第一部分:简答题

1. 一下C++中static\_cast 和 dynamic\_cast 的区别。

答：static\_cast 用于有比较明确定义的变换，包括不需要强制转换的变换。

    dynamic\_cast 适用于类型安全的向下转换，常用在继承中的父类指针向子类指针的转换。若转换成功则返回改类型的指针，若失败，则返回NULL。

1. Struct 和class 的区别。

答：struct是一种结构体类型的标志，它和C++中的Class很相似。

  C++中两者的唯一区别在于：Class定义中默认情况下的成员都是私有的，而结构Struct                  定义中默认情况下的成员都是公有的。

  在C中，结构中不能有成员函数，而在C++中可以有成员函数.

1. 是不是一个父类写了一个virtual 函数，如果子类覆盖它的函数不加virtual ,也能实现多态?

答：virtual修饰符会被隐形继承的。private 也被继承，只是派生类没有访问权限而已。virtual可加可不加。子类的空间里有父类的所有变量(static除外)。同一个函数只存在一个实体(inline除外)。子类覆盖它的函数不加virtual ,也能实现多态。在子类的空间里，有父类的私有变量。私有变量不能直接访问。

1. C和C++有什么不同?（从机制，适用领域等方面说说）

答：从机制上：C是面向过程的（但c也可以编写面向对象的程序）；C++是面向对象的，提供了类。但是，C++编写面向对象的程序比C容易

从适用的方向：C适合要求代码体积小的，效率高的场合，如嵌入式；C++适合更上层的，复杂的； llinux核心大部分是C写的，因为它是系统软件，效率要求极高。

从名称上也可以看出，C++比C多了+，说明C++是c的超集；那为什么不叫C+而叫C++呢，是因为C++比C来说扩充的东西太多了，所以就在C后面放上两个+；于是就成了c++

C语言是结构化编程语言，C++是面向对象编程语言。

C++侧重于对象而不是过程，侧重于类的设计而不是逻辑的设计。

1. 请说出const与#define 相比，有何优点？

答：

1） const 常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查。而对后者只进行字符替换，没有类型安全检查，并且在字符替换可能会产生意料不到的错误。

2） 有些集成化的调试工具可以对const 常量进行调试，但是不能对宏常量进行调试。

1. 简述数组与指针的区别？

答：数组要么在静态存储区被创建（如全局数组），要么在栈上被创建。指针可以随时指向任意类型的内存块。

(1)修改内容上的差别

char a[] = “hello”;

a[0] = ‘X’;

char \*p = “world”; // 注意p 指向常量字符串

p[0] = ‘X’; // 编译器不能发现该错误，运行时错误

(2) 用运算符sizeof 可以计算出数组的容量（字节数）。sizeof(p),p 为指针得到的是一个指针变量的字节数，而不是p 所指的内存容量。C++/C 语言没有办法知道指针所指的内存容量，除非在申请内存时记住它。注意当数组作为函数的参数进行传递时，该数组自动退化为同类型的指针。

char a[] = “hello world”;

char \*p = a;

cout<< sizeof(a) << endl; // 12 字节

cout<< sizeof(p) << endl; // 4 字节

计算数组和指针的内存容量

void Func(char a[100])

{

cout<< sizeof(a) << endl; // 4 字节而不是100 字节

}

1. 类成员函数的重载、覆盖和隐藏区别？

答：

a.成员函数被重载的特征：

（1）相同的范围（在同一个类中）；

（2）函数名字相同；

（3）参数不同；

（4）virtual 关键字可有可无。

b.覆盖是指派生类函数覆盖基类函数，特征是：

（1）不同的范围（分别位于派生类与基类）；

（2）函数名字相同；

（3）参数相同；

（4）基类函数必须有virtual 关键字。

c.“隐藏”是指派生类的函数屏蔽了与其同名的基类函数，规则如下：

（1）如果派生类的函数与基类的函数同名，但是参数不同。此时，不论有无virtual关键字，基类的函数将被隐藏（注意别与重载混淆）。

（2）如果派生类的函数与基类的函数同名，并且参数也相同，但是基类函数没有virtual 关键字。此时，基类的函数被隐藏（注意别与覆盖混淆）

1. 面向对象的三个基本特征，并简单叙述之？

答：

1. 封装：将客观事物抽象成类，每个类对自身的数据和方法实行protection(private, protected,public)

2. 继承：广义的继承有三种实现形式：实现继承（指使用基类的属性和方法而无需额外编码的能力）、可视继承（子窗体使用父窗体的外观和实现代码）、接口继承 （仅使用属性和方法，实现滞后到子类实现）。前两种（类继承）和后一种（对象组合=>接口继承以及纯虚函数）构成了功能复用的两种方式。

3. 多态：是将父对象设置成为和一个或更多的他的子对象相等的技术，赋值之后，父对象就可以根据当前赋值给它的子对象的特性以不同的方式运作。简单的说，就是一句话：允许将子类类型的指针赋值给父类类型的指针。

9. 请简单描述Windows内存管理的方法。

答：

内存管理是操作系统中的重要部分，两三句话恐怕谁也说不清楚吧～～

我先说个大概，希望能够抛砖引玉吧

当程序运行时需要从内存中读出这段程序的代码。代码的位置必须在物理内存中才能被运行，由于现在的操作系统中有非常多的程序运行着，内存中不能够完全放下，所以引出了虚拟内存的概念。把哪些不常用的程序片断就放入虚拟内存，当需要用到它的时候在load入主存（物理内存）中。这个就是内存管理所要做的事。内存管理还有另外一件事需要做：计算程序片段在主存中的物理位置，以便CPU调度。

内存管理有块式管理，页式管理，段式和段页式管理。现在常用段页式管理

块式管理：把主存分为一大块、一大块的，当所需的程序片断不在主存时就分配一块主存空间，把程 序片断load入主存，就算所需的程序片度只有几个字节也只能把这一块分配给它。这样会造成很大的浪费，平均浪费了50％的内存空间，但时易于管理。

页式管理：把主存分为一页一页的，每一页的空间要比一块一块的空间小很多，显然这种方法的空间利用率要比块式管理高很多。

段式管理：把主存分为一段一段的，每一段的空间又要比一页一页的空间小很多，这种方法在空间利用率上又比页式管理高很多，但是也有另外一个缺点。一个程序片断可能会被分为几十段，这样很多时间就会被浪费在计算每一段的物理地址上（计算机最耗时间的大家都知道是I/O吧）。

段页式管理：结合了段式管理和页式管理的优点。把主存分为若干页，每一页又分为若干段。好处就很明显，不用我多说了吧。

各种内存管理都有它自己的方法来计算出程序片断在主存中的物理地址，其实都很相似。

这只是一个大概而已，不足以说明内存管理的皮毛。无论哪一本操作系统书上都有详细的讲解

10.main 主函数执行完毕后，是否可能会再执行一段代码，给出说明？

答：可以，可以用\_onexit 注册一个函数，它会在main 之后执行int fn1(void), fn2(void), fn3(void), fn4 (void);

void main( void )

{

String str(“zhanglin”);

\_onexit( fn1 );

\_onexit( fn2 );

\_onexit( fn3 );

\_onexit( fn4 );

printf( “This is executed first.\n” );

}

int fn1()

{

printf( “next.\n” );

return 0;

}

int fn2()

{

printf( “executed ” );

return 0;

}

int fn3()

{

printf( “is ” );

return 0;

}

int fn4()

{

printf( “This ” );

return 0;

}

Csdn文档摘要：

The \_onexit function is passed the address of a function (func) to be called when the program terminates normally. Successive calls to \_onexit create a register of functions that are executed in LIFO (last-in-first-out) order. The functions passed to \_onexit cannot take parameters.

11.const 符号常量；

(1)const char \*p

(2)char const \*p

(3)char \* const p

说明上面三种描述的区别.

答：

如果const位于星号的左侧，则const就是用来修饰指针所指向的变量，即指针指向为常量；

如果const位于星号的右侧，const就是修饰指针本身，即指针本身是常量。

(1)const char \*p

一个指向char类型的const对象指针，p不是常量,我们可以修改p的值，使其指向不同的char，但是不能改变它指向非char对象，如：

const char \*p;

char c1=’a’;

char c2=’b’;

p=&c1;//ok

p=&c2;//ok

\*p=c1;//error

(3)char \* const p

此时\*p可以修改，而p不能修改。

(4)const char \* const p

这种是地址及指向对象都不能修改。

12.下面是C语言中两种if语句判断方式。请问哪种写法更好？为什么？

答：

int n;

if (n == 10) // 第一种判断方式

if (10 == n) // 第二种判断方式

如果少了个=号,编译时就会报错,减少了出错的可能行,可以检测出是否少了=

13. \*p++ 自增p 还是p 所指向的变量?

答:

后缀++ 和– 操作符本质上比前缀一目操作的优先级高, 因此\*p++ 和\*(p++) 等价, 它自增p 并返回p 自增之前所指向的值。

要自增p 指向的值, 使用(\*p)++, 如果副作用的顺序无关紧要也可以使用++\*p。

14. #pragma 是什么, 有什么用？

答:

#pragam 指令提供了一种单一的明确定义的 “救生舱”, 可以用作各种 (不可移植的) 实现相关的控制和扩展:

源码表控制、结构压缩、警告去除 (就像 lint 的老 /\* NOTREACHED \*/注释), 等等。

15.“#pragma once” 是什么意思?我在一些头文件中看到了它。

答:

这是某些预处理器实现的扩展用于使头文件自我识别; 它跟#ifndef技巧等价, 不过移植性差些。

16. 进程间通信的方式有?

答：

进程间通信的方式有 共享内存， 管道 ，Socket ，消息队列 , DDE等

17. 如何打印出当前源文件的文件名以及源文件的当前行号？

答：

cout << \_\_FILE\_\_ ;

cout<<\_\_LINE\_\_ ;

\_\_FILE\_\_和\_\_LINE\_\_是系统预定义宏，这种宏并不是在某个文件中定义的，而是由编译器定义的。

18. 如何判断一段程序是由C 编译程序还是由C++编译程序编译的？

答：#ifdef \_\_cplusplus

cout<<”c++”;

#else

cout<<”c”;

#endif

19. New delete 与malloc free 的联系与区别?

答：

都是在堆(heap)上进行动态的内存操作。用malloc函数需要指定内存分配的字节数并且不能初始化对象，new 会自动调用对象的构造函数。delete 会调用对象的destructor，而free 不会调用对象的destructor.

20.介绍一下STL，详细说明STL如何实现vector。

答:

     STL (标准模版库，Standard Template Library)它由容器算法迭代器组成。

     STL有以下的一些优点：

     可以方便容易地实现搜索数据或对数据排序等一系列的算法；

     调试程序时更加安全和方便；

     即使是人们用STL在UNIX平台下写的代码你也可以很容易地理解（因为STL是跨平台的）。

     vector实质上就是一个动态数组，会根据数据的增加,动态的增加数组空间。

21. 指针和引用有什么分别；如果传引用比传指针安全，为什么？如果我使用常量指针难道不行吗？

答：

     (1) 引用在创建的同时必须初始化，即引用到一个有效的对象；而指针在定义的时候不必初始化，可以在定义后面的任何地方重新赋值．

     (2) 不存在ＮＵＬＬ引用，引用必须与合法的存储单元关联;而指针则可以是NULL.

     (3) 引用一旦被初始化为指向一个对象，它就不能被改变为另一个对象的引用；而指针在任何时候都可以改变为指向另一个对象．给引用赋值并不是改变它和原始对象的绑定关系．

     (4) 引用的创建和销毁并不会调用类的拷贝构造函数

     (5) 语言层面，引用的用法和对象一样；在二进制层面，引用一般都是通过指针来实现的，只不过编译器帮我们完成了转换.

     不存在空引用，并且引用一旦被初始化为指向一个对象，它就不能被改变为另一个对象的引用，显得很安全。

     const 指针仍然存在空指针，并且有可能产生野指针.

     总的来说：引用既具有指针的效率，又具有变量使用的方便性和直观性．

22. 构造函数可否是虚汗数，为什么？析构函数呢，可否是纯虚的呢？

答：

    构造函数不能为虚函数，要构造一个对象，必须清楚地知道要构造什么，否则无法构造一个对象。

     析构函数可以为纯虚函数。

第二部分:阅读判断题

1. #include “stdafx.h”

#define SQR(X) X\*X

int main(int argc, char\* argv[])

{

int a = 10;

int k = 2;

int m = 1;

a /= SQR(k+m)/SQR(k+m);

printf(“%d\n”,a);

return 0;

}

这道题目的结果是什么啊?

答：define 只是定义而已，在编择时只是简单代换X\*X而已，并不经过算术法则的

a /= (k+m)\*(k+m)/(k+m)\*(k+m);

=>a /= (k+m)\*1\*(k+m);

=>a = a/9;

=>a = 1;

2. 下面的代码有什么问题？

void DoSomeThing(…)

{

char\* p;

…

p = malloc(1024); // 分配1K的空间

if (NULL == p)

return;

…

p = realloc(p, 2048); // 空间不够，重新分配到2K

if (NULL == p)

return;

…

}

答：

p = malloc(1024);     应该写成： p = (char \*) malloc(1024);

        没有释放p的空间，造成内存泄漏。

3. 写出运行结果:

{// test1

    char str[] = “world”; cout << sizeof(str) << “: “;

    char \*p    = str;     cout << sizeof(p) << “: “;

    char i     = 10;      cout << sizeof(i) << “: “;

    void \*pp   = malloc(10); cout << sizeof(p) << endl;

}

答：6：4：1：4

4. 下面的代码有什么问题？并请给出正确的写法。

void DoSomeThing(char\* p)

{

char str[16];

int n;

assert(NULL != p);

sscanf(p, “%s%d”, str, n);

if (0 == strcmp(str, “something”))

{

…

}

}

答:

sscanf(p, “%s%d”, str, n);   这句该写成： sscanf(p, “%s%d”, str, &n);

5.  int i=10, j=10, k=3; k\*=i+j; k最后的值是?

答：60

6.

struct A

{

char t:4;

char k:4;

unsigned short i:8;

unsigned long m;

}

sizeof(A)=?（不考虑边界对齐）

答：7

struct CELL             // Declare CELL bit field

{

   unsigned character : 8; // 00000000 ????????

   unsigned foreground : 3; // 00000??? 00000000

   unsigned intensity : 1; // 0000?000 00000000

   unsigned background : 3; // 0???0000 00000000

   unsigned blink      : 1; // ?0000000 00000000

} screen[25][80];       // Array of bit fields

二、位结构

    位结构是一种特殊的结构, 在需按位访问一个字节或字的多个位时, 位结构

比按位运算符更加方便。

    位结构定义的一般形式为:

     struct位结构名{

          数据类型 变量名: 整型常数;

          数据类型 变量名: 整型常数;

     } 位结构变量;

    其中: 数据类型必须是int(unsigned或signed)。 整型常数必须是非负的整

数, 范围是0~15, 表示二进制位的个数, 即表示有多少位。

    变量名是选择项, 可以不命名, 这样规定是为了排列需要。

    例如: 下面定义了一个位结构。

     struct{

          unsigned incon: 8; /\*incon占用低字节的0~7共8位\*/

          unsigned txcolor: 4;/\*txcolor占用高字节的0~3位共4位\*/

          unsigned bgcolor: 3;/\*bgcolor占用高字节的4~6位共3位\*/

          unsigned blink: 1; /\*blink占用高字节的第7位\*/

     }ch;

    位结构成员的访问与结构成员的访问相同。

    例如: 访问上例位结构中的bgcolor成员可写成:

      ch.bgcolor

    注意:

    1. 位结构中的成员可以定义为unsigned, 也可定义为signed, 但当成员长

度为1时, 会被认为是unsigned类型。因为单个位不可能具有符号。

    2. 位结构中的成员不能使用数组和指针, 但位结构变量可以是数组和指针,

如果是指针, 其成员访问方式同结构指针。

    3. 位结构总长度(位数), 是各个位成员定义的位数之和, 可以超过两个字

节。

    4. 位结构成员可以与其它结构成员一起使用。

    例如:

     struct info{

          char name[8];

          int age;

          struct addr address;

          float pay;

          unsigned state: 1;

          unsigned pay: 1;

          }workers;

    上例的结构定义了关于一个工人的信息。其中有两个位结构成员, 每个位结

构成员只有一位, 因此只占一个字节但保存了两个信息, 该字节中第一位表示工

人的状态, 第二位表示工资是否已发放。由此可见使用位结构可以节省存贮空间。

7.下面代码有什么问题?

Void test3(char\* str1)

{

char string[10];

if(strlen(str1)<=10)

{

    strcpy(string, str1);

}

}

答：数组越界

strcpy拷贝的结束标志是查找字符串中的\0 因此如果字符串中没有遇到\0的话 会一直复制，直到遇到\0,上面的123都因此产生越界的情况

建议使用 strncpy 和 memcpy

8.下面的代码有什么问题?

class A

{

public:

A() { p=this; }

~A() { if(p!=NULL) { delete p; p=NULL; } }

A\* p;

};

答：delete 会自动调用析构函数。所以析构中调用delete引起了无限递归。

9.i最后等于多少?

int i = 1;

int j = i++;

if((i>j++) && (i++ == j)) i+=j;

答：i = 5

10. 输出下面程序结果。

#include <iostream.h>

class A

{

public:

virtual void print(void)

{

    cout<<”A::print()”<<endl;

}

};

class B:public A

{

public:

virtual void print(void)

{

   cout<<”B::print()”<<endl;

};

};

class C:public B

{

public:

virtual void print(void)

{

   cout<<”C::print()”<<endl;

}

};

void print(A a)

{

   a.print();

}

void main(void)

{

   A a, \*pa,\*pb,\*pc;

   B b;

   C c;

   pa=&a;

   pb=&b;

   pc=&c;

   a.print();

   b.print();

   c.print();

   pa->print();

   pb->print();

   pc->print();

   print(a);

   print(b);

   print(c);

}

答：

A::print()

B::print()

C::print()

A::print()

B::print()

C::print()

A::print()

A::print()

A::print()

11.

union a {

int a\_int1;

double a\_double;

int a\_int2;

};

typedef struct

{

a a1;

char y;

} b;

class c

{

double c\_double;

b b1;

a a2;

};

输出cout<<sizeof(c)<<endl;的结果?

答:

VC6环境下得出的结果是32

另:

我(sun)在VC6.0+win2k下做过试验:

short - 2

int-4

float-4

double-8

指针-4

sizeof(union)，以结构里面size最大的为union的size

对于字节对齐的问题，有几点需要注意理解：

（1）字节对齐是语言实现相关（编译器），而非C＋＋特性

（2）对于cl.exe编译器：

A。 为了编译器寻址方便快捷，默认会对结构体进行处理（实际上其它地方的数据变量也是如此），让宽度为2的基本数据类型（short等）都位于能被2整除的地址上，让宽度为4的基本数据类型（int等）都位于能被4整除的地址上，以此类推。

B。 还要注意的是，编译器会让结构体的大小是【最大成员】的整数倍。

基于上述原则我们就可以算出结构体的内存大小，当然这也是sizeof的实现原理。

编译器看到的基本类型，而非使用了sizeof取大小的复合类型，再用这个例子解释一遍：

// 【最大成员是double 8字节，而非struct b 16字节，则整个struct的大小能被8整除】

class c

{

double c\_double; // 偏移0

b b1; // b1.a1偏移8(sizeof(c\_double))

      // b1.y偏移8(sizeof(b1.a1))

a a2; // 【注意：】由于a2为double，需要放在为8倍数的偏移地址上（原则2.a）偏移8，而非sizeof(b1.y)

 // a2占有8个字节，到此整个c的内存空间为32，也刚好为最大成员占有内存8的整数倍，不需要在尾部补齐字节

};

所以：

sizeof(c) = 32

如果我们再往后面添加两个成员，如下：

class c

{

double c\_double;  // 偏移0

b b1;  // 同上

a a2; // 同上

int x; // 偏移8（sizeof(a2))  【到此地址已经为c分配了32字节了】

short k; // 偏移4(sizeof(x)) 【到此c有36字节】

// k占有2个字节，为38，但由于不能被最大成员8整除，所以在编译器在尾部补上两个字节【原则2.b】

};

则sizeof(c) = 40

有个测试代码:

union a

{

int a\_int1;

double a\_double;

int a\_int2;

};

typedef struct

{

a a1;

char y;

} b;

class c

{

public:

double c\_double;

b b1;

a a2;

short x;

int k;

};

#include<iostream>

using namespace std;

void main()

{

 c test;

 void\* p = &(test.b1.y);

 cout << “Address of test.c\_double: ” << &(test.c\_double)<< endl;

 cout << “Address of test.b1.a1: ” << &(test.b1.a1)<< endl;

 cout << “Address of test.b1.y: ” << p << endl;

 cout << “Address of test.a2: ” << &(test.a2) << endl;

 cout << “Address of test.x: ” << &(test.x) << endl;

 cout << “Adrress of test.k: ” << &(test.k) << endl;

 cout << “Sizeof class c: ” << sizeof(test) << endl;

}

12.41. 分析一下这段程序的输出 (Autodesk)

class B

{

public:

B()

{

cout<<”default constructor”<<endl;

}

~B()

{

cout<<”destructed”<<endl;

}

B(int i):data(i)   //B(int) works as a converter ( int -> instance of B)

{

cout<<”constructed by parameter ” << data <<endl;

}

private:

int data;

};

B Play( B b)

{

return b ;

}

答：

(1)                                               results:

int main(int argc, char\* argv[])      constructed by parameter 5

{                                                   destructed B(5)形参析构 //构造函数可以做默认的类型转换

B t1 = Play(5); B t2 = Play(t1);    destructed t1形参析构

return 0;　　　　　　　　　　　destructed t2　注意顺序！//t1和t2是在程序的栈区存放的，先进后出

}                                                    destructed t1

(2)                                                     results:

int main(int argc, char\* argv[])           constructed by parameter 5

{                                                        destructed B(5)形参析构

B t1 = Play(5); B t2 = Play(10);   　   constructed by parameter 10

return 0;　　　　　　　　　　　　 destructed B(10)形参析构

}                                                        destructed t2　注意顺序！

                                                          destructed t1

13.写出程序结果:

void Func(char str[100])

{

printf(“%d\n”, sizeof(str));

}

答：4,指针长度

14，求下面函数的返回值

int func(x)

{

int countx = 0;

while(x)

{

countx ++;

x = x&(x-1);

}

return countx;

}

答：x中1的个数。

PS:

这里说点闲话，前两天我参加华硕的笔试，最后一道题就是这个题，可是他是叫我写出这个程序，可我只记得使用位运算做的，具体有点遗忘了，后来没写出来，这个对我触动很深，感觉自己学的还是流于表面，没有抓到精髓。希望大家不要像我一样，学东西还是要学的透彻，不然就容易错失机会。

15，程序改错

class mml

{

private:

    static unsigned int x;

public:

    mml(){ x++; }

    mml(static unsigned int &) {x++;}

    ~mml{x–;}

pulic:

    virtual mon() {} = 0;

    static unsigned int mmc(){return x;}

    ……

};

class nnl:public mml

{

private:

    static unsigned int y;

public:

    nnl(){ x++; }

    nnl(static unsigned int &) {x++;}

    ~nnl{x–;}

public:

    virtual mon() {};

     static unsigned int nnc(){return y;}

    ……

};

代码片断:

mml\* pp = new nnl;

……….

delete pp;

答:

基类的析构函数应该为虚函数

virtual ~mml{x–;}

16，请指出下列程序中的错误并且修改

void GetMemory(char \*p){

p=(char \*)malloc(100);

}

void Test(void){

char \*str=NULL;

GetMemory(str);

strcpy(str,”hello world”);

printf(str);

}

答:错误–参数的值改变后，不会传回

GetMemory并不能传递动态内存，Test函数中的 str一直都是 NULL。

strcpy(str, “hello world”);将使程序崩溃。

修改如下:

char \*GetMemory(){

char \*p=(char \*)malloc(100);

return p;

}

void Test(void){

char \*str=NULL;

str=GetMemory();

strcpy(str,”hello world”);

printf(str);

}

方法二:void GetMemory2(char \*\*p)变为二级指针.

void GetMemory2(char \*\*p, int num)

{

\*p = (char \*)malloc(sizeof(char) \* num);

}

17. 下面关于“联合”的题目的输出？

a)

＃i nclude <stdio.h>

union

{

int i;

char x[2];

}a;

void main()

{

a.x[0] = 10;

a.x[1] = 1;

printf(“%d”,a.i);

}

答案：266 (低位低地址，高位高地址，内存占用情况是Ox010A）

b)

main()

{

union{ /\*定义一个联合\*/

int i;

struct{ /\*在联合中定义一个结构\*/

char first;

char second;

}half;

}number;

number.i=0×4241; /\*联合成员赋值\*/

printf(“%c%c\n”, number.half.first, mumber.half.second);

number.half.first=’a’; /\*联合中结构成员赋值\*/

number.half.second=’b’;

printf(“%x\n”, number.i);

getch();

}

答案： AB (0×41对应‘A’,是低位；Ox42对应‘B’,是高位）

6261 (number.i和number.half共用一块地址空间）

第三部分:编程题

1.已知类String 的原型为：class String

{

public:

String(const char \*str = NULL); // 普通构造函数

String(const String &other); // 拷贝构造函数

~ String(void); // 析构函数

String & operate =(const String &other); // 赋值函数

private:

char \*m\_data; // 用于保存字符串

};

请编写String 的上述4 个函数。

答：

//构造函数

String::String(const char \*str)

{

     //首先判断传过的字符串是不是空字符串

   if(str == NULL)

    {

       m\_data = new char[1];  //如果是NULL，直接开辟空间后内容清零

       \*m\_data = ‘\0′;

    }

   else

    {

      int  length = strlen(str);

      m\_data = new char[length + 1];  //对于new出的空间不用判断是否开辟成功，因为在编译过程中有throw()，catch()机制

      strcpy(m\_data, str);

    }

}

//析构函数

~String::String(void)

{

   delete[]  m\_data;  //这里一定要加[]，否则删除不干净

}

//拷贝构造函数

String::String(const String &other)

{

  int length = strlen(other.m\_data);

  m\_data  =  new char[length + 1];

  strcpy(m\_data, other.m\_data);

}

//赋值重载函数

String& String:operate = (const String &other)

{

  //首先检查自赋值

  if(this != &other)

  {

    //分配新的内存空间， 并拷贝内容

   char \*temp = new char[strlen(other.m\_data) + 1];

    strcpy(temp,other.m\_data);

    //记住一定要释放原有的资源

   delete [] m\_data;

    strcpy(m\_data, temp);

  }

    return \*this;

}

2.

已知strcpy 函数的原型是：

char \*strcpy(char \*strDest, const char \*strSrc);

其中strDest 是目的字符串，strSrc 是源字符串。不调用C++/C 的字符串库函数，请编写函数 strcpy

答：

因为这道题用到了const，所以用的是C++既然是用C++，那么我们可以用异常，代码如下char\* mystrcpy ( char \* StrDest , const char \*strSrc ){

        if ( StrDest == NULL || strSrc == NULL ){

                throw runtime\_error ( “StrDest == NULL || strSrc == NULL” );

        }

        char \*temp = StrDest;

        while ( (\*StrDest++ = \*strSrc++ ) != ‘\0′ )

                ;//empty

        return temp;

}

但是C++语法中有一个漏洞，那就是支持const char\*到char\*的转换（还是默认的），所以如下代码，在编译期可以通过mystrcpy ( “Hello,world” , “hello,world” );

很明显，这是不允许的，C++中我们需要防止这一类事情发生所以我们可以参照类中禁止复制和赋值的方法，声明但不实现这个函数

char\* mystrcpy ( const char\* , const char\* );

这样子，在链接期，编译器会抛出一个链接错误，我们就可以修正，所以，这道题的完整答案，如下

char\* mystrcpy ( char \* StrDest , const char \*strSrc ){

        if ( StrDest == NULL || strSrc == NULL ){

                throw runtime\_error ( “StrDest == NULL || strSrc == NULL” );

        }

        char \*temp = StrDest;

        while ( (\*StrDest++ = \*strSrc++ ) != ‘\0′ )

                ;//empty

        return temp;

}

char\* mystrcpy ( const char\* , const char\* );

//声明但不定义

1. 输入一个字符串，将其逆序后输出。（使用C++，不建议用伪码）

答：#include <iostream>

using namespace std;

void main()

{

char a[50];memset(a,0,sizeof(a));

int i=0,j;

char t;

cin.getline(a,50,’\n’);

for(i=0,j=strlen(a)-1;i<strlen(a)/2;i++,j–)

{

   t=a[i];

      a[i]=a[j];

   a[j]=t;

}

cout<<a<<endl;

}

//第二种

string str;

cin>>str;

str.replace;

cout<<str;

4. 用C++写个程序，如何判断一个操作系统是16位还是32位的？不能用sizeof()函数

答：

A1:

16位的系统下，

int i = 65536;

cout << i; // 输出0；

int i = 65535;

cout << i; // 输出-1；

32位的系统下，

int i = 65536;

cout << i; // 输出65536；

int i = 65535;

cout << i; // 输出65535；

A2:

int a = ~0;

if( a>65536 )

{

    cout<<”32 bit”<<endl;

}

else

{

    cout<<”16 bit”<<endl;

}

5, 分别写出BOOL,int,float,指针类型的变量a 与“零”的比较语句。

答：

BOOL : if ( !a ) or if(a)

int : if ( a == 0)

float : const EXPRESSION EXP = 0.000001

if ( a < EXP && a >-EXP)

pointer : if ( a != NULL) or if(a == NULL)

6.在不用第三方参数的情况下，交换两个参数的值

答：

#include <stdio.h>

void main()

{

        int i=60;

        int j=50;

        i=i+j;

        j=i-j;

        i=i-j;

        printf(“i=%d\n”,i);

        printf(“j=%d\n”,j);

}

方法二：

i^=j;

j^=i;

i^=j;

方法三：

// 用加减实现，而且不会溢出

a = a+b-(b=a)

1. There are two int variables: a and b, don’t use “if”, “? :”, “switch”or other judgement statements, find out the biggest one of the two numbers.

答：( ( a + b ) + abs( a - b ) ) / 2

8. 文件中有一组整数，要求排序后输出到另一个文件中

答案：

＃i nclude<iostream>

＃i nclude<fstream>

using namespace std;

void Order(vector<int>& data) //bubble sort

{

int count = data.size() ;

int tag = false ; // 设置是否需要继续冒泡的标志位

for ( int i = 0 ; i < count ; i++)

{

for ( int j = 0 ; j < count - i - 1 ; j++)

{

if ( data[j] > data[j+1])

{

tag = true ;

int temp = data[j] ;

data[j] = data[j+1] ;

data[j+1] = temp ;

}

}

if ( !tag )

break ;

}

}

void main( void )

{

vector<int>data;

ifstream in(“c:\\data.txt”);

if ( !in)

{

cout<<”file error!”;

exit(1);

}

int temp;

while (!in.eof())

{

in>>temp;

data.push\_back(temp);

}

in.close(); //关闭输入文件流

Order(data);

ofstream out(“c:\\result.txt”);

if ( !out)

{

cout<<”file error!”;

exit(1);

}

for ( i = 0 ; i < data.size() ; i++)

out<<data<<” “;

out.close(); //关闭输出文件流

}

9.链表题：一个链表的结点结构

struct Node

{

int data ;

Node \*next ;

};

typedef struct Node Node ;

(1)已知链表的头结点head,写一个函数把这个链表逆序 ( Intel)

Node \* ReverseList(Node \*head) //链表逆序

{

if ( head == NULL || head->next == NULL )

return head;

Node \*p1 = head ;

Node \*p2 = p1->next ;

Node \*p3 = p2->next ;

r />p1->next = NULL ;

while ( p3 != NULL )

{

p2->next = p1 ;

p1 = p2 ;

p2 = p3 ;

p3 = p3->next ;

}

p2->next = p1 ;

head = p2 ;

return head ;

}

(2)已知两个链表head1 和head2 各自有序，请把它们合并成一个链表依然有序。(保留所有结点，即便大小相同）

Node \* Merge(Node \*head1 , Node \*head2)

{

if ( head1 == NULL)

return head2 ;

if ( head2 == NULL)

return head1 ;

Node \*head = NULL ;

Node \*p1 = NULL;

Node \*p2 = NULL;

if ( head1->data < head2->data )

{

head = head1 ;

p1 = head1->next;

p2 = head2 ;

}

else

{

head = head2 ;

p2 = head2->next ;

p1 = head1 ;

}

Node \*pcurrent = head ;

while ( p1 != NULL && p2 != NULL)

{

if ( p1->data <= p2->data )

{

pcurrent->next = p1 ;

pcurrent = p1 ;

p1 = p1->next ;

}

else

{

pcurrent->next = p2 ;

pcurrent = p2 ;

p2 = p2->next ;

}

}

if ( p1 != NULL )

pcurrent->next = p1 ;

if ( p2 != NULL )

pcurrent->next = p2 ;

return head ;

}

(3)已知两个链表head1 和head2 各自有序，请把它们合并成一个链表依然有序，这次要求用递归方法进行。 (Autodesk)

答案：

Node \* MergeRecursive(Node \*head1 , Node \*head2)

{

if ( head1 == NULL )

return head2 ;

if ( head2 == NULL)

return head1 ;

Node \*head = NULL ;

if ( head1->data < head2->data )

{

head = head1 ;

head->next = MergeRecursive(head1->next,head2);

}

else

{

head = head2 ;

head->next = MergeRecursive(head1,head2->next);

}

return head ;

}

10.写一个函数，输入的是整数,倒着输出整数的函数,要求用递归方法

答：

void fun( int a )

{

printf( “%d”, a%10 );

a /= 10;

if( a <=0 )return;

fun( a );

}

11. 写一个函数找出一个整数数组中，第二大的数 （microsoft）

答：

const int MINNUMBER = -32767 ;

int find\_sec\_max( int data[] , int count)

{

int maxnumber = data[0] ;

int sec\_max = MINNUMBER ;

for ( int i = 1 ; i < count ; i++)

{

if ( data > maxnumber )

{

sec\_max = maxnumber ;

maxnumber = data ;

}

else

{

if ( data > sec\_max )

sec\_max = data ;

}

}

return sec\_max ;

}

12. 如何判断一个单链表是有环的？（注意不能用标志位，最多只能用两个额外指针）

答：

struct node { char val; node\* next;}

bool check(const node\* head) {} //return false : 无环；true: 有环

一种O（n）的办法就是（搞两个指针，一个每次递增一步，一个每次递增两步，如果有环的话两者必然重合，反之亦然）：

bool check(const node\* head)

{

if(head==NULL) return false;

node \*low=head, \*fast=head->next;

while(fast!=NULL && fast->next!=NULL)

{

low=low->next;

fast=fast->next->next;

if(low==fast) return true;

}

return false;

}

13.. 写一个在一个字符串(n)中寻找一个子串(m)第一个位置的函数。

答：

#include <iostream>

using namespace std;

void GetNext(char \*t,int \*next)

{

       int i = 1,j = 0;

     int n = strlen(t);

       next[0] = -1;

       next[1] = 0;

   while(i < n)

{

               if((j == -1) || (t[j] == t[i]))

{

next[++i] = ++j;

                }

             else

              {

                     j = next[j];

              }

       }

}

int KMP(char \*s,char \*t)

{

       int n = strlen(s);

       int m = strlen(t);

       int \*next = new int[m];

       GetNext(t,next);

       for(int i=0,j=0;i<n&&j<m;)

       {

              if((j == -1)||(s[i] == t[j]))

              {

                     i++;

                     j++;

              }

              else

              {

                     j = next[j];

              }

       }

      //delete [] next;

      return (j==m?i-j:-1);

}

int main()

{

       int pos;

       char s[100];

       char t[10];

       cout<<”Input s:”;

       cin>>s;

       cout<<”Input t:”;

       cin>>t;

       pos = KMP(s,t);

       if(pos != -1)

       {

              int i = 0;

              while(i < strlen(t))

              {

                     cout<<s[pos+i];

                     i++;

              }

       }

       else

       {

              cout<<”Not found!”<<endl;

       }

       return 0;

}

14.. 试编写函数判断计算机的字节存储顺序是开序(little endian)还是降序(bigendian)

答:

bool IsBigendian()

{

unsigned short usData = 0×1122;

unsigned char \*pucData = (unsigned char\*)&usData;

return (\*pucData == 0×22);

}

15.用递归算法判断数组a[N]是否为一个递增数组。

答：

递归的方法，记录当前最大的，并且判断当前的是否比这个还大，大则继续，否则返回false结束：

bool fun( int a[], int n )

{

if( n= =1 )

return true;

if( n= =2 )

return a[n-1] >= a[n-2];

return fun( a,n-1) && ( a[n-1] >= a[n-2] );

}  
16.

1)写一个递归函数将内存中的字符串翻转“abc”->”cba”

2)写一个函数将“tom is cat” 将内存中的字符串翻转，即 “cat is tomm”

答：

#include <stdio.h>

#define SPACE ‘ ‘

#define ENDL ‘\0′

char\* s = “The quick brown fox jumps over the lazy dog”;

void str\_reverse(char\* p1,char\* p2)

{

if(p1==p2)

return;

\*p1 = (\*p1)+(\*p2);

\*p2 = (\*p1)-(\*p2);

\*p1 = (\*p1)-(\*p2);

if(p1==p2-1)

return;

else

str\_reverse(++p1,–p2);

}

void str\_word\_reverse(char\* str)

{

char \*q1=str, \*q2=str, \*t;

while(\*q1==SPACE)q1++;

if(\*q1==ENDL)

return;

else

q2=q1+1;

while( (\*q2!=SPACE) && (\*q2!=ENDL) )q2++;

t=q2–;

str\_reverse(q1,q2);

  if(\*t==ENDL)

return;

else

str\_word\_reverse(t);

}

int main(int a ,char\*\* b)

{

printf(“%s\n”,s);

str\_reverse(s,s+strlen(s)-1);

printf(“%s\n”,s);

str\_word\_reverse(s);

printf(“%s\n”,s);

return 0;

}

Output:

The quick brown fox jumps over the lazy dog

god yzal eht revo spmuj xof nworb kciuq ehT

dog lazy the over jumps fox brown quick The