
};
int main(void)
{
        cat kitty;
        dog jd;
        animal* pa;
        int* p = (int*)(&kitty);
        int* q = (int*)(&jd);
        p[0] = q[0];
        pa = &kitty;
        pa->print_age();
        return 0;

  }
```

A Wang, my age = 2
B Wang, my age = 1
C Miao, my age = 2
D Miao, my age = 1
E 程序编译报错
F 程序运行报错

链接：
https://www.nowcoder.com/questionTerminal/821815baa74e41cda5484b8962519db5

来源：牛客网

答案为 B。

含有虚函数的类的对象在内存的第一项是指向虚函数表的指针

p[0] = q[0];把 q 的虚表指针赋给了 p 的第一项

但是后面的内容没有变，age 没有变

pa 指向 kitty，交换指向虚函数表的指针后，pa 调用 jd 的 print_age，显示的 kitty 的 age

45 在上下文和头文件均正常的情况下，下面代码的输出结果是()

```
int main() {
        int pid;
        int num = 1;
        pid = fork();
        if (pid > 0) {
                num++;
                printf("in parent:num:%d addr:%x\n", num, &num);
        }
        else if (pid == 0) {
```

```
            printf("in child:num:%d addr:%x\n", num, &num);
        }
    }
```

A 父子进程中输出的 num 相同,num 地址不相同

B 父子进程中输出的 num 不同,num 地址相同

C 父子进程中输出的 num 相同,num 地址也相同

D 父子进程中输出的 num 不同,num 地址不相同

答案解析： B

链接：
https://www.nowcoder.com/questionTerminal/0d5acd7276324a6981d696fbe88f817e

来源：牛客网

虚拟地址空间。num 地址的值相同，但是其真实的物理地址却不一样。

linux 下实现了一下，发现地址值真的一样。

fork 之后子进程复制了父进程的数据、堆栈。

但是由于地址重定位器之类的魔法存在，

所以，看似一样的地址空间（虚拟地址空间），

其实却是不同的物理地址空间。

同时可以验证 c 程序中输出的地址空间其实都是虚拟地址空间。

46 下列关于数组的描述，错误的是（）

A C++中数组的存储方式为列优先存储

B 数组名可以作为实参赋值给指针类型的形参

C 数组下标索引从 1 开始，至数组长度 n 结束

D 数组指针的语法形式：类型名 *数组名[下标表达式]

正确答案：A C D

解析

链接：
https://www.nowcoder.com/questionTerminal/8b64a2d759d840f7b5cda1a994cb3ed7

来源：牛客网

二维数组在逻辑上可以理解为二维的,例如 int arr[3][4],可以想成其含有 3 行 4 列,共 3*4 个元素,当然也可按上述方式去理解,则理解为该二维数组中含有 3 个一维数组,其中每个一维数组中又含有四个 int 类型的元素,这两种方式其实是一致的。但是计算机的内存是线性的,这意味着内存对数据的存储方式都是一维线性的

47 当参数*x=1, *y=1, *z=1 时， 下列不可能是函数 add 的返回值的( )?

```
int add(int* x, int* y, int* z) {
        *x += *x;
        *y += *x;
        *z += *y;
        return *z;
}
```

A 4

B 5

C 6

D 7

链接：
https://www.nowcoder.com/questionTerminal/29d7b8d0a2c1451d8c020006dbd6e0a7

来源：牛客网

D

开始不知道啥意思，后经牛客网的大神指点才知道这题要考虑的是，x，y，z 三个参数是否指向同一地址（或者说调用该函数时是否实参相同），如：当 a=b=c=1 时，add(&a,&a,&a)，add(&a,&b,&c)。

通过写程序测试得出结果，不可能得到答案 7。

48 以下叙述中正确的是（）

A 即使不进行强制类型转换，在进行指针赋值运算时，指针变量的基类型也可以不同

B 如果企图通过一个空指针来访问一个存储单元，将会得到一个出错信息

C 设变量 p 是一个指针变量，则语句 p=0;是非法的，应该使用 p=NULL;

D 指针变量之间不能用关系运算符进行比较

链接：
https://www.nowcoder.com/questionTerminal/7f64dd32ec2f4417a36b413e89a0f49f

来源：牛客网

正确答案：B

题目解析：

解析 ：【解析】 A 选项描述不正确，指针变量的赋值只能赋予地址， 决不能赋予任何其它数据，否则将引起错误； C 选项中， p=NULL ；和 p=0 ；是等价的；D 选项中，指向同一数组的两指针变量进行关系运算可表示它们所值数组元素之间的关系。因此 B 选项正确。

49 关于类与对象，下面哪一种说法是错误的？

A 一个对象是某个类的一个实例

B 一个实例是某个类型经实例化所产生的一个实体

D 创建一个对象必须指定被实例化的一个类

C 一个类的多个对象之间不仅持有独立的数据成员，而且成员函数也是独立的

正确答案：D

解析

静态数据成员则独立于该类的任何对象，在所有对象之外，单独开辟空间存储。在为对象所分配的空间中不包含静态成员所占的空间。

50 以下程序的输出结果为（）

链接：
https://www.nowcoder.com/questionTerminal/f6c9d056bac94c03b21eb31a5e1db104

来源：牛客网

```
using namespace std;
        void print(char** str) {
                ++str;
                cout << *str << endl;
        }
        int main() {
```

```
        static char* arr[] = { "hello","world", "c++" };
        char** ptr;
        ptr = arr;
        print(ptr);
        return 0;
}
```

A hello

B world

C 字符 w 的起始地址

D 字符 e

正确答案：B

解析：

链接：
https://www.nowcoder.com/questionTerminal/f6c9d056bac94c03b21eb31a5e1db104

来源：牛客网

arr 是字符串"hello"首地址的地址，因此是二级指针。

str = arr; str++; 这时，str 是"world"首地址的地址。

*str: 拿到"world"的首地址，用 cout 输出字符串。

51 如下程序用于输出“Welcome to Huawei Test”，请指出其中潜在风险的位置（）

```
char* GetWelcome(void) {
        char* pcWelcome;
        char* pcNewWelcome;
```

```
        pcWelcome = "Welcome to Huawei Test";
        pcNewWelcome = (char*)malloc(strlen(pcWelcome));     //1
        if (NULL == pcNewWelcome) {
            return NULL;          //2
        }
        strcpy(pcNewWelcome, pcWelcome);             //3
        return pcNewWelcome;   //4
    }
    printf("%s\n", GetWelcome());
```

A 1

B 2

C 3

D 4

正确答案：A C

解析：

链接：
https://www.nowcoder.com/questionTerminal/74ae207cfbd846a7b05057d8cf8dcaaa

来源：牛客网

第一处：malloc 动态分配的内存是在堆上的，需要进行 free()函数释放，这里假设是分配正确的，这样第四处就可以认为是正确的，因为它还没有释放。 第二处：可以认为是在 strcpy 函数执行前的自检测，如果为 NULL，由实现者决定其返回值，一般为 NULL。 第三处：strcpy 函数的功能是把从 src 地址开始且含有'\0'结束符的字符串复制到以 dest 开始的地址空间。但是在第一处进行动态分配内存的时候使用的是 strlen 而不是 sizeof，这样就少了'\0'结束符，不能正确进行复制操作。 因此，第一处和第三处是错误的。

52 以下程序输出结果是____

```
#include <iostream>

using namespace std;
```

```
class A {
public:
        A() :m_iVal(0) { test(); }
        virtual void func() { std::cout << m_iVal << ' '; }
        void test() { func(); }
public:
        int m_iVal;
};
class B : public A {
public:
        B() { test(); }
        virtual void func() {
                ++m_iVal;
                std::cout << m_iVal << ' ';
        }
};
int main(int argc , char* argv[]) {
        A* p = new B;
        p->test();
        return 0;
}
```

A 1 0

B 0 1

C 0 1 2

D 2 1 0

E 不可预期

F 以上都不对

答案解析：

链接：
https://www.nowcoder.com/questionTerminal/94b0fd680ede438ca7fdde4888a39537

来源：牛客网

解析

本问题涉及到两个方面：

1.C++继承体系中构造函数的调用顺序。

2.构造函数中调用虚函数问题。

C++继承体系中，初始化时构造函数的调用顺序如下

(1)任何虚拟基类的构造函数按照他们被继承的顺序构造

(2)任何非虚拟基类的构造函数按照他们被继承的顺序构造

(3)任何成员对象的函数按照他们声明的顺序构造

(4)类自己的构造函数

据此可知 A*p = newB;先调用 A 类的构造函数再调用 B 类的构造函数。

构造函数中调用虚函数,虚函数表现为该类中虚函数的行为，即在父类构造函数中调用虚函数，虚函数的表现就是父类定义的函数的表现。why？原因如下：

假设构造函数中调用虚函数，表现为普通的虚函数调用行为，即虚函数会表现为相应的子类函数行为，并且假设子类存在一个成员变量 int a；子类定义的虚函数的新的行为会操作 a 变量，在子类初始化时根据构造函数调用顺序会首先调用父类构造函数，那么虚函数回去操作 a，而因为 a 是子类成员变量，这时 a 尚未初始化，这是一种危险的行为，作为一种明智的选择应该禁止这种行为。所以虚函数会被解释到基类而不是子类。

据此可以得到答案 C 正确

53 32 位编译器下，sizeof(void)的值是多少?

A 0

B 4

C 这取决于主机的字的大小。

D 8

E 编译错误或者为 1

正确答案：E

解析：

sizeof(void) 编译出错，提示 非法的 sizeof 操作数

sizeof(void*) 准确的说，与编译器的目标平台有关。如果目标平台是 32 位的，那么 sizeof(void*) 就是 4，如果是 64 位的，那么 sizeof 就是 8，如果是 16 位的，就是 2 。

54 下列程序编译时会出现错误，请根据行号选择错误位置( )

```
#include <iostream>

using namespace std;

class A {

        int a1;

protected:

        int a2;

public:

        int a3;

};

class B : public A {

        int b1;

protected:
```

```
        int b2;
public:
        int b3;
};
class C :private B {
        int c1;
protected:
        int c2;
public:
        int c3;
};
int main() {
        B obb;
        C obc;
        cout << obb.a1;//1
        cout << obb.a2;//2
        cout << obb.a3;//3
        cout << obc.b1;//4
        cout << obc.b2;//5
        cout << obc.b3;//6
        cout << obc.c3;//7
        return 0;
}
```

A 1,2

B 2,5,7

C 3,4,7

D 4,5,6

正确答案：A D

55 下列运算符，在 C++语言中不能重载的是（）

A *

B .*

C ::

D operator delete

链接：
https://www.nowcoder.com/questionTerminal/399c358367f7474ead9edeb10ed599d5

来源：牛客网

解析

答案选择 BC。 并不是所有的操作符都能被重载。除了 . ， .* ， :: ， ？ : ， sizeof ， typeid 这几个运算符不能被重载，其他运算符都能被重载。

. 表示成员选择

.*表示指向成员操作的指针

？=表示条件操作 expr?expr:expr

56 若 PAT 是一个类，则程序运行时，语句“PAT(*ad)[3];”调用 PAT 的构造函数的次数是（ ）。

A 2

B 3

C 0

D 1

链接：
https://www.nowcoder.com/questionTerminal/cdcb22d115c24c54a67c7e66be117526

来源：牛客网

解析

PAT(*ad)[3];

ad 首先是个指针；

ad 是个指向有着三个 PAT 元素的数组的指针；

这里只是声明了指针，虽然指针指向的数组有三个 PAT 对象，但是没有实例化其中的对象，所以并没有调用构造函数。

57 下面程序输出结果是什么?

```
#include<iostream>
        using namespace std;
        class A {
        public:
                A(char* s)
                {
                        cout << s << endl;
                }
                ~A() {}
        };
        class B :virtual public A
        {
        public:
```

```
        B(char* s1, char* s2) :A(s1) {
                cout << s2 << endl;
        }
};
class C :virtual public A
{
public:
        C(char* s1, char* s2) :A(s1) {
                cout << s2 << endl;
        }
};
class D :public B, public C
{
public:
        D(char* s1, char* s2, char* s3, char* s4) :B(s1, s2), C(s1, s3), A(s1)
        {
                cout << s4 << endl;
        }
};
int main() {
        D* p = new D("class A", "class B", "class C", "class D");
        delete p;
        return 0;
}
```

A class A class B class C class D

B class D class B class C class A

C class D class C class B class A

D class A class C class B class D

正确答案：A

解析：

链接：
https://www.nowcoder.com/questionTerminal/7c8af7312d1a4c8caccf73d72951a139

来源：牛客网

1 创建派生类对象时，程序先调用基类构造函数，再调用派生类构造函数

2 派生类有多个基类时，按派生类声明中的基类列表顺序调用构造函数

3 派生类有的多个基类中有虚基类时，优先调用虚基类的构造函数。

4 派生类并不继承基类的构造函数，所以将基类的构造函数声明为虚函数是没有意义的。（虽然语法上是对的）。

5 派生类构造函数的参数列表中，应当包含所有基类构造函数需要的参数

6 派生类构造函数的参数列表后，应当指明基类的构造函数，否则使用基类的默认构造函数；该指明顺序与基类构造函数调用顺序无关（派生类 有多个基类时）。

7 在多重派生中，由同一个初始基类派生的多个类作为派生类的基类时，该初代基类的构造函数只调用一次。

58 在 64 位系统以及 64 位编译器下，以下描述正确的是

```
struct T {
        char a;
        int* d;
        int b;
        int c : 16;
        double e;
};
T* p;
```

A sizeof(p) == 24

B sizeof(*p) == 24

C sizeof(p->a) == 1

D sizeof(p->e) == 4

链接：
https://www.nowcoder.com/questionTerminal/8e8b73ee8f3a402ba47876f8e0b2b62d

来源：牛客网

字节对齐的三个准则：

1） 结构体变量的首地址能够被其最宽基本类型成员的大小所整除；

2）结构体每个成员相对于结构体首地址的偏移量都是成员大小的整数倍，如有需要编译器会在成员之间加上填充字节；

3）结构体的总大小为结构体最宽基本类型成员大小的整数倍，如有需要编译器会在最末一个成员之后加上填充字节。

再来看这道题：

a 占一个字节（注：地址为[0]），d 作为 64 位指针占 8 个字节（注 1：32 位占四个字节，p 也一样）（注 2：根据上面的准则 2，d 的偏移量要为 8 的整数倍，所以 d 的地址为[8]-[15]，而非[1]-[8]，下同），b 占了 4 个字节（注：地址为[16][19]），c 指定为 16 为，占了两个字节（注：地址为[20,21]）,e 占 8 个字节，（同 d 的分析一样，e 的地址应该为[24][31]），所以 A 的答案应该是 8，B 的答案是 32，C 正确，D 的答案为 8。

59 设置虚基类的目的是

A 简化程序

B 消除二义性

C 提高运行效率

D 减少目标代码

正确答案：B

60 【网络题】Telnet、SNMP、FTP 和 HTTP 都是在 TCP 之上构建的应用层协议。（ ）

A 正确

B 错误

正确答案：B

解析：

链接：
https://www.nowcoder.com/questionTerminal/d8195e93af74465ca36b539447b3b83b

来源：牛客网

应用层协议：HTTP，FTP，TFTP，TELNET，DNS，EMAIL，SNMP 等。

其中 DNS，TFTP 和 SNMP 因为一次性传输的数据很少，所以是建立在 UDP 议之上的；

而 HTTP，FTP，TELNET，EMAIL 等协议是建立在 TCP 协议之上的。

61 关于迭代器失效，下面说法错误的有?

A 当向 vector 容器插入（push_back）一个元素后，end 操作返回的迭代器肯定失效

B 当向 vector 容器插入(push_back)一个元素后，capacity 返回值与没有插入元素之前相比有改变,此时 first 和 end 操作返回的迭代器都会失效

C 当 vector 容器 erase 一个元素后，仅指向删除点的迭代器失效

D 在 deque 容器的任何其他位置的插入和删除操作将使指向该容器元素的所有迭代器失效

E 对于节点式容器(map, list, set)元素的删除，插入操作会导致指向该元素的迭代器失效，其他元素迭代器不受影响

正确答案：C

题目解析：

当 vector 进行删除操作（erase，pop_back）后，指向删除点的迭代器全部失效；指向删除点后面的元素的迭代器也将全部失效

62 以下哪项不属于 STL container（）

A stack

B queue

C multimap

E string

正确答案：D

63 考虑以下二分查找的代码：

```
#include <stdio.h>

int bsearch(int array[], int n, int v)

{

        int left, right, middle;

        left = 0, right = n - 1;

        while (left <= right) {

                middle = left + (right - left) / 2;

                if (array[middle] > v) {

                        right = middle;

                }

                else if (array[middle] < v) {

                        left = middle;

                }

                else {

                        return middle;

                }
```

```
        }
        return -1;
```

}对于输入 array 为：{2, 6, 8, 10, 13, 25, 36, 45, 53, 76, 88, 100, 127}，n = 13, v = 127 时，

运行 bsearch 函数，while 循环调用的次数为____。

A 1

B 3

C 4

D 5

E 6

F 无数次

链接：
https://www.nowcoder.com/questionTerminal/dc0a96436543496d938962615e3b4436

来源：牛客网

解析

答案 F

本题是一个坑，这是不正确的二分查找法，当中间的没有找到的时候，left 和 right 下标应该左移或者右移（left++,right--），实质是（ right = middle-1，left=middle+1），才不会出现除以二向下取值的时候出现无限循环。

64 以下对结构体类型变量的定义中，不正确的是（）

A· typedef struct aa

{

int n;

float m;

} AA;

AA td1;

B #define AA struct aa

AA {

int n;

float m;

} td1;

C struct

{

int n;

float m;

} aa;

stuct aa td1;

D struct

{

int n;

float m;

} td1;

解析

【解释】在答案 C 中，aa 是结构体变量，不是结构体类型名，不能再利用 aa 来定义

别的结构体变量，而且 stuct 也是非法的。所以应选择 C。

65 以下关于类占用内存空间的说法正确的是

A 类所占内存的大小是由成员变量（静态变量除外）决定的

B 空类的内存大小是 1 个字节

C 类中无论有多少个虚函数，只会多占一个虚表指针空间

D 子类的内存大小等于父类的内存大小加上子类独有成员变量的内存大小

正确答案：B C

66 已知 ii，j 都是整型变量，下列表达式中，与下标引用 X[ii][j]不等效的是（）。

A *（X[ii]+j）

B *(X+ii)[j]

C *(X+ii+j)

D *(*(X+ii)+j)

正确答案：B C

67 对于下面程序

```
#include <stdio.h>

int main() {

        int a = 5, b = 0, c = 0;

        if (a = b + c) printf("***\n");

        else printf("$$$\n");

        return 0;

}
```

说法正确的是（）

A 有语法错不能通过编译

B 可以通过编译但不能通过连接

C 输出***

D 输出$$$

【解释】if 后面的表达式可以是任何类型的表达式，当然可以是赋值表达式，所以答

案 A 和 B 是错误的，因赋值表达式的值为 0（即为假），所以执行 else 后的语句，输出$$$。

故正确答案是 D。

68 在 32 位环境下，以下程序的输出结果是？

```
#include<iostream>

using namespace std;


class Base

{

public:

        virtual int foo(int x)

        {

                return x * 10;

        }


        int foo(char x[14])

        {

                return sizeof(x) + 10;

        }

};


class Derived : public Base

{

        int foo(int x)

        {
```

```
                return x * 20;
        }

        virtual int foo(char x[10])
        {
                return sizeof(x) + 20;
        }
};

int main()
{
        Derived stDerived;
        Base* pstBase = &stDerived;

        char x[10];
        printf("%d\n", pstBase->foo(100) + pstBase->foo(x));

        return 0;
}
```

在 32 位环境下，以上程序的输出结果是？

A 2000

B 2004

C 2014

D 2024

链接：
https://www.nowcoder.com/questionTerminal/daa5422cb468473c9e6e75cc98b771de

来源：牛客网

答案： C

解释：pstBase->foo(100) + pstBase->foo(x)

pstBase->foo(100)　调用继承的函数，因为父函数有 virtual 修饰，被子类的同名函数覆盖。

pstBase->foo(x)　　调用父类函数，因为 父函数没有有 virtual 修饰。int foo(char x[14]) 参数传的是指针，指针是 4 个字节。

所以结果是 2014。

69 下面程序的输出为（）

```
#include <iostream>

#include <stack>

using namespace std;


int main() {

	stack<int> st;

	int pos = 1;

	while (pos <= 3) {

		st.push(pos++);

	}

	cout << st.top();

	while (pos <= 5) {

		st.push(pos++);

	}

	while (!st.empty()) {

		cout << st.top();

		st.pop();

	}
```

```
        return 0;
}
```

上述程序的输出为（）

A 35421

B 354321

C 12453

D 123453

链接：
https://www.nowcoder.com/questionTerminal/c6d0f89ba8514904a799b101c414a4bc

来源：牛客网

正确答案：B

题目解析：

C++中 stack 为栈数据结构，后入先出。top 用于查看栈顶元素，不弹出。

第一次存入数据分别为 1,2,3，存入后自顶向下 st 中的元素为 3,2,1。此时 top 输出为 3。

第二次存入数据分别为 4,5，存入后自顶向下 st 中的元素为 5,4,3,2,1。

出栈顺序为 5,4,3,2,1，因此输出为 354321，B 选项正确。

70 下列 for 循环的循环体执行次数为

for(int i=10, j=1; i=j=0; i++, j--)

A 0

B 1

C 无限

D 以上都不对

答案 A

条件为假

71 有如下程序段：

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void GetMemeory(char* p) {
        p = (char *)malloc(100);
}

void Test() {
        char* str = NULL;
        GetMemeory(str);
        strcpy(str,"Thunder");
        strcat(str + 2,"Downloader");
        printf(str);
}
```

请问运行 Test 函数结果是：

A Thunder Downloader

B under Downloader

C Thunderownloader

D 程序崩溃

链接：
https://www.nowcoder.com/questionTerminal/681153b22f4d4622a7bf63cb6670aa23

来源：牛客网

答案：D

思路：

1. str 为一个指针，但实际上为 int 类型，传入函数内部并不会发生任何改变
2. GetMemory 函数执行完成后，str 仍然指向 NULL，所以赋值时回奔溃
3. 正确的做法应该使用双指针

72 括号处应填写（）。

```
#include <iostream>

using namespace std;


class A

{

public:

	int m;

	int* p;

};


int main()

{

	A s;

	s.m = 10;

	cout << s.m << endl; //10

	s.p = &s.m;
```

```
        () = 5;
        cout << s.m << endl; //5
        return 0;
}
```

A s.p = 5

B s->p = 5

C s.*p = 5

D *s.p = 5

正确答案：D

73 下面是对 s 的初始化，其中不正确的是（　　）

A char s[5]={"abc"};

B char s[5]={'a','b','c'};

C char s[5]=" ";

D char s[5]="abcdef";

正确答案：D

74 有以下语句定义

```
#include <iostream>
int main()
{
        int x  = 5;
        const int* const p = &x;
        const int & q = x;
        int const * next = &x;
        const int * j = &x;
}
```

则有语法错误的是（）

A * p =1;

B q++;

C next++;

D (*j)++;

正确答案：A B D

1>p 是指向常量的常量指针,(*p)是常量不能再赋值,

2>q 是常量的引用,不能赋值

3>next 是指向常量的指针,next 本身可以改变

4>j 是指向常量的指针,值不能改变

75 下面关于 const 正确的是（）

A 欲阻止一个变量被改变，可以使用 const 关键字。

B 在定义该 const 变量时，不用将其初始化。

C 在一个函数声明中，const 可以修饰形参，表明它是一个输入参数，在函数内部不能改变其值

D 对于类的成员函数，有时候必须指定其返回值为 const 类型，以使得其返回值不为“左值”

正确答案：A C D

在定义该 const 变量时，不用将其初始化（默认值）。只能在定义的时候初始化，以后不能再赋值。如果定义的时候未初始化，则以后也不能赋值

76 在 C 语言中，函数的隐含存储类别是（）

A auto

B static

C extern

D 无存储类别

【解释】如果在函数定义中没有说明 extern 或 static，则隐含为 extern。

正确答案是 C。

77 应用的 C 函数 main 函数原型定义是下（）

A void main(void)

B int main(void *arg)

C void main(void *arg)

D int main(int argc, const char *argv[])

正确答案：D

78 若数组 S[1..n]作为两个栈 S1 和 S2 的存储空间，对任何一个栈，只有当[1..n]全满时才不能进行进栈操作。为这两个栈分配空间的最佳方案是（）

A S1 的栈底位置为 0，S2 的栈底位置为 n+1

B S1 的栈底位置为 0，S2 的栈底位置为 n/2

C S1 的栈底位置为 1，S2 的栈底位置为 n

D S1 的栈底位置为 1，S2 的栈底位置为 1

正确答案：C

79 用数组 M[0..N-1] 用来表示一个循环队列， FRONT 指向队头元素，REAR 指向队尾元素的后一个位置，则当前队列中的元素个数是几个？（注：队列总的元素数不会超过队列大小）

A rear-front+1

B rear-front-1

C (rear-front+n)%n

D (rear-front)%n

解析

队列中 rear 指向为下一个地址，rear-front=已经存入的个数 。

由题意知队列为循环队列，rear 的序号可能就比 front 序号小，所以需要+n 再%n。

正确答案为 C。

80 若执行下面的程序时，从键盘上输入 5 ，则输出是（）

```
#include <iostream>

int main(int argc, char** argv)

{


	int x;

	scanf("%d", &x);

	if (x++ > 5)

		printf("%d\n", x);

	else

		printf("%d\n", x--);


	return 0;
```

}

A 7

B 4

C 6

D 5

解析

C

进行 if 判断时由于++在后所以先进行判断后++，不满足条件进入 else，此时 x++变为 6

进入 else 后由于--在后所以先打印后--，打印出的 x 为 6，然后 x--变为 5

81 关于重载和多态正确的是

· A 如果父类和子类都有相同的方法,参数个数不同,将子类对象赋给父类后,由于子类继承于父类,所以使用父类指针

调用父类方法时,实际调用的是子类的方法

·B 选项全部都不正确

C 重载和多态在 C++面向对象编程中经常用到的方法,都只在实现子类的方法时才会使用

D class A{

void test(float a){cout<<"1";}

};

class B:public A{

void test(int b){cout<<"2";}

};

A *a=new A;

B *b=new B;

a=b;

a.test(1.1);

结果是 1·

链接：
https://www.nowcoder.com/questionTerminal/dafc816c12b7415ab1ff298d4be616d3

来源：牛客网

解析

答案是 B

A 调用的是父类的方法,C 重载也包含函数重载,D 中 a.test(1.1)是错误的,应该为(*a).test(1.1);然后在 class A 里面加 public 就对了,是这样:

```
class A{public: void test(float a){cout<<"1";}
};
```

82 在同一个源文件中，外部变量与局部变量同名，则在局部变量的作用范围内，外部变量不起作用

A TRUE

B FALSE

正确答案：B

83 有以下程序，若运行时输入：1 2 3<回车>，则输出结果是_____。

```
#include <iostream>

int main()

{
```

```
        int a[3][2] = { 0 }, (*ptr)[2], i, j;
        for (i = 0; i < 2; i++)
        {
                ptr = a + i;
                scanf("%d", ptr);
        }

        for (i = 0; i < 3; i++)
        {
                for (j = 0; j < 2; j++)
                        printf("%2d", a[i][j]);
                printf("\n");
        }
}
```

A 产生错误信息

B 1 0

2 0

0 0

C 1 2

3 0

0 0

D 1 0

2 0

3 0

正确答案：B

84 c/c++中，关于类的静态成员的不正确描述是（）

A 静态成员不属于对象，是类的共享成员

B c++11 之前，非 const 的静态数据成员要在类外定义和初始化

C 静态成员函数不拥有 this 指针，需要通过类参数访问对象成员

D 只有静态成员函数可以操作静态数据成员

正确答案：D

解析：

A. 静态成员是与类本身直接相关，而不是与类的各个对象保持关联，故类的静态成员不属于对象。但可以用类的对象、引用或指针来访问静态成员。

B. 和其他函数一样，静态成员函数可以再类内部和外部定义。初始化一般在外部，在内部可以为静态成员提供一个 const 整数类型的类内初始化值。

C. 静态成员不与任何对象绑定，不包含 this 指针，故其也不能声明为 const 的。

D. 静态数据成员属于类，非静态成员函数也可访问。

85 以下表达式选择结果是（）

```
int a = 0;

int b = (a = -1) ? 2 : 3;

int c = (a = 0) ? 2 : 3;
```

A b=2, c=2

B b=3, c=3

C b=2, c=3

D b=3, c=2

正确答案：C

86 C++当中，以下关于抽象类的说法正确的有( )

A 抽象类只能用作其他类的基类

B 不能使用抽象类定义对象

C 抽象类不能用作参数类型、函数返回类型或显式转换的类型

D 抽象类不能有构造函数和析构函数

正确答案：A B C

解析：

链接：
https://www.nowcoder.com/questionTerminal/f920f00fb1e3450ca1f326559338 36d5

来源：牛客网

抽象类的定义

纯虚函数是在基类中声明的虚函数，它在基类中没有定义，但要求任何派生类都要定义自己的实现方法。在基类中实现纯虚函数的方法是在函数原型后加“=0”，有虚函数的类就叫做抽象类。

抽象类有以下几个特点：

（1）抽象类只能用作其他类的基类，不能建立抽象类对象。

（2）抽象类不能用作参数类型、函数返回类型或显式转换的类型。

（3）可以定义指向抽象类的指针和引用，此指针可以指向它的派生类，进而实现多态性

87 类 B 是类 A 的公有派生类，类 A 和类 B 中都定义了虚函数 func(),p 是一个指向类 A 对象的指针，则 p->A::func()将（）？

A 调用类 B 中函数 func()

B 即调用类 A 中函数，也调用类 B 中的函数

C 调用类 A 中函数 func()

D 根据 p 所指的对象类型而确定调用类 A 中或类 B 中的函数 func()

正确答案：C

88 以下程序段的输出结果是

```
#include<stdio.h>

void main() {

        char s[] = "\\123456\123456\t";

        printf("%d\n", strlen(s));

}
```

A 12

B 13

C 16

D 以上都不对

链接：
https://www.nowcoder.com/questionTerminal/055bba09489348439fac5c76ed9c98d8

来源：牛客网

【正确答案】A

【解析】strlen() 函数用于获取字符串的长度（即字符串中字符的个数，不包括 \0）。\\、\123、\t 是转义字符。所以长度是 12。

\\ 表示单个字符\

\123 表示八进制的 ASCII 码值为 123 对应的字符

\t 表示制表符

89 有以下程序

```
#include <iostream>
int main()
{
	int i;
	for (i = 0; i < 3; i++)
	{
		switch (i)
		{
		case 0:
			printf("%d", i);
		case 2:
			printf("%d", i);
		default:
			printf("%d", i);
		}
	}
	return 0;
}
```

程序运行后的输出结果是（）

A 022111

B 021021

C 000122

D 012

正确答案：C

解析：

本题关键点：循环，没有 break。

没有 break，swith 会从满足条件处开始执行到最后结束。

90 下面程序的功能是输出数组的全排列,选择正确的选项,完成其功能。

```
#include <iostream>

using namespace std;

void perm(int list[], int k, int m)

{

        if ()

        {

                copy(list, list + m, ostream_iterator<int>(cout, " "));

                cout << endl;

                return;

        }

        for (int i = k; i <= m; i++)

        {

                swap(&list[k], &list[i]);

                ();

                swap(&list[k], &list[i]);

        }

}
```

A k!=m 和 perm（list，k+1，m）

B k==m 和 perm（list，k+1，m）

C k!=m 和 perm（list，k，m）

D k==m 和 perm（list，k，m）

正确答案：B

91 以下程序执行后控制台输出为（）

```
#include <stdio.h>

void g1(int* a, int n, int i) {
	while (2 * i <= n) {
		int j = 2 * i;
		int v = a[j - 1];
		if (j < n && v < a[j]) {
			v = a[j];
			j += 1;
		}
		if (a[i - 1] < v) {
			int tmp = a[i - 1];
			a[i - 1] = v;
			a[j - 1] = tmp;
			i = j;
		}
 else {
 break;
		}
	}
}
int g2(int * a, int n, int m) {
	int i;
	for (i = n / 2; i > 0;--i)
		g1(a, n, i);
```

```
        for (i = 0; i <n && a[i] != m; ++i);
        int j = 0;
        for (++i; i > 0; i /= 2)
                ++j;
        return j;
}
int main(int argc, char* argv[]) {
        int a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 };
        int n = sizeof(a) / sizeof(a[0]);
        printf("%d", g2(a, n, 8));
        return 0;
}
```

A 3

B 4

C 5

D 7

正确答案：B

92 关于 C++中的友元函数说法正确的是（　　）

A 友元函数需要通过对象或指针调用

B 友元函数是不能被继承的

C 友元函数没有 this 指针

D 友元函数破坏了继承性机制

正确答案：B C

解释

链接：
https://www.nowcoder.com/questionTerminal/ddeb509c86434661bc0ccc9f37bdcbfb

来源：牛客网

友元函数是一种在类外定义，在类内特殊声明(加关键字 friend)，并且可以在类外访问类的所有成员的非成员函数。友元函数相对于普通函数，增加了访问类成员的权利。A、友元函数可以像普通函数一样直接调用，不需要通过对象或指针；BC、友元函数不是成员函数，所以不能被继承，也同样没有 this 指针；D、由于友元函数和普通函数的区别仅仅是具有访问类成员的权利，和继承性机制没有关系

93 运行以下 C 语言代码，输出的结果是（）

```
#include <stdio.h>

int main()

{

	char * str[3] = { (char*)"stra",(char*)"strb",(char*)"strc" };


	char * p = str[0];

	int i = 0;

	while (i < 3)

	{

		printf("%s ", p++);

		i++;

	}

	return 0;

}
```

A stra strb strc

B s t r

C stra tra ra

D s s s

正确答案：C

94 有如下程序段：

```
#include <stdio.h>

class A

{

public:

    A()

    {

        printf("1");

    }

    A(A& a)

    {

        printf("2");

    }

    A& operator=(const A& a)

    {

        printf("3");

        return *this;

    }

};

int main()

{

    A a;

    A b = a;

}
```

则程序输出为：

A 12

B 13

C 无法确定

D 编译出错

链接：
https://www.nowcoder.com/questionTerminal/f2b9ad6d2d904314bbfe5faa18ef0ce2
来源：牛客网

答案 A

解析

A a,定义一个对象，毫无疑问调用构造函数

A b=a，这是定义了对象 b，且以 a 对 b 进行初始化，这个时候需要调用拷贝构造函数。

如果写成 A a；A b；b=a；则是调用后面重载的赋值函数，这种情况应该输出 113。

这个题主要考察赋值和初始化的区别。

95

```
#include <stdio.h>


void swap_int(int* a, int* b) {
        *a = *a + *b;
        *b = *a - *b;
        *a = *a - *b;
}


int main()
{
```

```
    int m = 2112340000, n = 2100001234;
    swap_int(&m, &n);

}
```

以下说法正确的是：

A 结果不正确，因为会溢出，用位与的方式就没问题

B 结果正确，即使会溢出

C 结果正确，不会溢出

D 其他选项都不对

正确答案：B

解析

链接：
https://www.nowcoder.com/questionTerminal/55bf0d019af549a48f5dc72fc9221377

来源：牛客网

以 signed char 为例 a=127 b=1 为例

a=a+b 时 溢出 此时 a=-128

b=a-b -128-1 溢出 b=127

a=a-b -128-127 溢出 为 1 仍然能正确交换。

96 派生类对象可以访问基类成员中的（）？

A 公有继承的私有成员

B 私有继承的公有成员

C 公有继承的保护成员

D 以上都错

【正确答案】D

【解析】无论是什么继承方式，派生类的对象只能访问基类中的公有成员。

97 下面的程序的输出是什么？

```
#include<stdio.h>

#include<string.h>

int main(void) {

        int n;

        char y[10] = "ntse";

        char* x = y;

        n = strlen(x);

        *x = x[n];

        x++;

        printf("x=%s,", x);

        printf("y=%s\n", y);

        return 0;

}
```

A x=atse,y=

B x=tse,y=

C x=atse,y=e

D x=tse,y=e

链接：
https://www.nowcoder.com/questionTerminal/6ba5e61f344a413ea37a429f3c065efb

来源：牛客网

答案：选 B

答案解析：n = strlen(x)，此时 n=4，因为 x 指向 y 数组，所以 x[4]就是 y[4]='\0',那么*x=x[n]就是把 x 指向的字符串首元素改为'\0'，x++之后 x 指向第二个字符 t，所以第一个输出 x=tse，而 y 遇到第一个字符就是'\0'，所以结束，y 输出为空

98 在 C++中，下列哪一个可以做为对象继承之间的安全转换（）

A static_cast

B dynamic_cast

C const_cast

D reinterpret_cast

答案：A、B。

转换指的是通过改变一个变量的类型为别的类型从而改变该变量的表示方式。

C++标准定义了四个新的转换符：reinterpret_cast、static_cast、dynamic_cast 和 const_cast，

目的在于控制类（class）之间的类型转换。

对于选项 A，static_cast 可以用于类层次结构中基类和子类之间指针或引用的转换。

把子类的指针或引用转换成基类表示是安全的，但把基类指针或引用转换成子类指针或引用时，

由于没有动态类型检查，所以，它是不安全的。

基类和子类之间的动态类型转换一般建议使用 dynamic_cast。

static_cast 可以用作对象继承之间转换，只不过有安全隐患。因此，选项 A 正确。

对于选项 B，dynamic_cast 用于对象的指针和引用，当用于多态类型转换时，

允许隐式转换及相反的转换操作，与 static_cast 的不同之处在于，在相反的转换过程中，

dynamic_cast 会检测操作的有效性，如果返回的不是被请求的有效完整对象，则返回 null，

反之返回这个有效的对象，如果是引用返回无效时，则会抛出 bad_cast 异常。所以，选项 B 正确。

对于选项 C，const_cast 用来修改类型的 const 或 volatile 属性，具体而言，

const_cast 会操纵传递对象的 const 属性，设置或者移除该属性。所以，选项 C 错误。

对于选项 D，reinterpret_cast 用来处理无关类型之间的转换，可以转换任意一个 32 位整数，

包括所有的指针和整数。可以把任何整数转成指针，也可以把任何指针转成整数，

以及把指针转化为任意类型的指针，但不能将非 32 位的实例转成指针。所以，选项 D 错误。

所以，本题的答案为 A、B。

99 函数 fun 的声明为 int fun(int *p[4]),以下哪个变量可以作为 fun 的合法参数（）

A int a[4][4];

B int **a;

C int **a[4]

D int (*a)[4];

正确答案：B

解析

链接：
https://www.nowcoder.com/questionTerminal/e2ac8bddb9e5434a92511320221c8513

来源：牛客网

二维指针和二级指针的区别 二维数组指针形式相当于一级指针，因为它可以通过指针的偏移量访问所有的元素。 int *a[4] a 指向数组，数组里面存放的是 int*，可以看做是二级指针 int** a[4] a 指向数组，数组里面存放的是二级指针,a 可以看做三级指针 int (**a)[4] a 是一个二级指针，数组里面存放 int 型 int (*a)[4]; a 是个一级指针，数组里面存放 int 型 int(*a)[4]和 int a[4]的区别在于一个是指针，一个是数组名 记住类似于 int** (**a)[4]在 int/doulble 等 type 类型后面描述的是数组存储的类型，括号里面是指针的级别 此题需要的是二级指针