## 一、选择题

1. 以下程序段，下面说法正确的是（C）

int x = -1;

do {

x = x\*x;

}

while(!x);

A、死循环

B、循环执行二次

C、循环之行一次

D、循环之行零次

1. 如果是声明一个常指针，下面语句正确的是（C）

A、char \*p;

B、const char \*p;

C、char \*const p;

D、const char \*const p;

1. 64位平台上，使用g++编译下面程序，编译器默认进行内存对齐，下面程序的输出错误的一项是（D，内存进行8字节对齐，输出应该是16）

int main()

{

char arr[100];

char \*p = arr;

union TestUnion {

double var1;

char var2[9];

};

cout << "sizeof(arr) = " << sizeof(arr) << endl;

cout << "sizeof(p) = " << sizeof(p) << endl;

cout << "sizeof(\*p) = " << sizeof(\*p) << endl;

cout << "sizeof(TestUnion) = " << sizeof(TestUnion) << endl;

return 0;

}

1. sizeof(arr) = 100
2. sizeof(p) = 8
3. sizeof(\*p) = 1

D、sizeof(TestUnion) = 9

1. 下面func1、func2、func3、func4函数调用时，参数传递会出现几个编译错误（A，没有编译错误）

void func1(char (\*arr1)[64])

{

cout << "call func1" << endl;

}

void func2(char \*arr2[64])

{

cout << "call func2" << endl;

}

void func3(char \*\*arr3)

{

cout << "call func3" << endl;

}

void func4(char arr4[100][64])

{

cout << "call func4" << endl;

}

int main()

{

char (\*arr1)[64];

char \*arr2[64];

char \*\*arr3;

char arr4[100][64];

func1(arr4);

func2(arr3);

func3(arr2);

func4(arr1);

return 0;

}

A、0

B、1

C、2

D、3

1. 下面代码最终会输出什么（D，li.erase(it)应该改为it=li.erase(it)）

int main()

{

list<int> li = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

list<int>::iterator it = li.begin();

while(it != li.end()) {

if(\*it % 2)

li.erase(it);

else

it++;

}

list<int>::const\_iterator it\_last = li.end();

it\_last--;

for(list<int>::const\_iterator it=li.begin(); it!=li.end(); it++) {

if(it != it\_last)

cout << \*it << ", " ;

else

cout << \*it << endl;

}

return 0;

}

A、0, 2, 4, 6, 8

B、1, 3, 5, 7, 9

C、0, 1, 2, 3, 4, 5, 6, 7, 8, 9

D、segment fault

## 二、简答题

1. 指针和引用的区别
   1. 指针是一个实体，而引用仅是个别名
   2. 引用使用时无需解引用(\*)，指针需要解引用
   3. 引用只能在定义时被初始化一次，之后不可变，指针可变
   4. 引用没有 const，指针有 const
   5. 引用不能为空，指针可以为空
   6. sizeof 引用得到的是所指向的变量(对象)的大小，而sizeof 指针得到的是指针本身(所指向的变量或对象的地址)的大小
   7. 指针和引用的自增(++)运算意义不一样
   8. 从内存分配上看：程序为指针变量分配内存区域，而引用不需要分配内存区域
2. 列出一些C++11新特性，越多越好
   1. nullptr，用于表示空指针，可以解决某些函数重载时的二义性问题
   2. 使用using定义类型别名：例如typedef int INT32;可以表示为using INT32=int;
   3. auto，用于自动类型推导
   4. explicit，用于禁止构造函数进行隐式转换
3. C++的强制类型转换有哪几种，以及各自的试用场景
   1. static\_cast

用来进行比较低风险的转换，比如整形和实数型、字符型之间互相转换。不能在不同类型类型的指针之间互相转换，也不能用于整型和指针之间的互相转换，也不能用于不同类型的引用之间的转换。

* 1. reinterpret\_cast

用来进行各种不同类型的指针之间的转换、不同类型的引用之间转换、以及指针和能容纳得下指针的整数类型之间的转换。转换的时候，执行的是逐个比特拷贝的操作。

* 1. const\_cast

用来进行去除const属性的转换。将const引用转换成同类型的非const引用，将const指针转换为同类型的非const指针时用它。

* 1. dynamic\_cast

专门用于将多态基类的指针或引用，强制转换为派生类的指针或引用，而且能够检查转换的安全性。对于不安全的指针转换，转换结果返回NULL。不能用于将非多态基类的指针或引用，强制转换为派生类的指针或引用。

1. 一般情况下C++程序编译后的内存布局
   1. 栈，由编译器自动分配释放
   2. 堆，一般由程序员分配释放，或由系统回收
   3. 全局区（静态区），全局变量和静态变量是存放在一起的，初始化的全局变量和静态变量在一块区域，未初始化的全局变量和静态变量在相邻的另一块区域。这部分内存在程序编译时就已经分配好，并且在程序运行期间都在，最后由系统回收。
      1. 初始化的全局变量和静态变量存放在.data段
      2. 未初始化的全局变量和静态变量存放在.bss段
   4. 常量区，专门用来存放一些常量，如字符串常量，最后由系统回收
   5. 代码区，存放二进制代码
2. new/delete与malloc()/free()的异同
   1. malloc() 与 free() 是C语言的标准库函数，new/delete 是C++的运算符，他们都可以用来申请和释放内存
   2. malloc()和free()不在编译器控制权限之内, 不能把构造函数和析构函数的任务强加给他们
      1. new：分配空间->初始化->构造函数
      2. malloc：分配空间