

# 前言

当求职者应聘一份程序设计、软件开发方面的工作时，招聘方一般都会安排一次面试来考察其程序设计能力。由于C/C++语言具有很大的灵活性，被广泛运用于各个领域，很多公司将C/C++编程能力作为考察程序员基本素质的主要方式。尤其像我们从达内学习完C/C++、嵌入式课程之后，更应该有针对性和专业性的面试方面的指导书籍以帮助学员更快、更好、更强的找到理想工作，我们通过收集各面试书籍和网络内容为广大达内学员编写了此工具用书。

本书涵盖了C/C++面试中出现的各个知识点，包括C语言基础、C++面向对象、算法、数据结构、win32、MFC、oracle和嵌入式等。所有知识点都结合例题进行说明，每道题都取材于各大公司的实际招聘面试题，并在题后紧跟详细的分析步骤和解答。

**感谢以下中心为该书提供资料（排名无先后）：**

长沙五一路中心、广州岗顶中心、成都高升桥中心、北京天安门中心、成都天府中心、深圳罗湖中心、上海交大中心、上海南京路中心、北京天坛中心、济南中心和杭州下沙中心等。

**目录**

[前言 2](#_Toc938)

[第1篇：C/C++程序基础 7](#_Toc5582)

[1.1 变量及赋值 7](#_Toc7626)

[1.2 类型转换 16](#_Toc27279)

[1.3 数值交换 18](#_Toc13889)

[1.4 C和C++的联系与区别 21](#_Toc23703)

[1.5 C/C++函数 25](#_Toc12031)

[1.6 C++输入输出流 31](#_Toc633)

[第2篇 预处理、const、static与sizeof 33](#_Toc5925)

[2.1 预处理 33](#_Toc7747)

[2.2 const关键字 37](#_Toc8754)

[2.3 static关键字 38](#_Toc20603)

[2.4 sizeof操作符 48](#_Toc17808)

[2.5 inline与宏定义 64](#_Toc7157)

[第3篇 引用和指针 67](#_Toc6024)

[3.1 引用 67](#_Toc2421)

[3.2 指针基础 82](#_Toc8942)

[3.3 this指针 93](#_Toc28980)

[3.4 指针数组与数组指针 97](#_Toc15612)

[3.5 函数指针与指针函数 98](#_Toc25029)

[3.6 野指针 101](#_Toc5943)

[3.7 动态内存 101](#_Toc5870)

[3.8 指针与句柄的区别 105](#_Toc21844)

[第4篇 字符串 107](#_Toc18539)

[4.1 数字与字符串的转化 107](#_Toc28539)

[4.2 字符串与数组 113](#_Toc6077)

[第5篇 位运算 129](#_Toc11092)

[5.1 位制转换与位运算 129](#_Toc23915)

[5.2 嵌入式编程 132](#_Toc2751)

[第6篇 C++面向对象 137](#_Toc22888)

[6.1 new/delete 138](#_Toc17713)

[6.2 访控属性 140](#_Toc10099)

[6.3 模板和容器 143](#_Toc18695)

[6.4 数据成员指针 147](#_Toc8109)

[6.5 类的数据成员 147](#_Toc27721)

[6.6 面向对象基础 153](#_Toc19646)

[6.7 友元 154](#_Toc28852)

[6.8 class和struct的区别 157](#_Toc11326)

[6.9 成员变量 158](#_Toc23824)

[6.10 构造函数和析构函数 162](#_Toc21304)

[6.11 复制构造函数和赋值函数 170](#_Toc32272)

[6.12 函数重载和运算符重载 176](#_Toc8201)

[第7篇 C++继承和多态 181](#_Toc13134)

[7.1 继承的概念 181](#_Toc11235)

[7.2 多态的概念 185](#_Toc4392)

[7.3 多重继承和虚拟继承 204](#_Toc444)

[7.4 纯虚函数和抽象基类 207](#_Toc26248)

[第8篇 数据结构 210](#_Toc3449)

[8.1 单链表 211](#_Toc28122)

[8.2 循环链表 228](#_Toc3465)

[8.3 双向链表 233](#_Toc27323)

[8.4 队列 240](#_Toc25171)

[8.5 栈 244](#_Toc16942)

[8.6 二叉树 249](#_Toc12839)

[第9篇 排序 255](#_Toc18769)

[9.1 插入排序 255](#_Toc19849)

[9.2 交换排序 258](#_Toc9962)

[9.3 选择排序 263](#_Toc24345)

[9.4 归并排序 270](#_Toc17997)

[9.5 各种排序方法比较 275](#_Toc2907)

[第10篇 Oracle、Pro\*C、Pl/Sql 276](#_Toc3680)

[第11篇 Win32、MFC 281](#_Toc2098)

[第12篇：ARM体系结构和接口编程 288](#_Toc26057)

[第13篇：Linux移植和驱动 303](#_Toc5515)

# 第1篇：C/C++程序基础

## 1.1 变量及赋值

【题目编号】 CSD-01-101-1001

【知 识 点】 C/C++程序基础 - 变量及赋值 - 变量隐藏

【难易程度】 50

【题目描述】

局部变量可以隐藏全局变量，那么在有同名全局变量和局部变量的情形时，可以用（ ）提供对全局变量的访问。

【选 项】

A、域运算符

B、类运算符

C、重载

D、引用

【正确答案】 A

【解释】 无

【题目编号】 CSD-01-101-1002

【知 识 点】 C/C++程序基础 - 变量及赋值 - 变量作用域

【难易程度】 50

【题目描述】

所有在函数中定义的变量,连同形式参数,都属于( )。

【选 项】

A、全局变量

B、局部变量

C、静态变量

D、寄存器变量

【正确答案】 B

【解释】 无

【题目编号】 CSD-01-101-1003

【知 识 点】 C/C++程序基础 - 变量及赋值 - 局部变量

【难易程度】 50

【题目描述】

在C++语言中，局部变量的隐含存储类型是( )。

【选 项】

A、auto B、static C、extern D、无存储类型

【正确答案】 A

【解释】 无

【题目编号】 CSD-01-101-1004

【知 识 点】 C/C++程序基础 - 变量及赋值 - 变量

【难易程度】 60

【题目描述】

有一个int型变量，在程序中频繁使用，最好定义它为( )。

【选 项】

A、register B、auto C、extern D、static

【正确答案】 A

【解释】 无

【题目编号】 CSD-01-101-1005

【知 识 点】 C/C++程序基础 - 变量及赋值 - ++i和i++

【难易程度】 70

【题目描述】

下面程序的输出结果为( )。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | #include<iostream>  void prt(int \*x,int \*y,int \*z)  {  cout<< ++\*x << "," << ++\*y << "," << \*(z++) << endl;  }  int a=10;  int c=20;  int b=40;  void main()  {  prt(&a,&b,&c);  prt(&a,&b,&c);  } |

【选 项】

A、11,42,31 12,21,41 B）11,41,20 12,42,20

【正确答案】 B

【解释】 无

【题目编号】 CSD-01-101-4001

【知 识 点】 C/C++程序基础 - 变量及赋值 - 变量隐藏

【难易程度】 60

【题目描述】

局部变量能否和全局变量重名？

【正确答案】

能，局部会屏蔽全局。要用全局变量，需要使用"::"。

【解释】 无

【题目编号】 CSD-01-101-4002

【知 识 点】 C/C++程序基础 - 变量及赋值 -全局变量声明

【难易程度】 70

【题目描述】

如何引用一个已经定义过的全局变量？

【正确答案】

可以用引用头文件的方式，也可以用extern关键字，如果用引用头文件方式来引用某个在头文件中声明的全局变量，假定你将那个变量写错了，那么在编译期间会报错；如果你用extern方式引用时，假定你犯了同样的错误，那么在编译期间不会报错，而在连接期间报错。

【解释】 无

【题目编号】 CSD-01-101-4003

【知 识 点】 C/C++程序基础 - 变量及赋值 - 全局变量

【难易程度】 60

【题目描述】

全局变量可不可以定义在可被多个.c文件包含的头文件中？为什么？

【正确答案】

可以，在不同的C文件中以static形式来声明同名全局变量。可以在不同的C文件中声明同名的全局变量，前提是其中只能有一个C文件中对此变量赋初值，此时连接不会出错。

【解释】 无

【题目编号】 CSD-01-101-4004

【知 识 点】 C/C++程序基础 - 变量及赋值 - 变量隐藏

【难易程度】 80

【题目描述】

分别写出bool,int,float指针类型的变量a与“零”的比较语句。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5 | 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) |

【解释】 无

【题目编号】 CSD-01-101-4005

【知 识 点】 C/C++程序基础 - 变量及赋值 - 变量赋值

【难易程度】 60

【题目描述】

分析代码写输出

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | #include <stdio.h>  int main(void)  {  int x = 3, y, z;  x \*= (y = z = 4); printf("x = %d\n", x);  z = 2;  x = (y = z); printf("x = %d\n", x);  x = (y == z); printf("x = %d\n", x);  x = (y & z); printf("x = %d\n", x);  x = (y && z); printf("x = %d\n", x);  y = 4;  x = (y | z); printf("x = %d\n", x);  x = (y || z); printf("x = %d\n", x);  x = (y == z) ? 4 : 5;  printf("x = %d\n", x);  x = (y == z) ? 1 : (y < z) ? 2 : 3;  printf("x = %d\n", x);  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | x = 12  x = 2  x = 1  x = 2  x = 1  x = 6  x = 1  x = 5  x = 3 |

【解释】

程序执行至第8行时，x的值为3,y和z未被初始化．此行的执行顺序是首先执行z=4,然后y=z，最后是x\*=y。因此x的值为3\*4=12。

程序执行至第10行时，z的值为2。此行的执行顺序是首先执行y=z，然后x=y。因此x的值为2。

程序执行至第11行时，y和z的值都为2。此行的执行顺序是首先执行y==z比较y和z的值是否相等，然后将比较的结果赋给x。因此x的值为1。

程序执行至第12行时，y和z的值都为2。此行把y和z做按位与（&）运算的结果赋给变量x。y和z的二进制都是10,y&z的结果为二进制10。因此x的值为2。

程序执行至第13行时，y和z的值都为2。此行把y和z做逻辑与（&&）运算的结果赋给变量x。此时y和Z的值都不是0,y&&z的结果为1．因此x的值为1。

程序执行至第16行时，y的值为4,z的值为2。此行把y和z做按位或（|）运算的结果赋给变量x。此时y和z的二进制表示分别为100和010，因此y|z的结果为110。因此x的值为110，十进制表示为6。

程序执行至第17行时，y的值为4,z的值为2。此行把y和z做逻辑或（||）运算的结果赋给变量x．此时y和z的值都不是0，y||z的结果为1。因此x的值为1。程序执行至第19行时，y的值为4,z的值为2。此行首先比较y和z的大小是否相等，如果相等，则将x取4；否则x取5。在这里y不等于z，因此x的值为5。

程序执行至第22行时，y的值为4,z的值为2。此行首先比较y和z大小是否相等，如果相等，x取1；否则判断y是否大于z，如果是则x取2，否则x取3。在这里y的值大于z的值，因此x的值为3。

总结：这个考题只是考察各种基本的赋值运算，要注意位运算与逻辑运算的区别，以及三元操作符的用法，通过程序代码17行以及19行的举例，可以发现三元操作符有时可以代替条件判断if/else/else/if的组合。

【题目编号】 CSD-01-101-4006

【知 识 点】 C/C++程序基础 - 变量及赋值 - C++域操作符的作用。

【难易程度】 70

【题目描述】

分析代码写运行结果

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | #include <stdio.h>  int value = 0;  void printvalue()  {  printf("value = %d\n", value);  }  int main()  {  int value = 0;    value = 1;  printf("value = %d\n", value);    ::value = 2;  printvalue();    return 0;  } |

【正确答案】

在C中编译不能通过，并指示17行符号错误。

在C++中的输出如下：

value=1（局部变量value）

value=2（全局变最value）

【解释】

如果将文件保存为后缀名为.c的文件，在场Visual C++ 6.0中不能通过编译并且提示17行有语法错误。而如果文件保存为后级名为.cpp的文件，在Visual C++ 6.0中就能顺利通过编译并且运行。

这段程序有两个变量，其名字都是value。不同的是其中一个是在main函数之前就声明的全局变量，而另外一个是在main函数内部声明的局部变量。这两个变量的作用域是不一样的。

注意：在函数printvalue里打印的是全局变量的值，在main函数的15行打印的是局部变量的值。这是因为在main函毅里的局部变量value引用优先。在C++中可以通过域操作符"::"来直接操作全局变量（即代码的17行操作的value是全局变量），但是在C中不支持这个操作符，因此会报错．在C中不推荐这种局部变量与全局变量同名的设计方式。

【题目编号】 CSD-01-101-4007

【知 识 点】 C/C++程序基础 - 变量及赋值 - ++i和i++

【难易程度】 70

【题目描述】

分析代码写输出。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #include <stdio.h>  int main(void)  {  int i = 8;    printf("%d\n", ++i);  printf("%d\n", --i);  printf("%d\n", i++);  printf("%d\n", i--);  printf("%d\n", -i++);  printf("%d\n", -i--);  printf("-------\n");    return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | 9  8  8  9  -8  -9  ------- |

【解释】

程序第7行，此时i的值为8。这里先i自增1再打印i的值。因此输出9，并且i的值也变为9。

程序第8行，此时i的值为9。这里先i自减1再打印i的值。因此输出8，并且i的值也变为8。

程序第9行，此时i的值为8。这里先打印i的值再i自增1。因此愉出8，并且i的值变为9。

程序第10行，此时i的值为9。这里先打印i的值再i自减1。因此输出9，并且i的值变为8。

程序第11行，此时i的值为8。这里的“-”表示负号运算符。因此先打印-i的值再i自增1。因此输出-8，并且i的值变为9。

程序第12行，此时i的值为9。这里的第一个“-”表示负号运算符，后面的--”表示自减运算符。因此先打印-i的值再i自减1。因此输出-9，并且i的值也变为8。

【题目编号】 CSD-01-101-4008

【知 识 点】 C/C++程序基础 - 变量及赋值 - i++和++i效率

【难易程度】 80

【题目描述】

i++与++i哪个效率更高？

【正确答案】

在内建数据类型的情况下，效率没有区别。在自定义数据类型的情况下，++i效率较高。

【解释】

简单的比较前缀自增运算符和后级自增运算符的效率是片面的，因为存在很多因素影响这个问题的答案。首先考虑内建数据类型的情况：如果自增运算表达式的结果没有被使用，而是仅仅简单地用于增加一元操作数，答案是明确的，前级法和后缀法没有任何区别。编译器的处理都应该是相同的，很难想象得出有什么编译器实现可以别出心裁地在二者之间制造任何差异。

考虑内建数据类型时，它们的效率差别不大（去除编译器优化的影响）。再考虑自定义数据类型（主要是指类）的情况。此时不需要再做很多汇编代码的分析，因为前缀式（++i）可以返回对象的引用，而后缀式（i++）必须返回对象的值，所以导致在大对象的时候产生了较大的复制开销，引起效率降低，因此使用自定义类型（注意不是指内建类型）的时候，应该尽可能地使用前缀式递增或递减。

## 1.2 类型转换

类型转换是将一种类型的值映射为另一种类型的值。类型转换包含自动隐含和强制的两种。C++语言编译系统提供的内部数据类型的自动隐式转换规则如下。

（1）程序在执行算术运算时，低类型自动隐式转换为高类型。在赋值表达式中，右边表达式的值自动隐式转换为左边变量的类型，并赋值给左边变量。

（2）在函数调用时，将实参值赋给形参，系统隐式地将实参转换为形参的类型，并赋值给形参。

（3）函数有返回值时，系统自动地将返回表达式类型转换为函数类型，并赋值给调用函数。当在程序中发现两个数据类型不相容时，又不能自动完成隐式转换，则将出现编译错误。

【题目编号】 CSD-01-102-4001

【知 识 点】 C/C++程序基础 - 类型转换 - 有符号变量与无符号变量的区别和联系

【难易程度】 70

【题目描述】

分析代码写结果。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | #include <stdio.h>  char getChar(int x, int y)  {  char c;  unsigned int a = x;    (a + y > 10) ? (c = 1) : (c = 2);  return c;  }  int main(void)  {  char c1 = getChar(7, 4);  char c2 = getChar(7, 3);  char c3 = getChar(7, -7);  char c4 = getChar(7, -8);    printf("c1 = %d\n", c1);  printf("c2 = %d\n", c2);  printf("c3 = %d\n", c3);  printf("c4 = %d\n", c4);    return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4 | c1 = 1  c2 = 2  c3 = 2  c4 = 1 |

【解释】

首先说明一下getChar()函数的作用。getChar()有两个输入参数，分别是整型的x和y。在函数体内，把参数x的值转换为无符号整型后再与y相加，其结果与10进行比较，如果大于10则函数返回1，否则返回2。

注意：当表达式中存在有符号类型和无符号类型时，所有的操作数都自动转换成无符号类型。

由于a是无符号数，代码第8行中y值会首先自动转换成无符号的整数然后再与a相加，最后再与10进行比较。以下是在main函数中各调用getChar()函数的分析。

代码第14行，传入的参数分别为7和4，两个数相加后为11，因此c1返回1。

代码第15行，传入的参数分别为7和3，两个数相加后为10，因此c1返回2。

代码第16行，传入的参数分别为7和-7，-7首先被转换成一个很大的数，然后与7相加后正好溢出，其值为0，因此c1返回2。

代码第17行，传入的参数分别为7和-8，-8首先被转换成一个很大的数，然后与7相加。两个数相加后为很大的整数（差1就溢出了），因此c1返回1。

可以看出，由于无符号整数的特性，当参数x为7时，如果y等于区间卜7,3］中的任何整数值，getChar()函数都将返回2。当y的值在区间[-7,3]之外时，函数返回1。

总之，使用表达式时要注意符号变量与无符号变量之间的转换，占用不同字节内存的变量之间的赋值等操作，否则可能会出现意想不到的运行结果。

## 1.3 数值交换

如果一个函数要修改传入参数的值，直接传值是不行的。因为这样只是在函数的栈中复制了参数的值，并没有修改实际的参数值。在C中可以通过将参数声明为指针类型达到目的。而在C++中，又多了一种选择，就是可以将参数声明为引用类型，实际上引用就是某种意义上的安全指针。

【题目编号】 CSD-01-103-4001

【知 识 点】 C/C++程序基础 – 数值交换 - 交换两个变量的值

【难易程度】 70

【题目描述】

将数a、b的值进行交换，并且不使用任何中间变量。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | #include <stdio.h>  void swap1(int &a, int &b)  {  int temp = a;  a = b;  b = temp;  }  void swap2(int &a, int &b)  {  a = a + b;  b = a - b;  a = a - b;  }  void swap3(int &a, int &b)  {  a ^= b;  b ^= a;  a ^= b;  }  int main(void)  {  int a1 = 1, b1 = 2;  int a2 = 3, b2 = 4;  int a3 = 5, b3 = 6;  int a = 2147483647, b = 1;    swap1(a1, b1);  swap2(a2, b2);  swap3(a3, b3);    printf("after swap ...\n");  printf("a1 = %d, b1 = %d\n", a1, b1);  printf("a2 = %d, b2 = %d\n", a2, b2);  printf("a3 = %d, b3 = %d\n", a3, b3);    swap2(a, b);  printf("a = %d, b = %d\n", a, b);    return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5 | after swap……  a1 = 2, b1 = 1  a1 = 4, b1 = 3  a1 = 6, b1 = 5  a1 = 1, b1 = 2147483647 |

【解释】

以上的C++程序中有3个swap函数都是采用引用传参的方式。

swap1()采用的是教科书里常见的方式，用一个局部变量temp保存其中一个值来达到交换目的。当然，这种方式不是本题要求的答案。

swap2()采用的是一种简单的加减算法来达到交换a、b的目的。这种方式的缺点是做“a+b“和“a-b”运算时可能会导致数据溢出。

swap()采用了按位异或的方式交换a、b。按位异或运算符“^”的功能是将参与运算的两数各对应的二进制位相异或，如果对应的二进制位相同，则结果为0，否则结果为1。这样运算3次即可交换a、b的值。

代码第31行～第32行调用3种swap函数。注意40行的调用，在swap2函数栈中的运算会有数据溢出发生。我们知道在32位平台下int占4个字节内存，其范围是一2147483648~2147483647，因此2147453647加1就变成了-2147483648。不过通过运行结果我们可以看到，虽然产生了溢出，但是交换操作依然是成功的。

注意：采用程序代码中swap2和swap3的交换方式，swap2有可能发生数据溢出的缺点。相比较swap2，推荐swap3采用的按位异或的方式。

## 1.4 C和C++的联系与区别

C++从C基础上发展而来，且大大扩充了C的内容和功能，提供了更多更全面的支持。只会C语言还远远不能说会了C++。对于C语言程序员，完全可以把C++作为一门新语言来学。

【题目编号】 CSD-01-104-1001

【知 识 点】 C/C++程序基础 - C和C++的联系与区别 - C++与C的联系及区别

【难易程度】 60

【题目描述】

C++对C语言做了很多改进，下列描述中（ ）使得C语言发生了质变，即从面向过程变成面向对象。

【选 项】

A、增加了一些新的运算符

B、允许函数重载，并允许设置默认参数

C、规定函数说明必须用原型

D、引进类和对象的概念

【正确答案】 D

【解释】 无

【题目编号】 CSD-01-104-1002

【知 识 点】 C/C++程序基础 - C和C++的联系与区别 - C++概念

【难易程度】 60

【题目描述】

下列关于C++类的描述中错误的是（ ）。

【选 项】

A、类与类之间可以通过一些手段进行通信和联络

B、类用于描述事物的属性和对事物的操作

C、类与类之间必须是平等的关系，而不能组成层次关系

D、类与类之间可以通过封装而具有明确的独立性

【正确答案】 C

【解释】 无

【题目编号】 CSD-01-104-1003

【知 识 点】 C/C++程序基础 - C和C++的联系与区别 - C++概念

【难易程度】 60

【题目描述】

下列关于C++与C语言关系的描述中错误的是（ ）。

【选 项】

A、C++是C语言的超集

B、C++是C语言进行了扩充

C、C++是C语言都是面向对象的程序设计语言

D、C++包含C语言的全部语法特征

【正确答案】 C

【解释】

C语言是面向过程的程序设计语言。

【题目编号】 CSD-01-104-4001

【知 识 点】 C/C++程序基础 - C和C++的联系与区别 - 概念区分

【难易程度】 60

【题目描述】

请简述C、C++、VC、MFC在概念上的区别。

【正确答案】

C是面向过程编程的语言；C++是面向对象编程的语言；VC是微软提供的方便于开发C++程序的一套开发工具。MFC（Microsoft Function Class），是微软提供的一套函数类库，里面封装了许多Windows API函 数，方便开发者调用。

【解释】 无

【题目编号】 CSD-01-104-4002

【知 识 点】 C/C++程序基础 - C和C++的联系与区别 - C++和C的联系及区别。

【难易程度】 60

【题目描述】

C++与C有什么不同？

【正确答案】

C是一个结构化语言，它的重点在于算法和数据结构。对语言本身而言，C是C++的子集。

C程序的设计首要考虑的是如何通过一个过程，对输入进行运算处理得到输出。对于C++，首要考虑的是如何构造一个对象模型，让这个模型能够配合对应的问题，这样就可以通过获取对象的状态信息得到输出或实现过程控制。

因此C与C++的最大区别在于它们解决问题的思想方法不一样。

C实现了C++中过程化控制及其他相关功能。而在C++中的C，相对于原来的C还有所加强，引入了重载、内联函数、异常处理等，C++拓展了面向对象设计的内容，如类、继承、虚函数、模板和包容器类等。

在C++中，不仅需要考虑数据封装，还需要考虑对象粒度的选择、对象接口的设计和继承、组合与继承的使用等问题。

【解释】 无

【题目编号】 CSD-01-104-4003

【知 识 点】 C/C++程序基础 - C和C++的联系与区别 - 标准头文件的通用结构。

【难易程度】 70

【题目描述】

为什么标准头文件都有类似以下所示的结构？

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | #ifndef \_\_INCvxWorksh  #define \_\_INCvxWorksh  #ifdef \_\_cplusplus  extern "C"{  #endif  /\* ... \*/  #ifdef \_\_cplusplus  }  #endif  #endif // \_\_INCvxWorksh |

【正确答案】

代码第1、2、10行的作用是防止该头文件被重复引用。

代码第3行的作用是表示当前使用的是C++编译器。

代码第4行～第8行中的extern “C”是C++编译器提供的与C连接交换指定的符号，用来解决名字匹配问题。

【解释】

显而易见，代码1、2、10行的作用是防止该头文件被重复引用。代码第3行的作用是表示当前使用的是C++编译器。如需要表示当前使用的是C编译器，可以这样指定：

#ifdef \_\_STDC\_\_

那么代码第4行～第8行中的extern "C"有什么作用呢？

extern "C"包含双重含义。

（1）被它修饰的目标是“extern”的。也就是告诉编译器，其声明的函数和变量可以在本模块或其他模块中使用。通常，在模块的头文件中对本模块供给其他模块引用的函数和全局变最以关键字extern声明。例如，如果模块B打算引用该模块A中定义的全局变量和函数，只需包含模块A的头文件即可。这样，模块B调用模块A的函数时，在编译阶段，模块B即便找不到该函数，编译器也不会报错，它会在连接阶段中从模块A编译生成的目标代码中找到此函数。

（2）被它修饰的目标是“C”的。意思是其修饰的变量和函数是按照C语言方式编译和连接的。首先必须弄清C++中对类似C的函数是怎样编译的。作为一种面向对象的语言，C++支持函数重载，而过程式语言C则不支持。函数被C++编译后在符号库中的名字与C语言的不同。

【题目编号】 CSD-01-104-4004

【知 识 点】 C/C++程序基础 - C和C++的联系与区别 - 头文件引用中<>与””的区别。

【难易程度】 50

【题目描述】

#include <head.h>和#include “head.h”有什么区别？

【正确答案】

<>表明括号中的文件是一个工程或标准头文件。查找过程会首先检查预定义的目录，开发者可以通过设置搜索路径环境变量或命令行选项来修改这些目录。

如果文件名用一对引号括起来则表明该文件是用户提供的头文件，查找该文件时将从当前文件目录或文件名指定的其他目录中寻找，然后再在标准位置寻找。

【解释】 无

## 1.5 C/C++函数

main函数代表进程的主线程。程序开始执行时，系统为程序创建一个进程，main函数其实并不是首先被调用的函数，而是操作系统调用了C/C++运行期启动函数，该函数负责对C/C++运行期库进行初始化。它还能够保证己经声明了的任何全局对象和静态对象能够在代码执行之前正确的创建。

完成这些工作后，就调用进入点函数（控制台程序为main函数），并在main函数里面执行一系列操作。

在main执行完毕后，从main函数返回，启动函数调用C运行期的exit函数，将返回值传递给它。其中在exit函数会调用ExitProcess函数，结束进程。

【题目编号】 CSD-01-105-1001

【知 识 点】 C/C++程序基础 - C/C++函数 - 全局变量与局部变量

【难易程度】 50

【题目描述】

下面程序的执行结果是( )。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | #include<iostream>  void main()  {  void swap();  extern int a,b;  a=3;  b=10;  swap();  cout<<"a="<<a<<",b="<<b<<endl;  }  int a,b;  void swap()  {  int temp;  temp=a;  a=b;  b=temp;  } |

【选 项】

A、a=3,b=10 B、a=10,b=3 C、a=temp,b=temp D、语法错误，不能输出正确结果

【正确答案】 B

【解释】

函数swap是无参，所以修改是是main()函数下面的的a,b的值，而不是main()函数体内a，b的值。

【题目编号】 CSD-01-105-1002

【知 识 点】 C/C++程序基础 - C/C++函数 - 值传递

【难易程度】 50

【题目描述】

下面程序的输出结果是( )。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | #include<iostream>  int x=10;  void fun(int c);  void main()  {  fun(x);  cout<<"x="<<x<<endl;  }  void fun(int x)  {  x=1;  } |

【选 项】

A、x=0 B、x=1 C、x=10 D、语法错误，不能输出正确结果

【正确答案】 C

【解释】

值传递不会修改传入的实参的值。

【题目编号】 CSD-01-105-1003

【知 识 点】 C/C++程序基础 - C/C++函数 - 内联函数

【难易程度】 60

【题目描述】

要求通过函数来实现一种不太复杂的功能，并且要加快执行速度，选用( )。

【选 项】

A、内联函数 B、重载函数 C、递归函数 D、嵌套函数

【正确答案】 A

【解释】

注意内联函数所实现的功能应该尽量简单，太复杂的功能不适合用内联函数。

【题目编号】 CSD-01-105-1004

【知 识 点】 C/C++程序基础 - C/C++函数 - 函数参数默认值

【难易程度】 60

【题目描述】

考虑下面的函数原型：void testDefaulParam(int a,int b=7,char z='\*');下面函数调用中不合法的( )。

【选 项】

A、testDefaulParam(5);

B、testDefaulParam(5,8);

C、testDefaulParam(5,'#');

D、testDefaulParam(0,0,'\*');

【正确答案】 C

【解释】

因为有默认参数，所以调用testDefaulParam函数时只需要传递一个实参就可以了。如果传递了更多的参数，仍然需要遵守参数传递原则。

【题目编号】 CSD-01-105-1005

【知 识 点】 C/C++程序基础 - C/C++函数 - 函数参数值传递

【难易程度】 60

【题目描述】

在传值调用中，要求( )。

【选 项】

A、形参和实参类型任意，个数相等

B、实参和形参类型都完全一致，个数相等

C、实参和形参对应的类型一致，个数相等

D、实参和形参对应的类型一致个数任意

【正确答案】 C

【解释】 无

【题目编号】 CSD-01-105-1006

【知 识 点】 C/C++程序基础 - C/C++函数 - 函数声明

【难易程度】 50

【题目描述】

在C++中，函数原型不能标识( )。

【选 项】

A、函数的返回类型

B、函数参数的个数

C、函数参数类型

D、函数的功能

【正确答案】 D

【解释】

函数可以先声明，不实现。

【题目编号】 CSD-01-105-1007

【知 识 点】 C/C++程序基础 - C/C++函数 - 函数返回类型

【难易程度】 50

【题目描述】

当一个函数无返回值时，函数的类型应定义为( )。

【选 项】

A、void B、任意 C、int D、无

【正确答案】 A

【解释】 无

【题目编号】 CSD-01-105-1008

【知 识 点】 C/C++程序基础 - C/C++函数 - 函数调用

【难易程度】 50

【题目描述】

下面叙述中不正确的是（ ）。

【选 项】

A、在函数中，通过return语句传回函数值

B、在函数中，可以有多条return语句

C、主函数名main()后面的一对圆括号中也可以带有形参

D、调用函数必须在一条独立的语句中完成

【正确答案】 D

【解释】

可以在任意地方完成函数调用。

【题目编号】 CSD-01-105-1009

【知 识 点】 C/C++程序基础 - C/C++函数 - 函数返回值

【难易程度】 60

【题目描述】

在函数的返回值类型与返回值表达式的类型的描述中，错误的是（ ）。

【选 项】

A、函数返回值的类型是在定义函数时确定，在函数调用时是不能改变的

B、函数返回值的类型就是返回值表达式的类型

C、函数返回值表达式类型与返回值类型不同时，函数表达式类型应转换成返回值类型

D、函数返回值类型确定了返回值表达式的类型

【正确答案】 B

【解释】

可隐式转换的类型。

【题目编号】 CSD-01-105-1010

【知 识 点】 C/C++程序基础 - C/C++函数 - 函数声明

【难易程度】 60

【题目描述】

在函数声明时，下列（ ）项是不必要的。

【选 项】

A、函数的类型 B、函数参数类型 C、函数的名字 D、返回值表达式

【正确答案】 D

【解释】 无

【题目编号】 CSD-01-105-1011

【知 识 点】 C/C++程序基础 - C/C++函数 - 函数参数

【难易程度】 60

【题目描述】

下面函数调用语句中实参的个数是( )。

xxx((v1,v2),(v3,v4,v5),v6);

【选 项】

A、3 B、4 C、5 D、6

【正确答案】 A

【解释】

括号操作符。

【题目编号】 CSD-01-105-1012

【知 识 点】 C/C++程序基础 - C/C++函数 - 函数返回值

【难易程度】 50

【题目描述】

C++语言中规定函数的返回值的类型是由（ ）。

【选 项】

A、return语句中的表达式类型所决定的

B、调用该函数式的主调用函数类型所决定的

C、调用该函数式系统临时确定的

D、在定义该函数时所指定的数据类型所决定的

【正确答案】 D

【解释】 无

【题目编号】 CSD-01-105-1013

【知 识 点】 C/C++程序基础 - C/C++函数 - 函数概念

【难易程度】 70

【题目描述】

下列关于C++函数的描述中正确的是（ ）。

【选 项】

A、主函数是程序的入口，它是由用户定义的函数调用。

B、函数是C++中面向对象程序设计框架的基本组成元素。

C、函数在调用前必须先被声明。

D、每个用户定义函数只能被其他函数调用一次。

【正确答案】 C

【解释】

A运行时库会调用主函数

B 类

D 可以被随意调用

【题目编号】 CSD-01-105-1014

【知 识 点】 C/C++程序基础 - C/C++函数 - main函数

【难易程度】 60

【题目描述】

每个C++程序都必须由且仅有一个（ ）。

【选 项】

A、预处理命令 B、主函数

C、函数 D、语句

【正确答案】 B

【解释】 无

【题目编号】 CSD-01-105-4001

【知 识 点】 C/C++程序基础 - C/C++函数 - atexit函数的使用。

【难易程度】 70

【题目描述】

C++中main函数执行完后还执行其他语句吗？

【正确答案】

可以用atexit函数来注册程序正常终止时要被调用的函数，并且在main函数结束时调用这些函数的顺序与注册它们的顺序相反。

【解释】

在程序退出的时候经常需要做一些诸如释放资源的操作，但程序退出的方式有很多种，例如main函数运行结束、在程序的某个地方用exit结束程序、用户通过快捷键Ctrl+C等操作发信号来终止程序等，因此需要一种与程序退出方式无关的方法来进行程序退出时的必要处理。atexit函数用来注册程序正常终止时要被调用的函数。

atexit函数的参数是一个函数指针，函数指针指向一个没有参数也没有返回值的函数。atexit函数原型如下：int atexit( void (\*) (void) );

在一个程序中最多可以用atexit函数注册32个处理函数，这些处理函数的调用顺序与其注册的顺序相反，即最先注册的最后调用，最后注册的最先调用。

## 1.6 C++输入输出流

【题目编号】 CSD-01-106-1001

【知 识 点】 C/C++程序基础 - C++输入输出流 - C++文件操作

【难易程度】 60

【题目描述】

进行文件操作时需要包含（ ）头文件。

【选 项】

A、iostream

B、fstream

C、stdio

D、stdlib

【正确答案】 B

【解释】

fstream是文件操作的最高父类，他又包括两个子类：ifstream和ofstream。

【题目编号】 CSD-01-106-1002

【知 识 点】 C/C++程序基础 - C++输入输出流 - 流类

【难易程度】 70

【题目描述】

当使用ifstream流类定义一个流对象并打开一个磁盘文件时，文件的隐含打开方式为（）。

【选 项】

A、ios::in

B、ios::out

C、ios::trunc

D、ios::binary

【正确答案】 A

【解释】

fstream有两个子类：ifstream(input file stream)和ofstream(outpu file stream)，ifstream默认以输入方式打开文件，而ofstream默认以输出方式打开文件。

【题目编号】 CSD-01-106-1003

【知 识 点】 C/C++程序基础 - C++输入输出流 - 格式输出

【难易程度】 70

【题目描述】

使用如setw()的操作符对数据进行格式输出时，应包含（ ）文件。

【选 项】

A、iostream

B、fstream

C、stdio

D、iomanip

【正确答案】 D

【解释】

io代表输入输出，manip是manipulator（操纵器）的缩写。iomanip的作用:主要是对cin,cout之类的一些操纵运算子，比如setfill,setw,setbase,setprecision等等。它是I/O流控制头文件,就像C里面的格式化输出一样。

【题目编号】 CSD-01-106-1004

【知 识 点】 C/C++程序基础 - C++输入输出流 - 输入输出流类

【难易程度】 70

【题目描述】

下列类中( )不是输入输出流类iostream的派生类。

【选 项】

A、 fstream B、 ofstream C、 strstream D、 ostrstream

【正确答案】 BD

【解释】

ofstream和ostrstream派生自ostream，而不是iostream。

【题目编号】 CSD-01-106-1005

【知 识 点】 C/C++程序基础 - C++输入输出流 - ostream类

【难易程度】 60

【题目描述】

在下列选项中（ ）不是ostream类的对象。

【选 项】

A、 cin B、 cerr C、 clog D、 cout

【正确答案】 A

【解释】

cin是istream对象。

# 第2篇 预处理、const、static与sizeof

## 2.1 预处理

预处理是指通过预处理的内建功能对一个资源进行等价替换，最常见的预处理有：文件包含（#include）、条件编译（#ifdef、#else等）、而已控制（#progma）以及宏替换（#define）。

【题目编号】 CSD-02-201-4001

【知 识 点】 预处理、const、static与sizeof - 预处理 - 宏判断

【难易程度】 70

【题目描述】

.h头文件中的ifndef/define/endif 的作用？

【正确答案】

ifndef/define/endif是预处理命令中的条件编译，作用是防止头文件被重复引用。

【解释】 无

【题目编号】 CSD-02-201-4002

【知 识 点】 预处理、const、static与sizeof - 预处理 - 头文件包含

【难易程度】 60

【题目描述】

#include<file.h>与#include "file.h"的区别？

【正确答案】

#include<file.h>表示从标准库中查找头文件，#include”file.h“，表示从当前路径开始查找，如果找不到再到标准库中查找。

【解释】 无

【题目编号】 CSD-02-201-4003

【知 识 点】 预处理、const、static与sizeof - 预处理 - 系统定义宏

【难易程度】 70

【题目描述】

如何打印出当前源文件的文件名以及源文件的当前行号？

【正确答案】

cout << \_\_FILE\_\_ ;

cout<<\_\_LINE\_\_ ;

【解释】

\_\_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。

【题目编号】 CSD-02-201-4004

【知 识 点】 预处理、const、static与sizeof - 预处理 - 宏定义

【难易程度】 80

【题目描述】

用#define实现宏，并求最大值和最小值。

【正确答案】

#define MAX(x, y) (((x) > (y)) ? (x) : (y))

#define MIN(x, y) (((x) < (y)) ? (x) : (y))

【解释】

在这里MAX(x, y)表示取x和y的最大值，MIN(x, y)表示取x和y的最小值。此题有以下3个目的。

考查＃define在宏上应用的基本知识。

考查三重条件操作符的应用。这个操作符能产生比if-else更优化的代码，并且书写上更加简洁明了。

考查宏定义的写法。在宏中需要把参数小心地用括号括起来。因为宏只是简单的文本并换，如果不注意很容易引起歧义。

【题目编号】 CSD-02-201-4005

【知 识 点】 预处理、const、static与sizeof - 预处理 - 宏定义的使用

【难易程度】 70

【题目描述】

分析代码写结果

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | #include <stdio.h>  #define SQR(x) (x \* x)  int main(void)  {  int a, b = 3;  a = SQR(b + 2);  printf("a = %d\n", a);  return 0;  } |

【正确答案】 a=11。

【解释】

这里定义的SQR(x)函数显然是想要获得x的二次乘方，在第7行中调用的参数为b+2，原本想将a赋为（b+2)\*(b+2）也就是5的二次方，即25，但是由于宏定义展开是在预处理时期，也就是在编译之前。此时b并没有被赋值，这时的b只是一个符号。因此在第7行被展开成：a = (b + 2 \* b + 2);于是程序执行后，可以看到a被赋成11。为了达到原来的目的，可以将SQR(x)改成如下定义：#define SQR(x) ((x) \* (x));这样在第7行会被展开成：a = ((b + 2) \* (b + 2));程序执行后，a被赋成25。

【题目编号】 CSD-02-201-4006

【知 识 点】 预处理、const、static与sizeof - 预处理 - 连接宏参数

【难易程度】 80

【题目描述】

分析代码写结果。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | #include <stdio.h>  #define STR(s) #s  #define CONS(a, b) (int)(a##e##b)  int main(void)  {  printf(STR(vck));  printf("\n");  printf("%d\n", CONS(2, 3));    return 0;  } |

【正确答案】

vck

2000

【解释】

在本程序中，使用＃把宏参数变为一个字符串，用##把两个宏参数贴合在一起。

代码第3行STR(s)定义的是一个参数s表示的字符串，在第8行的调用中，STR(vck)实际表示的就是字符串“vck”。

代码第4行CONS(a,b)定义的是一个将参数a与b按aeb连接起来的一个整型值，在第10行的调用中，CONS(2,3)实际表示就是整型值2e3，也就是十进制数2000。

【题目编号】 CSD-02-201-4007

【知 识 点】 预处理、const、static与sizeof - 预处理 - 宏定义与位运算的使用

【难易程度】 70

【题目描述】

用宏定义得到一个字的高位和低位字节。

【正确答案】

#define WORD\_LO(xxx) ((byte)((word)(xxx)&255))

#define WORD\_HI(xxx) ((byte)((word)(xxx)>>8))

【解释】

一个字由2字节组成。因此WORD\_LO(xxx)取参数xxx的低8位，WORD\_HI(xxx)取参数xxx的高8位。

【题目编号】 CSD-02-201-4008

【知 识 点】 预处理、const、static与sizeof - 预处理 - 宏定义与sizeof关键字的使用

【难易程度】 70

【题目描述】

用宏定义得到一个数组所含的元素个数。

【正确答案】

#define ARR\_SIZE(a) (sizeof((a)) / sizeof(a[0])))

【解释】 无

## 2.2 const关键字

const是C语言的一个关键字，它所限定变量不允许被改变。使用const可以在一定程度上增强程序的健壮型，减少程序出错。虽然const有诸多优势，但是const的使用却并不简单。

【题目编号】 CSD-02-201-1001

【知 识 点】 预处理、const、static与sizeof - const关键字 - const关键字

【难易程度】 60

【题目描述】

关于使用const关键字修饰的对象的说法正确的是（ ）。

【选 项】

A、不能用一般的成员函数来访问

B、可以用一般的成员函数来访问

C、可以访问const成员函数及volatile成员函数

D、不能访问const 成员函数

【正确答案】 A

【解释】 无

【题目编号】 CSD-02-201-4001

【知 识 点】 预处理、const、static与sizeof - const关键字 - const与#define

【难易程度】 70

【题目描述】

请说明const与#define的各自特点及区别。

【正确答案】

#define只是用来做文本替换的，它是一个编译期的概念#define常量的生命周期止于编译期，它存在于程序的代码段，在实际程序中它只是一个常数，一个命令中的参数，并没有实际的存在。

const是C语言的一个关键字，它所限定变量不允许被改变。使用const可以在一定程度上增强程序的健壮型，减少程序出错。虽然const有诸多优势，但是const的使用却并不简单。

【解释】 无

【题目编号】 CSD-02-201-4002

【知 识 点】 预处理、const、static与sizeof - const关键字 - const作用

【难易程度】 70

【题目描述】

C++中const有什么作用？至少说明3种。

【正确答案】

const用于定义常量：const定义的常量编译器可以对其进行数据静态类型安全检查。

const修饰函数形式参数：当输入参数为用户自定义类型和抽象数据类型时，将“值传递”改为“const＆传递”可以提高效率。

const修饰函数的返回值：如果给“指针传递”的函数返回值加const，则返回值不能被直接修改，且该返回值只能被赋值给const修饰的同类型指针。

const修饰类的成员函数（函数定义体）：任何不需要修改数据成员的函数都应该使用const修饰，这样即使不小心修改了数据成员或调用了非const成员函数，编译器也会报错。

【解释】 无

## 2.3 static关键字

static关键字代表静态，它可以作用于变量以及函数。例如，在局部变量前加上static关键字后，就定义了静态局部变量，在函数的返回类型前加上static关键字后，就定义了静态函数。在C++的面向对象编程中，static还可以加在类的数据成员或成员函数之前，这样定义的数据成员或成员函数就被类所拥有，而不再属于类的对象。

【题目编号】 CSD-02-203-1001

【知 识 点】 预处理、const、static与sizeof - static关键字 - static变量的生命周期

【难易程度】 60

【题目描述】

在横线处填上适当的语句（ ），使下面的程序的执行结果是19 24 30 37 45。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #include<iostream>  int fun(int a)  {  static int x=10;  int y=1;  x+=a;  a++;  y++;  return(x+y+a);  }  void main()  {  int i=3;  while(i<8)  \_\_\_\_\_\_\_\_\_\_\_\_\_;  cout<<endl;  } |

【选 项】

A、cout<<fun(i++)<<" "

B、cout<<fun(i)<<" "

C、cout<<fun(i++)<<" ,"

D、cout<<fun(i)<<" ,"

【正确答案】 A

【解释】 无

【题目编号】 CSD-02-203-1002

【知 识 点】 预处理、const、static与sizeof - static关键字 - static变量的生命周期

【难易程度】 60

【题目描述】

下列程序的输出结果是（ ）。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | #include<iostream>  void f()  {  static int i=15;  i++;  cout<<"i="<<i<<endl;  }  void main()  {  for(int k=0;k<2;k++)  f();  } |

【选 项】

A、i=16 i=17

B、i=16 i=15

C、i=15 i=16

D、i=15 i=17

【正确答案】 A

【解释】 无

【题目编号】 CSD-02-203-1003

【知 识 点】 预处理、const、static与sizeof - static关键字 - static变量的生命周期

【难易程度】 60

【题目描述】

下面程序的输出结果是( )。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | #include<iostream>  void fun(int \*s,int \*y)  {  static int t=3;  \*y=s[t];  t--;  }  void main()  {  int a[]={1,2,3,4};  int i,x=10;  for(i=0;i<4;i++)  {  fun(a,&x);  cout<<x<<",";  }  cout<<endl;  } |

【选 项】

A、1,2,3,4 B、0,0,0,0 C、4,4,4,4 D、4,3,2,1

【正确答案】 D

【解释】 无

【题目编号】 CSD-02-203-1004

【知 识 点】 预处理、const、static与sizeof - static关键字 - static变量的生命周期

【难易程度】 60

【题目描述】

下面程序的正确运行结果为( )。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #include<iostream>  fun(int a,int b)  {  static int m=1,i=2;  i+=m+1;  m=i+a+b;  return(m);  }  int main()  {  int k=5,m=2,p;  p=fun(k,m);  cout<<p<<",";  p=fun(k,m);  cout<<p<<endl;  } |

【选 项】

A、11,22 B、12,23 C、11,23 D、10,23

【正确答案】 C

【解释】 无

【题目编号】 CSD-02-203-1005

【知 识 点】 预处理、const、static与sizeof - static关键字 - static变量的生命周期

【难易程度】 60

【题目描述】

下面程序的输出结果是( )。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | #include<iostream>  void fun(int i);  int main()  {  int n=2;  fun(n);  return 0;  }  void fun(int i)  {  static int x=1;  int y=1;  if(i>0)  {  ++x;  ++y;  cout<<x<<' '<<y<<' ';  fun(i-1);  }  } |

【选 项】

A、语法错误，不能输出正确结果

B、2 2 2 2

C、2 2 3 3

D、2 2 3 2

【正确答案】 D

【解释】 无

【题目编号】 CSD-02-203-1006

【知 识 点】 预处理、const、static与sizeof - static关键字 - static变量

【难易程度】 60

【题目描述】

下面程序的正确执行结果为。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #include<iostream>  int f(int);  void main()  {  int a=2,i;  for(i=0;i<3;i++)  cout<<f(a)<<" ";  cout<<endl;  }  int f(int a)  {  int b=0;  static int c=3;  b++;  c++;  return (a+b+c);  } |

【选 项】

A、7 7 7 B、7 10 13 C、7 9 11 D、7 8 9

【正确答案】 D

【解释】 无

【题目编号】 CSD-02-203-4001

【知 识 点】 预处理、const、static与sizeof - static关键字 - static作用的理解

【难易程度】 70

【题目描述】

static有什么作用？至少说明两种。

【正确答案】

在函数体，一个被声明为静态的变量在函数被调用的过程中维持其值不变。

在模块内（但在函数体外），一个被声明为静态的变量可以被模块内所有函数访问，但不能被模块外其他函数访问。它是一个本地的全局变量。

在模块内，被声明为静态的函数只能被这一模块内的其他函数调用。即函数被限制在声明它的模块范围内。

【解释】 无

【题目编号】 CSD-02-203-4002

【知 识 点】 预处理、const、static与sizeof - static关键字 - static的作用的理解。

【难易程度】 70

【题目描述】

static全局变量与普通全局变量有什么区别？static局部变量和普通局部变量有什么区别？static函数与普通函数有什么区别？

【正确答案】

static全局变量与普通全局变量的区别是：static全局变量只初始化一次，防止在其他文件单元中被引用。

static局部变量和普通局部变量的区别是：static局部变量只被初始化一次，下一次依据上一次结果值。

static函数与普通函数的区别是：static函数在内存中只有一份，普通函数在每个被调用中维持一份复制品。

【解释】

全局变量的声明之前加上static就构成了静态的全局变量。全局变量本身就是静态存储变量，静态全局变量当然也是静态存储变量。这两者在存储方式上并无不同。这两者的区别在于非静态全局变量的作用域是整个源程序，当源程序由多个源文件组成时，非静态的全局变量在各个源文件中都是有效的，而静态全局变量则限制了其作用域，即只在定义该变量的源文件内有效，在同一源程序的其他源文件中不能使用它。由于静态全局变量的作用域局限干一个源文件内，只能为该源文件内的函数公用，因此可以避免在其他源文件中引起错误。

从以上分析可以看出，把局部变量改变为静态变量后是改变了它的存储方式，即改变了它的生存期。把全局变量改变为静态变量后是改变了它的作用域，即限制了它的使用范围。static函数与普通函数作用域不同，其作用域仅在本文件。只在当前源文件中使用的函数应该说明为内部函数（static)，内部函数应该在当前源文件中说明和定义。对于可在当前源文件以外使用的函数，应该在一个头文件中说明，要使用这些函数的源文件要包含这个头文件。

【题目编号】 CSD-02-203-4003

【知 识 点】 预处理、const、static与sizeof - static关键字 - 静态成员的理解

【难易程度】 60

【题目描述】

分析代码写结果。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | #include <iostream>  class widget  {  public:  widget()  {  count++;  }  ~widget()  {  --count;  }  static int num()  {  return count;  }  private:  static int count;  };  int widget::count = 0;  int main()  {  widget x, y;  cout << "The Num is " << widget::num() << endl;  if(widget::num() > 1)  {  widget x, y, z;  cout << "The Num is " << widget::num() << endl;  }  widget z;  cout << "The Num is " << widget::num() << endl;    return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3 | The Num is 2  The Num is 5  The Num is 3 |

【解释】

类widget有一个静态成员count和一个静态方法num()。类中的静态成员或方法不属于类的实例，而属于类本身并在所有类的实例间共享。在调用它们时应该用类名加上操作赋”::”来引用。

在代码第7行，类widget的构造方法里把静态成员count的值加1，在代码第11行类widget的析构方法里把静态成员count的值减1。也就是说静态成员count的值表示类widget实例的个数。

通过以上的分析，可以看到运行到代码第26行时，只有两个类widget的实例，运行到代码第30行时，又产生了3个实例，这3个实例在第31行结束后被销毁。最后运行到代码第32行又产生了1个实例。

## 2.4 sizeof操作符

sizeof是一种单目操作符，而不是函数。sizeof操作符以字节形式给出了其操作数的存储空间。操作数可以是一个表达式或括在括号内的类型名。操作数的存储空间由操作数的类型决定。

【题目编号】 CSD-02-204-4001

【知 识 点】 预处理、const、static与sizeof - sizeof关键字 - sizeof的使用

【难易程度】 70

【题目描述】

sizeof计算类对象所占空间大小。在32位操作系统下。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65 | #include <iostream>  class A  {  public:  int i;  };  class B  {  public:  char ch;  };  class C  {  public:  int i;  short j;  };  class D  {  public:  int i;  short j;  char ch;  };  class E  {  public:  int i;  int ii;  short j;  char ch;  char chr;  };  class F  {  public:  int i;  int ii;  int iii;  short j;  char ch;  char chr;  };  int main()  {  cout << "sizeof(int) = " << sizeof(int) << endl;  cout << "sizeof(short) = " << sizeof(short) << endl;  cout << "sizeof(char) = " << sizeof(char) << endl;  cout << endl;  cout << "sizeof(A) = " << sizeof(A) << endl;  cout << "sizeof(B) = " << sizeof(B) << endl;  cout << "sizeof(C) = " << sizeof(C) << endl;  cout << "sizeof(D) = " << sizeof(D) << endl;  cout << "sizeof(E) = " << sizeof(E) << endl;  cout << "sizeof(F) = " << sizeof(F) << endl;    return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | sizeof(A) = 4  sizeof(B) = 1  sizeof(C) = 8  sizeof(D) = 8  sizeof(E) = 12  sizeof(F) = 16 |

【解释】

这里定义了6个类，很多人遇到对结构体或类做sizeof运算时，只是简单的把各个成员所占的内存数量相加。在32位操作系统环境下，char占1个字节，int占4个字节，short

占2个字节，类F占15个字节。但实事并非如此。

这个问题会使许多初学者疑惑不解，而这种情况是由于字节对齐引起的。由于各个硬件平台对存储空间的处理上有很大的不同。一些平台对某些特定类型的数据只能从某些特定地址开始存取，然而其他平台可能没有这种情况，为了提高效率，要求工作对数据进行对齐。例如有些平台每次都是从偶地址开始读入，如果一个int型（假设为32位系统）存放在偶地址开始的地方，那么一个读周期就可以读出，而存放在奇地址开始的地方，会需要2个读周期，并对两次读出的结果的高低字节进行拼凑才能得到该int数据。显然读取效率会低很多。

通常写程序的时候，不需要考虑对齐问题。编译器会替我们选择适合目标平台的对齐策略。当然，也可以通过给编译器传递预编译指令而改变对指定数据的对齐方法。

字节对齐的细节与编译器实现相关，一般而言，需要满足3个准则：（1）结构体变量的首地址能够被其最宽的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晕（offset）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的基本类型成员的整数倍，如有需要编译器会在最末一个成员之后加上填充字节。

【题目编号】 CSD-02-204-4002

【知 识 点】 预处理、const、static与sizeof - static关键字 - sizeof的使用

【难易程度】 70

【题目描述】

分析代码写结果一一使用sizeof计算含有虚函数的类对象的空间大小。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75 | #include <iostream>  using namespace std;  class Base  {  public:  Base(int x) : a(x)  {  }  void print()  {  cout << "base" << endl;  }  private:  int a;  };  class Derived : public Base  {  public:  Derived(int x) : Base(x - 1), b(x)  {  }  void print()  {  cout << "derived" << endl;  }  private:  int b;  };  class A  {  public:  A(int x) : a(x)  {  }  virtual void print()  {  cout << "A" << endl;  }  private:  int a;  };  class B : public A  {  public:  B(int x) : A(x - 1), b(x)  {  }  virtual void print()  {  cout << "B" << endl;  }  private:  int b;  };  int main()  {  Base obj1(1);  cout << "size of Base obj is " << sizeof(obj1) << endl;  Derived obj2(2);  cout << "size of Base obj is " << sizeof(obj2) << endl;  A a(1);  cout << "size of Base obj is " << sizeof(a) << endl;  B b(2);  cout << "size of Base obj is " << sizeof(b) << endl;  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4 | size of Base obj is 4  size of Base obj is 8  size of Base obj is 8  size of Base obj is 12 |

【解释】

上面这个程序定义了4个类，分别是Base、Derived、A和B。其中Derived类是Base的子类，B是A的子类。

对于Base类来说，sizeof(int)等于4，print()函数不占内存。

对于Derived类来说，比Base类多一个整型成员，因而多4字节，一共是12字节。

对于A类来说，由于它含有虚函数，因此占用的内存除了一个整型变量之外，还包括一个隐含的虚表指针成员，一共是8字节。

对于B类来说，比A类多一个整型成员，因而多4字节，一共是12字节。

通过这个例子可以看出，普通函数不占用内存，只有虚函数会占用一个指针大小的内存，原因是系统用一个指针维护这个类的虚函数表，并且注意这个虚函数无论含有多少项（类中含有多少个虚函数）都不会影响类的大小。

【题目编号】 CSD-02-204-4003

【知 识 点】 预处理、const、static与sizeof - static关键字 - sizeof的使用。

【难易程度】 80

【题目描述】

分析代码写结果——使用sizeof计算虚拟继承的类对象的空间大小。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | #include <iostream>  using namespace std;  class A  {  };  class B  {  };  class C : public A, public B  {  };  class D : virtual public A  {  };  class E : virtual public A, virtual public B  {  };  class F  {  public:  int a;  static int b;  };  int F::b = 10;  int main()  {  cout << "sizeof(A) = " << sizeof(A) << endl;  cout << "sizeof(B) = " << sizeof(B) << endl;  cout << "sizeof(C) = " << sizeof(C) << endl;  cout << "sizeof(D) = " << sizeof(D) << endl;  cout << "sizeof(E) = " << sizeof(E) << endl;  cout << "sizeof(F) = " << sizeof(F) << endl;  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | sizeof(A) = 1  sizeof(B) = 1  sizeof(C) = 1  sizeof(D) = 4  sizeof(E) = 4  sizeof(F) = 4 |

【解释】

代码第35行，由于A是空类，编译器会安插一个char空类，标记它的每一个对象，因此其大小为1字节。

代码第36行，类B大小和A相同，都是1字节。

代码第37行，类C是多重继承自A和B，其大小仍然为1。

代码第38行，类D是虚继承自A，编译器为该类安插一个指向父类的指针，指针大小为4。由于此类有了指针，编译器不会安插一个char了，因此其大小是4字节。

代码第39行，类E虚继承自A并且也虚继承自B，因此它有指向父类A的指针与父类B的指针，加起来大小为8字节。注意：在GNU编译器下编译是4字节。

代码第40行，类F含有一个静态成员变量，这个静态成员的空间不在类的实例中，而

是像全局变量一样在静态存储区中，被类共享，因此其大小是4字节。

【题目编号】 CSD-02-204-4004

【知 识 点】 预处理、const、static与sizeof - static关键字 - sizeof与strlen的区别

【难易程度】 70

【题目描述】

sizeof与strlen有哪些区别？

【正确答案】

sizeof是算符，strlen是函数。

sizeof操作符的结果类型是size\_t，它在头文件中typedef为unsigned int类型，该类型保证能容纳实现所建立的最大对象的字节大小。

sizeof可以用类型做参数，strlen只能用char＊做参数，且必须是以”\0”结尾。

数组做sizeof的参数不退化，但是传递给strlen就退化为指针了。

大部分编译程序在编译的时候sizeof就被计算过了，这就是sizeof(x)可以用来定义数组维数的原因。strlen的结果要在运行的时候才能计算出来，它用来计算字符串的长度，不是类型占内存的大小。

sizeof后如果是类型必须加括弧，如果是变量名可以不加括弧。这是因为sizeof是个操作符不是个函数。

在计算字符串数组的长度上有区别。例如：

char str[20] = "0123456789";

int a = strlen(str);

int b = sizeof(str);

a计算的是以0x00结束的字符串的长度（不包括0x00结束符），这里结果是10。b计算的则是分配的数组strl20］所占的内存空间的大小，不受里面存储内容的改变而改变，这里结果是20。

如果要计算指针指向的字符串的长度，则一定要使用strlen。例如：

char \*ss = "0123456789";

int a = sizeof(ss);

int b = strlen(ss);

a计算的是ss指针占用的内存空间大小，这里结果是4。b计算的是ss指向的字符串的长度，这里结果是10。

【解释】 无

【题目编号】 CSD-02-204-4005

【知 识 点】 预处理、const、static与sizeof - static关键字 - 理解sizeof的用途

【难易程度】 60

【题目描述】

sizeof有哪些用途？

【正确答案】

与存储分配和I/O系统的例程进行通信。

查看某个类型的对象在内存中所占的单元字节。

void\* memset(void\* s, int c, sizeof(s));

动态分配对象时，可以使系统知道要分配多少内存。

便于一些类型的扩充，在windows中很多结构类型有一个专用的字段是用来存放该类型的字节大小。

由于操作数的字节数在实现时可能出现变化，建议在涉及操作数字节大小时用sizeof来代替常量计算。

如果操作数是函数中的数组形参或函数类型的形参，sizeof给出其指针的大小。

【解释】 无

【题目编号】 CSD-02-204-4006

【知 识 点】 预处理、const、static与sizeof - static关键字 - sizeof计算联合体大小

【难易程度】 60

【题目描述】

分析代码写结果。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | #include<iostream>  union u  {  double a;  int b;  };  union u2  {  char a[13];  int b;  };  union u3  {  char a[13];  char b;  };  int main()  {  std::cout << sizeof(u) << std::endl;  std::cout << sizeof(u2) << std::endl;  std::cout << sizeof(u3) << std::endl;  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3 | 8  16  13 |

【解释】

这个程序定义了3个联合体：u、u2和u3。联合体的大小取决于它所有的成员中占用空间最大的一个成员的大小，并且对于复合数据类型，例如union、struct、class，对齐方式为成员中最大的成员对齐方式。

对于u来说，大小取决于最大的double类型成员a，即sizeof(u)=sizeof(double)=8。

对于u2来说，最大的空间是char[13]型的数组。这里要注意，由于它的另一个成员int b的存在，使u2的对齐方式变成4，也就是说u2的大小必须在4倍数的对界上，所以占用的空间变为最接近13的对界，即16。

对于u3来说，最大的空间是char[13]类型的数组，即sizeof(u3)=13。

这里又出现了CPU对齐的问题。编译器会尽量把成员对齐以提高内存的命中率。对齐是可以更改的，使用”#pragma pack(x)”可以改变编译器的对齐方式。C++固有类型的对界取编译器对齐方式与自身大小中较小的一个。例如，指定编译器按2对齐，int类型的大小是4，则int的对界为2和4中较小的2。在默认的对界方式下，几乎所有的数据类型都不大于默认的对界方式8（除了long和double)，因此所有的固有类型的对齐方式可以认为就是类型自身的大小。

【题目编号】 CSD-02-204-4007

【知 识 点】 预处理、const、static与sizeof - static关键字 - #pragma pack指令的作用

【难易程度】 80

【题目描述】

分析代码写结果。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #include <iostream>  using namespace std;  #pragma pack(1)  struct test{  char c;  short s1;  short s2;  int i;  };  int main()  {  cout << sizeof(test) << endl;    return 0;  } |

【正确答案】 9

【解释】

代码第3行用“#pragma pack”将对齐设为1．由于结构体test中的成员s1、s2和i的自身对齐为分别为2、2和4，都大于1．因此它们都是用1作为对齐，sizeof(test) = 1+2+2+4 = 9。

如果注释代码第3行，则编译器默认对齐为8。各个成员自身的对齐都小于8，因此它们使用自身的对齐，sizeof(test) = 1 + 1（补齐）+ 2 + 2 + 2（补齐）+ 4 = 12。

## 2.5 inline与宏定义

inline（内联函数）是C++引入的机制，其目的是解决使用宏定义的一些缺点。面试中求职者经常会被问及inline和宏定义之间的区别。显然，这类的题目需要求职者对于C的宏定义以及C++的inline知识都十分清楚，掌握各自的特点以及区别。

【题目编号】 CSD-02-205-1001

【知 识 点】 预处理、const、static与sizeof - inline与宏定义 - 内联函数

【难易程度】 60

【题目描述】

下面说法正确的是（ ）

【选 项】

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

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关键字inline定义

【正确答案】 B

【解释】 无

【题目编号】 CSD-02-205-4001

【知 识 点】 预处理、const、static与sizeof - inline与宏定义 - 理解内联函数的作用。

【难易程度】 60

【题目描述】

为什么要引入内联函数？

【正确答案】

引入内联函数的主要目的是用它替代C中表达式形式的宏定义，解决程序中函数调用的效率问题。在C语言里可以使用宏定义，宏定义在形式及使用上像一个函数，但它使用预处理器实现，没有了参数压栈、代码生成等一系列的操作，因此效率很高。这种宏定义在形式上类似于一个函数，但在使用它时，只是做预处理器符号表中的简单替换，因此它不能进行参数有效性的检测，也就不能享受C++编译器严格类型检查的好处，另外它的返回值也不能被强制转换为可转换的合适类型，这样，它的使用就存在着一系列的隐患和局限性。

另外，在C++中引入了类及类的访问控制，这样，如果一个操作或者说一个表达式涉及类的保护成员或私有成员，你就不可能使用这种宏定义来实现（因为无法将this指针放在合适的位置）。

inline推出的目的是为了取代这种表达式形式的宏定义，消除了宏定义的缺点，同时又很好地继承了它的优点。

【解释】 无

【题目编号】 CSD-02-205-4002

【知 识 点】 预处理、const、static与sizeof - inline与宏定义 - 内联函数与宏定义。

【难易程度】 70

【题目描述】

为什么inline能很好地取代表达式形式的预定义呢？

【正确答案】

inline定义类的内联函数，函数代码被放入符号表中，在使用时直接进行替换（像宏一样展开），没有了调用的开销，效率很高。

类的内联函数也是一个真正的函数。编译器在调用一个内联函数时，首先会检查它的参数的类型，保证调用正确，然后再进行一系列的相关检查，就像对待任何一个真正的函数一样。这样就消除了它的隐患和局限性。

inline可以作为某个类的成员函数，当然可以使用所在类的保护成员及私有成员。

【解释】 无

【题目编号】 CSD-02-205-4003

【知 识 点】 预处理、const、static与sizeof - inline与宏定义 - 内联函数的作用场合。

【难易程度】 70

【题目描述】

说明内联函数使用的场合。

【正确答案】

首先使用inline函数可以完全取代表达式形式的宏定义。

内联函数在C++类中应用最广的，是用来定义存取函数。定义的类中一般会把数据成员定义成私有的或者保护的，这样，外界就不能直接读写类成员的数据了。对于私有或者保护成员的读写就必须使用成员接口函数来进行。如果把这些读写成员函数定义成内联函数的话，将会获得比较好的效率。

【解释】 无

【题目编号】 CSD-02-205-4004

【知 识 点】 预处理、const、static与sizeof - inline与宏定义 - 内联函数的缺点。

【难易程度】 60

【题目描述】

为什么不把所有的函数都定义成内联函数？

【正确答案】

内联是以代码膨胀（复制）为代价，仅仅省去了函数调用的开销，从而提高函数的执行效率。一方面，如果执行函数体内代码的时间相比于函数调用的开销较大，那么效率的收获会很少。另一方面，每一处内联函数的调用都要复制代码，将使程序的总代码量增大，消耗更多的内存空间。以下情况不宜使用内联。

如果函数体内的代码比较长，使用内联将导致内存消耗代价较高。

如果函数体内出现循环，那么执行函数体内代码的时间要比函数调用的开销大。

另外，类的构造函数和析构函数容易让人误解成使用内联更有效。要当心构造函数和析构函数可能会隐藏一些行为，如“偷偷地”执行了基类或成员对象的构造函数和析构函数。所以不要随便地将构造函数和析构函数的定义体放在类声明中。

一个好的编译器将会根据函数的定义体，自动地取消不值得的内联（这说明了inline不应该出现在函数的声明中）。

【解释】 无

【题目编号】 CSD-02-205-4005

【知 识 点】 预处理、const、static与sizeof - inline与宏定义 - 内联函数与宏定义。

【难易程度】 60

【题目描述】

内联函数与宏有什么区别？

【正确答案】

内联函数在编译时展开，宏在预编译时展开。

在编译的时候内联函数可以直接被嵌入到目标代码中，而宏只是一个简单的文本替换。

内联函数可以完成诸如类型检测、语句是否正确等编译功能，宏就不具有这样的功能。

宏不是函数，inline函数是函数。

宏在定义时要小心处理宏参数（一般情况是把参数用括号括起来），否则容易出现二义性。而内联函数定义时不会出现二义性。

【解释】 无

# 第3篇 引用和指针

引用是C++常用的重要内容之一，正确、灵活地使用引用，可以使程序简洁、高效。

指针是C语言中广泛使用的一种数据类型。使用指针可以编写出精练而高效的程序。学习指针是学习C语言最重要的一环，同时，也是最为困难的一环。

指针问题是各公司针对求职者的重点考点，常见问题涉及数组指针、函数指针、常量指针、指针传值、多维指针等。

本章基于求职者实际会遇到的公司面试题，对指针的各个方面的重点难点进行全面且细致的分析。通过阅读本章，读者能解决在运用指针时可能面对的各个难点。

## 3.1 引用

引用（reference)，又称为别名（alias)，可以作为某一对象的另一个名字，是由C++引入的特性。通过引用可以间接地操纵对象，其使用方式类似于指针，但是不需要指针的语法。

【题目编号】 CSD-03-301-1001

【知 识 点】 引用和指针 - 引用 - 引用

【难易程度】 60

【题目描述】

下列( )的调用方式是引用调用。

【选 项】

A、形参是指针，实参是地址值

B、形参和实参都是变量

C、形参是数组，实参是变量

D、形参是引用，实参是变量

【正确答案】 D

【解释】

函数参数是引用时，传递类型相同的实参时，相当于给实参起了一个别名。

【题目编号】 CSD-03-301-1002

【知 识 点】 引用和指针 - 引用 - 引用声明

【难易程度】 60

【题目描述】

下列引用的定义中，（ ）是错误的。

【选 项】

A、int i; int &j=i;

B、int i; int &j; j=i;

C、float i; float &j=i;

D、char d; char &k=d;

【正确答案】 B

【解释】

引用被声明时必须初始化。

【题目编号】 CSD-03-301-1003

【知 识 点】 引用和指针 - 引用 - 引用

【难易程度】 60

【题目描述】

（ ）是给对象取了一个别名，他引入了对象的同义词。

【选 项】

A、指针

B、引用

C、枚举

D、结构

【正确答案】 B

【解释】 无

【题目编号】 CSD-03-301-1004

【知 识 点】 引用和指针 - 引用 - 引用

【难易程度】 60

【题目描述】

下列表示引用的方法中，（ ）是正确的。已知：int m=10;

【选 项】

A、int &x=m;

B、int &y=10;

C、int &z;

D、float &t=&m;

【正确答案】 A

【解释】 无

【题目编号】 CSD-03-301-1005

【知 识 点】 引用和指针 - 引用 - 引用

【难易程度】 60

【题目描述】

下面程序的运行结果是（ ）。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | #include<iostream>  void fun(int &a,int &b)  {  int p;  p=a;  a=b;  b=p;  }  void exchange(int &a,int &b,int &c)  {  if(a<b)fun(a,b);  if(a<c)fun(a,c);  if(b<c)fun(b,c);  }  void main()  {  int a,b,c;  a=12;  b=639;  c=78;  exchange(a,b,c);  cout<<"a="<<a<<",b="<<b<<",c="<<c<<endl;  } |

【选 项】

A、a=639,b=78,c=12

B、a=12,b=639,c=78

C、a=639,b=12,c=78

D、a=78,b=12,c=639

【正确答案】 A

【解释】

fun()函数交换传入的两个参数的值

exchange()函数把传入的参数按照从大到小的顺序排列

【题目编号】 CSD-03-301-1006

【知 识 点】 引用和指针 - 引用 - 引用

【难易程度】 60

【题目描述】

下列表示引用的方法中，（ ）是正确的。已知：int k=1000;

【选 项】

A、int &x=k; B、char &y; C、int &z=1000; D、float &t=&k;

【正确答案】 A

【解释】

B声明引用时必须赋值

C非const引用不能用常量赋值

D 类型不同

【题目编号】 CSD-03-301-1007

【知 识 点】 引用和指针 - 引用 - 引用

【难易程度】 60

【题目描述】

下面说法错误的选项是（ ）。

【选 项】

A、引用表达式是一个左值表达式，因此它可以出现在形参和实参的任何一方

B、若一个函数返回了引用，那么该函数的调用也可以被赋值

C、所有函数都可以返回引用

D、引用返回值时，不生成值的副本

【正确答案】 C

【解释】

比如析构函数。

【题目编号】 CSD-03-301-1008

【知 识 点】 引用和指针 - 引用 - 引用

【难易程度】 70

【题目描述】

下列程序的执行结果为（ ）。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | #include<iostream>  int mian()  {  int n=10;  int\* pn=&n;  int\* &rn=pn;  (\*pn)++;  cout<<"n="<<n<<endl;  (\*rn)++;  cout<<”n=”<<n<<endl;  } |

【选 项】

A、 n=11 n=12

B、 n=11 n=11

C、 n=10 n=11

D、 n=11 n=10

【正确答案】 A

【解释】

(\*pn)++ 相当于 n++ ，所以n的值被修改为11。

同理，(\*rn)++ 相当于再一次的n++，所以n的值被修改为12。

【题目编号】 CSD-03-301-4001

【知 识 点】 引用和指针 - 引用 - 引用

【难易程度】 60

【题目描述】

将“引用”作为函数返回值类型的格式、好处和需要遵守的规则？

【正确答案】

格式：类型标识符 &函数名（形参列表及类型说明）{ // 函数体 }

好处：在内存中不产生被返回值的副本；（注意：正是因为这点原因，所以返回一个局部变量的引用是不可取的。因为随着该局部变量生存期的结束，相应的引用也会失效，产生错误！）

规则：

（1）不能返回局部变量的引用。主要原因是局部变量会在函数返回后被销毁，因此被返回的引用就成为了无所指“的引用，程序会进入未知状态。 当一定要返回时，可以将局部变量声明为static。

（2）可以返回函数内部new分配的内存的引用，但是注意需要手动释放。（见下例）

（3）可以返回类成员的引用，但最好是const。

（4）流操作符重载返回值申明为“引用”的作用：流操作符 << 和 >>，这两个操作符常常希望被连续使用。

（5）在另外的一些操作符中，却千万不能返回引用：+、-、\*、/ 四则运算符。它们不能返回引用，主要原因是这四个操作符没有side effect，因此，它们必须构造一个对象作为返回值，可选的方案包括：返回一个对象、返回一个局部变量的引用，返回一个new分配的对象的引用、返回一个静态对象引用。根据前面提到的引用作为返回值的三个规则，第2、3两个方案都被否决了。静态对象的引用又因为((a+b) == (c+d))会永远为true而导致错误。所以可选的只剩下返回一个对象了。

【解释】 无

【题目编号】 CSD-03-301-4002

【知 识 点】 引用和指针 - 引用 - 引用

【难易程度】 80

【题目描述】

“引用”与多态的关系？

【正确答案】

引用是除指针外另一个可以产生多态效果的手段。这意味着，一个基类的引用可以指向它的派生类实例。

【解释】 无

【题目编号】 CSD-03-301-4003

【知 识 点】 引用和指针 - 引用 - 一般变量引用。

【难易程度】 60

【题目描述】

分析代码写结果。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | #include <iostream>  #include <string>  using namespace std;  int main(int argc, char \*argv[])  {  int a = 10;  int b = 20;  int &rn = a;  int equal;  rn = b;  cout << "a = " << a << endl;  cout << "b = " << b << endl;  rn = 100;  cout << "a = " << a << endl;  cout << "b = " << b << endl;  equal = (&a == &rn) ? 1 : 0;    cout << "equal = " << equal << endl;  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5 | a = 20  b = 20  a = 100  b = 20  equal = 1 |

【解释】 无

【题目编号】 CSD-03-301-4004

【知 识 点】 引用和指针 - 引用 - 指针变量引用

【难易程度】 70

【题目描述】

分析代码写结果。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | #include <iostream>  using namespace std;  int main(int argc, char \*argv[])  {  int a = 1;  int b = 10;  int \*p = &a;  int\* &pa = p;  (\*pa)++;  cout << "a = " << a << endl;  cout << "b = " << b << endl;  cout << "\*p = " << \*p << endl;  pa = &b;  (\*pa)++;  cout << "a = " << a << endl;  cout << "b = " << b << endl;  cout << "\*p = " << \*p << endl;  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | a = 2  b = 10  \*p = 2  a = 2  b = 11  \*p = 11 |

【解释】

代码第5行和第6行，整型变量a和整型变量b分别被初始化为1和10。

代码第7行，声明整型的指针变量p并初始指向a。

代码第8行，声明p的一个指针引用pa。

代码第10行，将pa指向的内容加1。由于pa是p的引用，所以此时实际上是对p指向的内容加1，也就是a加1，结果为a变成了2。

代码第15行，将pa指向变量b的地址，由于pa是p的引用，所以此时p也指向了b的地址。

代码第16行，将pa指向的内容加1．由于pa是p的引用，所以此时实际上是对p指向的内容加1，也就是b加1，结果为b变成了12。

【题目编号】 CSD-03-301-4006

【知 识 点】 引用和指针 - 引用 - 引用

【难易程度】 60

【题目描述】

将“引用”作为函数参数有哪些特点？

【正确答案】

（1）传递引用给函数与传递指针的效果是一样的。这时，被调函数的形参就成为原来主调函数中的实参变量或对象的一个别名来使用，所以在被调函数中对形参变量的操作就是对其相应的目标对象(在主调函数中)的操作。

（2）使用引用传递函数的参数,在内存中并没有产生实参的副本,它是直接对实参操作;而使用一般变量传递函数的参数,当发生函数调用时,需要给形参分配存储单元,形参变量是实参变量的副本;如果传递的是对象,还将调用拷贝构造函数。因此,当参数传递的数据较大时,用引用比用一般变量传递参数的效率和所占空间都好。

（3）使用指针作为函数的参数虽然也能达到与使用引用的效果.但是,在被调函数中同样要给形参分配存储单元,且需要重复使用"\*指针变量名"的形式进行运算,这很容易产生错误且程序的阅读性较差;另一方面，在主调函数的调用点处,必须用变量的地址作为实参。而引用更容易使用,更清晰。

【解释】 无

【题目编号】 CSD-03-301-4007

【知 识 点】 引用和指针 - 引用 - 参数引用

【难易程度】 60

【题目描述】

交换两个字符串―参数引用。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  > File Name: swap\_ref.cpp  > Author: Robean Zhou  > Mail: zhouyb@tarena.com.cn  > Created Time: 2014.10.16  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include <iostream>  #include <string>  using namespace std;  void swap(char \*&x, char \*&y)  {  char \*temp;  temp = x;  x = y;  y = temp;  }  int main()  {  const char \*ap = "hello";  const char \*bp = "how are you?";  cout << "ap: " << ap << endl;  cout << "bp: " << bp << endl;  swap(ap, bp);  cout << "swap ap, bp" << endl;  cout << "ap: " << ap << endl;  cout << "bp: " << bp << endl;  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5 | ap: hello  bp: how are you?  swap ap, bp  ap: how are you?  bp: hello |

【解释】 无

【题目编号】 CSD-03-301-4008

【知 识 点】 引用和指针 - 引用 - 常引用

【难易程度】 60

【题目描述】

在什么时候需要使用“常引用”？

【正确答案】

如果既要利用引用提高程序的效率，又要保护传递给函数的数据不在函数中被改变，就应使用常引用。常引用声明方式：const 类型标识符 &引用名 = 目标变量名；

【解释】 无

【题目编号】 CSD-03-301-4009

【知 识 点】 引用和指针 - 引用 - 参数引用

【难易程度】 60

【题目描述】

程序查错。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | #include <iostream>  using namespace std;  class Test  {  public:  void f(const int &arg);  private:  int value;  };  void Test::f(const int &arg){  arg = 10;  cout << "arg = " << arg << endl;  value = 20;  }  int main()  {  int a = 7;  const int b = 10;  int &c = b;  const int &d = a;    a++;  b++;    Test test;  test.f(a);  cout << "a = " << a << endl;    return 0;  } |

【正确答案】

代码第13行错误。原因是arg的值不能被修改。

代码第22行错误。原因是常量类型变量不能定义非常量的引用。

代码第26行错误。原因是不能使用常量引用修改变量的值。

【解释】 无

【题目编号】 CSD-03-301-4010

【知 识 点】 引用和指针 - 引用 -引用和指针的区别。

【难易程度】 80

【题目描述】

指针和引用有什么区别？

【正确答案】

初始化要求不同。引用在创建的同时必须初始化，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。

可修改性不同。引用一旦被初始化，它就不能被另一个对象引用：而指针在任何时候都可以指向另一个对象。

不存在NULL引用。引用不能使用指向空值的引用，它必须指向某个对象；而指针则可以是NULL，不需要总是指向某些对象，可以把指针一指向任意对象，所以指针更加灵活，也容易出错。

测试时的区别。由于引用不会指向空值，这意味着使用引用之前不需要测试它的合法性；而指针则需要经常进行测试。因此使用引用的代码效率比使用指针的要高。

应用的区别。如果指向一个对象后就不会改变指向，那么应该使用引用。如果指向NULL（不指向任何对象）或在不同的时刻指向不同的对象，应该使用指针。

实际上，在二进制层面，引用·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。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。

【解释】 无

【题目编号】 CSD-03-301-4011

【知 识 点】 引用和指针 - 引用 - 引用和指针的区别。

【难易程度】 70

【题目描述】

为什么引用比指针安全？

【正确答案】

由于不存在空引用，并且引用一旦被初始化指向一个对象，它就不能被改变为另一个对象的引用，因此引用很安全。指针可以随时指向别的对象，并且可以不被初始化，或初始化为，所以不安全。

【解释】 无

## 3.2 指针基础

C语言之所以功能强大，很大部分是因为它有灵活的指针运用。C++作为从C语言发展而来的一门高级语言，并没有摒弃指针，而是积极地采纳它，与此同时又注入了面向对象的高级思想，这使得C++成为了最优秀的语言之一。

指针的概念很简单，可以把它看做一种数据类型，定义指针变量就像定义int、char型变最一样，只不过int型变量存储整数，char型变量存储字符，而指针存储的是内存地址。

【题目编号】 CSD-03-302-1001

【知 识 点】 引用和指针 - 指针基础 - 指针

【难易程度】 70

【题目描述】

下面程序的正确执行结果为( )。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | #include<iostream>  void fun(char \*\*m)  {  m++;  cout<<\*m<<endl;  }  void main()  {  static char \*a[]={"MORNING","AFTERNOON","EVENING"};  char \*\*n;  n=a;  fun(n);  } |

【选 项】

A、为空 B、MORNING C、AFTERNOON D、EVENING

【正确答案】 C

【解释】

m++相当于a+1，指向数组的下一个元素。

【题目编号】 CSD-03-302-1002

【知 识 点】 引用和指针 - 指针基础 - 指针

【难易程度】 60

【题目描述】

对于指针，下（ ）是正确的。

【选 项】

A、声明为void类型的指针可以指向任何类型的对象，并且通过该指针可以间接访问它所指向的对象。

B、如果两个指针的类型不是同一个类型，它们就不能相互赋值。（void类型的指针可以与其它类型的指针相互赋值）。

C、一个int 类型的指针既可以指向int类型的对象，又可以指向int类型的函数。

D、数组名就是指针。

【正确答案】 D

【解释】

A不能用void指针直接访问对象

B不同类型之间的指针可以互相赋值

C指向函数的需要是函数指针

【题目编号】 CSD-03-302-1003

【知 识 点】 引用和指针 - 指针基础 - 指针

【难易程度】 70

【题目描述】

下列说明中const char \* ptr; ptr 应该是( )。

【选 项】

A、指向字符的指针

B、指向字符的常量指针

C、指向字符串常量的指针

D、指向字符串的常量指针

【正确答案】 C

【解释】 无

【题目编号】 CSD-03-302-1004

【知 识 点】 引用和指针 - 指针基础 - 类成员指针

【难易程度】 70

【题目描述】

运算符->\*的功能是( )。

【选 项】

A、用来表示指向对象指针对指向类成员指针的操作。

B、用来表示对象类成员的操作。

C、用来表示指向对象指针对类成员的操作。

D、用来表示对象指向类成员指针的操作。

【正确答案】 A

【解释】 无

【题目编号】 CSD-03-302-1005

【知 识 点】 引用和指针 - 指针基础 - 类成员指针

【难易程度】 70

【题目描述】

已知p是一个指向类X数据成员m的指针,s是类X的一个对象。如果要给m赋值为3,( )是正确的。

【选 项】

A、s.p=3

B、s->p=3

C、s.\*p=3

D、\*s.p=3

【正确答案】 C

【解释】 无

【题目编号】 CSD-03-302-1006

【知 识 点】 引用和指针 - 指针基础 - 类成员指针

【难易程度】 70

【题目描述】

已知f1(int)是类X的公有成员函数,p是指向成员f1()的指针，采用它赋值, ( )是正确的。

【选 项】

A、p=f1

B、p=X::f1

C、p=x::f1

D、p=f1()

【正确答案】 B

【解释】

需要指定类作用域。

【题目编号】 CSD-03-302-1007

【知 识 点】 引用和指针 - 指针基础 - 参数传递

【难易程度】 70

【题目描述】

下列程序的输出结果是( )。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | #include<iostream>  void sub1(char a,char b)  {  char c;  c=a;  a=b;  b=c;  }  void sub2(char \*a,char b)  {  char c;  c=\*a;  \*a=b;  b=c;  }  void sub3(char \*a,char \*b)  {  char c;  c=\*a;  \*a=\*b;  \*b=c;  }  void main()  {  char a,b;  a='A';  b='B';  sub1(a,b);  putchar(a);  putchar(b);  a='A';  b='B';  sub2(&a,b);  putchar(a);  putchar(b);  a='A';  b='B';  sub3(&a,&b);  putchar(a);  putchar(b);  putchar('\n');  } |

【选 项】

A、ABABAB B、ABBBBA C、ABBABA D、ABABBA

【正确答案】 B

【解释】

考察指针值传递和地址传递问题。

【题目编号】 CSD-03-302-4001

【知 识 点】 引用和指针 - 指针基础 - 函数指针

【难易程度】 70

【题目描述】

请解释“func”为何种类型，这种类型的作用什么，变量ttt的值是多少？

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | typedef int (\*func)(int, int\*);  int xxx(int a, int \*p)  {  return a + \*p;  }  int dowork(func aaa, int bbb, int \*ccc)  {  return aaa(bbb, ccc);  }  int sss = 4;  int ttt = dowork(&xxx, 3, &sss); |

【正确答案】

func表示一个函数指针，它指向参数为int，int\*，返回值为int的函数。ttt的值为7。

【解释】 无

【题目编号】 CSD-03-302-4002

【知 识 点】 引用和指针 - 指针基础 - 复杂指针的声明

【难易程度】 80

【题目描述】

用变量a给出下面的定义：

（1）定义一个整型数。

（2）定义一个指向整型数的指针。

（3）定义一个指向指针的的指针，它指向的指针是一个指向整型数指针。

（4）定义一个有10个整型数的数组。

（5）定义一个有10个指针的数组，该指针是指向一个整型数的指针。

（6）定义一个指向有10个整型数数组的指针。

（7）定义一个指向函数的指针，该函数有一个整型参数并返回一个整型数。

（8）定义一个有10个指针的数组，该指针指向一个函数，该函数有一个整型参数并返回一个整型数。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | int a;  int \*a;  int \*\*a;  int a[10];  int \*a[10];  int (\*a)[10];  int (\*a)(int);  int (\*a[10])(int); |

【解释】 无

【题目编号】 CSD-03-302-4003

【知 识 点】 引用和指针 - 指针基础 - 用指针赋值。

【难易程度】 70

【题目描述】

分析代码写结果。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #include <stdio.h>  int main(void)  {  char a[] = "hello, world";  char \*ptr = a;  printf("%c\n", \*(ptr + 4));  printf("%c\n", ptr[4]);  printf("%c\n", a[4]);  printf("%c\n", \*(a + 4));  \*(ptr + 4) += 1;  printf("%s\n", ptr);  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5 | o  o  o  o  hellp, world |

【解释】 无

【题目编号】 CSD-03-302-4004

【知 识 点】 引用和指针 - 指针基础 - 指针加减操作。

【难易程度】 70

【题目描述】

分析代码写结果。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | #include <stdio.h>  int main(void)  {  int a[5] = {1, 2, 3, 4, 5};  int \*ptr = (int\*)(&a + 1);    printf("%d\n", \*(a + 1));  printf("%d\n", \*(ptr - 1));  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2 | 2  5 |

【解释】 无

【题目编号】 CSD-03-302-4005

【知 识 点】 引用和指针 - 指针基础 - 指针比较操作

【难易程度】 60

【题目描述】

分析代码写结果.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | #include <iostream>  using namespace std;  int main(void)  {  char str1[] = "abc";  char str2[] = "abc";  const char str3[] = "abc";  const char str4[] = "abc";  const char \*str5 = "abc";  const char \*str6 = "abc";  char \*str7 = "abc";  char \*str8 = "abc";  cout << (str1 == str2) << endl;  cout << (str3 == str4) << endl;  cout << (str5 == str6) << endl;  cout << (str6 == str7) << endl;  cout << (str7 == str8) << endl;  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5 | 0  0  1  1  1 |

【解释】

程序考查有关是内存中各个数据的存储方式。

数组str1、str2、str3和str4都是在栈中分配的，内存中的内容都是”abc”加一个”\0”，但是它们的存储位置不同，因此代码15行和16行的输出都是0。

指针str5、str6、str7和str8也是在栈中分配的，它们都指向”abc”字符串，注意”abc”存放在数据区，所以str5、str6、str7和str8指向同一块数据区的内存，因此17、18和19行的输出是1。

【题目编号】 CSD-03-302-4006

【知 识 点】 引用和指针 - 指针基础 - 指针操作内存违规

【难易程度】 70

【题目描述】

分析代码找错误。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #include <iostream>  #include <cstring>  using namespace std;  int main()  {  char a;  char \*str1 = &a;  char \*str2 = "AAA";  strcpy(str1, "hello");  cout << str1 << endl;  str2[0] = 'B';  cout << str2 << endl;  return 0;  } |

【正确答案】

代码第10行，str1指向一个字节大小的内存区。由于复制”hello”字符串最少需要6字节，显然内存大小不够。因此会因为越界进行内存读写而导致程序崩溃。

代码第13行，str2指向”AAA”这个字符串常量。因为是常量，所以对str2[0]的赋值操作是不合法的，也会导致程序崩溃。

【解释】 无

【题目编号】 CSD-03-302-4007

【知 识 点】 引用和指针 - 指针基础 - 指针常量与常量指针的区别

【难易程度】 60

【题目描述】

指针常量与常量指针的区别。

【正确答案】

常量指针是指向常量的指针，它所指向的地址的内容是不可修改的。

指针常量就是指针的常量，它是不可改变地址的指针，但是可以对它指向的内容进行修改。

【解释】

像两个连着的词，通常前面的词是修饰部分，后面的词则是中心词。

常量指针，即指针是常量形成，它首先应该是一个指针。

指针常量，常量是指针形式，它首先应该是一个常量。

常量指针，它是一个指向常量的指针。常量指针指向一个常量，是防止对指针误操作出现修改常量这样的错误。因此，常量指针是指向常量的指针，指针所指向的地址的内容是不可修改的。

指针常量，它首先是一个常量，然后才是一个指针。指针常量不能修改指针所指向的地址，一旦初始化，地址就固定了，不能对它进行移动操作。如果对指针常量进行自增操作，系统会提示出错。但是注意，指针常量的内容是可以替换的，这和常量指针是完全不同的概念。总之，指针常量是不可改变地址的指针，但是可以对它所指向的内容进行修改。

【题目编号】 CSD-03-302-4008

【知 识 点】 引用和指针 - 指针基础 - const关键字在指针声明时的作用

【难易程度】 60

【题目描述】

分析代码回答问题——指出下列几种指针的区别。

|  |  |
| --- | --- |
| 1  2  3  4 | char\* const p1;  char const \*p2;  const char \*p3;  const char\* const p4; |

【正确答案】

p1是指针常量，它本身不能被修改，但是指向的内容可以被修改。

p2和p3是常量指针，它们本身可以被修改，但是指向的内容不可以被修改。

p4是常量指针，并且指向的内容也不可以被修改。

【解释】

const位于\*号的左侧，则const是用来修饰指针所指向的变量，即指针指向常量；如果const位于\*号的右侧，const就是修饰指针本身，即指针本身是常量。因此，p1指针本身是常量，但它指向的内容可以被修改。p2和p3都是指针所指向的内容为常量。p4则表示它指针本身是常量，并且它指向的内容也不可被修改。

## 3.3 this指针

在C++中，对象的this指针并不是对象本身的一部分，不会影响sizeof（对象）的结果。this指针的作用域是类内部，在类的非静态成员函数中访问类的非静态成员时，编译器会自动将对象地址作为一个隐含参数传递给函数。也就是说，即使没有this指针，编译器在编译的时候也是加上this指针，它作为非静态成员函数的隐含形参，对各成员的访问均通过this进行。

【题目编号】 CSD-03-303-1001

【知 识 点】 引用和指针 - this指针 - this指针

【难易程度】 70

【题目描述】

this指针是C++实现( )的一种机制。

【选 项】

A、抽象

B、封装

C、继承

D、重载

【正确答案】 B

【解释】 无

【题目编号】 CSD-03-303-1002

【知 识 点】 引用和指针 - this指针 - this指针

【难易程度】 60

【题目描述】

系统为每个类提供一个this指针，在类的成员函数内，通过this指针可以间接访问这个类的（ ）。

【选 项】

A、在public段中声明的友元

B、所有成员

C、友元类的public成员

D、所有派生类中的public成员

【正确答案】 B

【解释】 无

【题目编号】 CSD-03-303-1003

【知 识 点】 引用和指针 - this指针 - this指针的基本概念

【难易程度】 60

【题目描述】

下列关于this指针的叙述中，正确的是：

【选 项】

A、任何与类相关的函数都有this指针。

B、类的成员函数都有this指针。

C、类的友元函数都有this指针。

D、类的非静态成员函数才有this指针。

【正确答案】 D

【解释】

A错误。类的非静态成员函数属于类的对象，含有this指针。而类的static函数属于类本身，不含this指针。

B错误。类的非静态成员函数属于类的对象，含有this指针。而类的static函数属于类本身，不含this指针。

C错误。友元函数是非成员函数，所以它无法通过this指针进行复制。

【题目编号】 CSD-03-303-4001

【知 识 点】 引用和指针 - this指针 - this指针的使用

【难易程度】 70

【题目描述】

下面的代码输出结果是什么？如果取消代码14行的注释，输出又是什么？

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | #include <iostream>  using namespace std;  class MyClass  {  public:  int data;  MyClass(int data)  {  this->data = data;  }  void print()  {  //cout << data << endl;  cout << "hello!" << endl;  }  };  int main()  {  MyClass \*pMyClass;  pMyClass = new MyClass(1);  pMyClass->print();  pMyClass[0].print();  pMyClass[1].print();  pMyClass[100000000].print();  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
|  | 注释代码第14行的输出结果 |
| 1  2  3  4 | hello!  hello!  hello!  hello! |
|  | 去掉代码14行的注释输出结果 |
| 1  2  3  4  5  6  7 | 1  hello!  1  hello!  0  hello!  段错误 (核心已转储) |

【解释】

对于类成员函数而言，并不是一个对象对应一个单独的成员函数体，而是类的所有对象共用一个成员函数体，程序被编译后，此成员函数地址即已确定。调用类成员函数时，会将当前对象的this指针传递给成员函数。一个类的成员函数体只有一个，而成员函数之所以能把属于此类的各个对象的数据区别开，是因为每次执行类成员函数时，都会把当前对象的this指针（也即对象首地址）传入成员函数，函数体内所有对类数据成员的访问，都会转化为“this->数据成员”的方式。

当print函数里没有任何访问对象的数据成员时，此时传进来对象的this指针实际上是没有任何用处的，这样的函数，其特征与全局函数没有太大区别。如果取消14行的注释，print函数要访问类的数据成员data，而程序只构造了一个Myclass，显然，下标“1“和下标“10000000”的MyClass对象根本不存在，那么对它们的数据成员访问也显然是非法的。

## 3.4 指针数组与数组指针

指针数组与数组指针是两个不同的概念，它们区别如下：指针的数组，即数组的元素是指针形成；数组的指针，即指向数组的指针。

【题目编号】 CSD-03-304-1001

【知 识 点】 引用和指针 - 指针数组与数组指针 - 对象数组指针

【难易程度】 80

【题目描述】

下列定义中，X是一个类, ( )是定义指向对象数组的指针p。

【选 项】

A、x\*p[4]

B、x(\*p)[4]

C、(x\*)p[4]

D、x\*p[]

【正确答案】 B

【解释】

数组指针的定义方法：

第一步：写一个该类型的数组

第二步：把数组名前面加上\*

第三步：把\*和数组名一起用小括号括起来

【题目编号】 CSD-03-304-4001

【知 识 点】 引用和指针 - 指针数组与数组指针 - 数组和链表的区别

【难易程度】 70

【题目描述】

数组和链表的区别？

【正确答案】

数组的数据是顺序存储的，存储大小固定。链表的数据可以随机存储，存储大小可以动态的改变。

【解释】 无

【题目编号】 CSD-03-304-4002

【知 识 点】 引用和指针 - 指针数组与数组指针 - 指针数组与数组指针的区别

【难易程度】 60

【题目描述】

指针数组与数组指针的区别是什么？

【正确答案】

指针数组表示一个数组，并且数组中的每一个元素都是指针类型。

数组指针表示一个指针，并且是指向数组的指针。

【解释】 无

## 3.5 函数指针与指针函数

函数指针与指针函数也是两个容易混淆的概念，尤其是当它们一起出现时候。因此，许多公司的笔试或者面试题中出现了这两个概念的辨析，以此考查应聘者是否有扎实的基础。大家只需要记住一点：函数指针是函数，指针函数是指针。

【题目编号】 CSD-03-305-4001

【知 识 点】 引用和指针 - 函数指针与指针函数 - 函数指针与指针函数的区别。

【难易程度】 70

【题目描述】

函数指针与指针函数的区别。

【正确答案】

指针函数是返回指针类型的函数。

函数指针是指向函数地址的指针。

【解释】 无

【题目编号】 CSD-03-305-4002

【知 识 点】 引用和指针 - 函数指针与指针函数 - 数组指针与函数指针的定义

【难易程度】 60

【题目描述】

定义下面的几种类型变量。

（1）含有十个元素的指针数组

（2）数组指针

（3）函数指针

（4）指向函数的指针数组

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  5 | int \*a[10];  int \*a = new int[10]; // 或int a[10];int (\*pa)[10]=&a;  void (\*fn)(int, int);  int (\*fnArray[10])(int, int); |

【解释】 无

【题目编号】 CSD-03-305-4003

【知 识 点】 引用和指针 - 函数指针与指针函数 - 函数指针的使用

【难易程度】 70

【题目描述】

析代码写结果。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | #include <stdio.h>  int add1(int a1, int b1);  int add2(int a2, int b2);  int main(int argc, char \*argv[])  {  int numa1 = 1, numb1 = 2;  int numa2 = 2, numb2 = 3;  int (\*op[2])(int a, int b);  op[0] = add1;  op[1] = add2;  printf("%d %d\n", op[0](numa1, numb1), op[1](numa2, numb2));  getchar();  return 0;  }  int add1(int a1, int b1)  {  return (a1 + b1);  }  int add2(int a2, int b2)  {  return (a2 + b2);  } |

【正确答案】 3 5

【解释】

代码第8行，定义了一个函数指针数组op，它含有两个指针元素，在代码第9行和代码第10行把这两个元素分别指向add1和add2个函数地址。最后在代码11行打印出使用函数指针调用add1和add2这两个函数返回的结果。

## 3.6 野指针

野指针是指程序员或操作者不能控制的指针。当程序里定义了一个指针而又没有给这个指针一个具体地址指向时，这个指针会随意地指向一个地址，这样的指针就是一个野指针。如果这个地址后面的内存空间没有什么重要的数据则不会造成不严重后果，但是一旦里面存放了有用的数据，那么这些数据随时都有被野指针存取的危险，如果这样，数据就会被破坏，程序也会崩溃。因此在程序里禁止野指针的存在。

【题目编号】 CSD-03-306-4001

【知 识 点】 引用和指针 - 野指针 - 野指针

【难易程度】 70

【题目描述】

什么是“野指针“？

【正确答案】

“野指针“不是NULL指针，而是指向“垃圾”内存的指针，其主要成因是指针变量没有被初始化，或指针被free或者delete之后，没有置为NULL。

【解释】 无

【题目编号】 CSD-03-306-4002

【知 识 点】 引用和指针 - 野指针 - malloc/free和new/delete的区别

【难易程度】 80

【题目描述】

有了malloc/free，为什么还要new/delete？

【正确答案】

对于非内部数据类型的对象而言，对象在消亡之前要自动执行析构函数。由于malloc/free函数是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于malloc/free函数，因此只能使用new/delete运算符。

【解释】 无

## 3.7 动态内存

C/C++编程时，会多次与内存管理打交道，因此掌握内存分配、传递、赋值与释放等知识十分必要。

【题目编号】 CSD-03-307-1001

【知 识 点】 引用和指针 - 动态内存 - 动态内存分配

【难易程度】 60

【题目描述】

关于动态分配的对象，下面的说法（ ）是正确的？

【选 项】

A、对于数组A[N]，如果是静态分配,N必须是常量；如果是动态分配，N可以是变量。

B、动态分配的对象必须由同类型的指针指出，不管该指针是局部的还是全局的，动态分配对象的生命周期随指针的生命期结束而结束。

C、静态和动态分配的对象在生命期结束时，都自动调用析构函数进行析构。

D、如果在某个函数内动态分配了一个对象，那么这个对象只有在这个函数内才能被访问。

【正确答案】 C

【解释】 无

【题目编号】 CSD-03-307-4001

【知 识 点】 引用和指针 - 动态内存 - 动态内存分配

【难易程度】 80

【题目描述】

几种内存分配方式以及它们的区别。

【正确答案】

（1）从静态存储区域分配。内存在程序编译的时候就已经分配好，这块内存在程序的整个运行期间都存在。例如全局变量，static 变量。

（2）在栈上创建。在执行函数时，函数内局部变量的存储单元都可以在栈上创建，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令。

（3）从堆上分配，亦称动态内存分配。程序在运行的时候用malloc 或new 申请任意多少的内存，程序员自己负责在何时用free 或delete 释放内存。动态内存的生存期由程序员决定，使用非常灵活，但问题也最多。

【解释】 无

【题目编号】 CSD-03-307-4002

【知 识 点】 引用和指针 - 动态内存 - 堆栈溢出

【难易程度】 60

【题目描述】

堆栈溢出一般是由什么原因导致的？

【正确答案】

没有回收垃圾资源；层次太深的递归调用。

【解释】 无

【题目编号】 CSD-03-307-4003

【知 识 点】 引用和指针 - 动态内存 - 动态内存的传递

【难易程度】 60

【题目描述】

程序分析。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | #include <iostream>  #include <cstdlib>  #include <cstring>  using namespace std;  void GetMemory(char \*p, int num)  {  p = (char\*)malloc(sizeof(char) \* num);  }  int main()  {  char \*str = NULL;  GetMemory(str, 10);  strcpy(str, "hello");  return 0;  } |

【正确答案】

调用“strcpy(str, "hello")”时程序崩溃。因为GetMemory()函数不能传递动态内存，并且str始终都是NULL。

【解释】 无

【题目编号】 CSD-03-307-4004

【知 识 点】 引用和指针 - 动态内存 - 动态内存的传递

【难易程度】 70

【题目描述】

比较分析两个代码段的输出。

|  |  |
| --- | --- |
|  | 程序1 |
| 1  2  3  4  5  6  7  8  9  10  11  12 | char\* GetMemory()  {  char p[] = "hello world";  return p;  }  void Test(void)  {  char \*str = NULL;  str = GetMemory();  printf(str);  } |
|  | 程序2 |
| 1  2  3  4  5  6  7  8  9  10  11  12 | void GetMemory(char \*p)  {  p = (char\*)(malloc(100);  }  void Test(void)  {  char \*str = NULL;  GetMemory(str);  strcpy(str, "hello world");  printf(str);  } |

【正确答案】

程序1输出结果可能是乱码。

程序2有内存泄漏，在Test函数调用strcpy()时程序崩溃。

【解释】

程序1的GetMemory()返回的是指向栈内存的指针，该指针的地址不是NULL，但是当栈退出后，内容不定，有可能会输出乱码。

程序2的GetMemory()没有返回值，这个函数不能传递动态内存。在Test函数中str变量的值通过参数传值的方式赋值给GetMemory()的局部变量p。但是Test()中的str一直为NULL，所以在代码第10行的调用会使程序崩溃。此外，由于在GetMemory()执行之后没有指针引用堆内存，因此会产生内存泄漏。

【题目编号】 CSD-03-307-4004

【知 识 点】 引用和指针 - 动态内存 - 静态存储区、栈、堆的内存分配

【难易程度】 80

【题目描述】

内存的分配方式有几种？

【正确答案】

从静态存储区域分配内存。程序编译的时候内存己经分配好了，并且在程序的整个运

行期间都存在，例如全局变量。

在栈上创建。在执行函数时，函数内局部变量的存储单元可以在栈上创建，函数结束时这些存储单元自动被释放。处理器的指定集中有关于栈内存的分配运算，因此效率很高，但是分配的内存容量有限。

从堆上分配内存，亦称动态内存分配。程序在运行的时候用malloc函数或new运算符申请任意大小的内存，程序员要用free函数或delete运算符释放内存。动态内存使用非常灵活，但问题也很多。

【解释】 无

## 3.8 指针与句柄的区别

很多人以为句柄就是指针，实际上句柄并不是那么简单的概念。应聘时会遇到有关句柄的知识，以考查应聘者是否对Windows编程有一定的了解。

【题目编号】 CSD-03-308-4001

【知 识 点】 引用和指针 - 指针与句柄的区别 - 对于Windows句柄的理解

【难易程度】 70

【题目描述】

什么是句柄？

【正确答案】

句柄在Windows编程中是一个很重要的概念，在很多程序中都扮演着重要的角色。在Windows环境中，句柄是用来标识项目的，这些项目包括：模块、任务、实例、文件、内存块、菜单控制、字体、资源（包括图标（icon)，光标（cursor)，字符串（string）等）和GDI对象（包括位图，画刷、元文件、调色板、画笔、区域以及设备描述表）。

Windows是以虚拟内存为基础的操作系统。在这种操作系统下，Windows内存管理器在内存中来回移动对象，从而满足各种应用程序的内存需要。对象被移动意味着它的地址变化了。由于地址总是如此变化，所以Windows操作系统为各应用程序腾出一些内存储地址，专门用来登记各应用对象在内存中的地址变化，而这些地址（存储单元的位置）本身是不变的。Windows内存管理器在内存中移动对象位置后，把对象新的地址告知句柄地址来保存。这样只需记住这个句柄地址就可以间接地知道对象在内存中具体的位置。这个地址在对象装载时由系统分配，当系统卸载时又释放给系统。

因此Windows程序中并不是用物理地址来标识内存块、文件、任务或动态装入模块的，相反，Windows API给这些项目分配确定的句柄，并将句柄返回给应用程序，然后通过句柄来进行操作。

程序执行顺序是：句柄地址（稳定）->记载对象在内存中的地址->对象在内存中的地址（不稳定）->实际对象。但是程序每次重新启动，系统分配给程序的句柄并不一定还是原来的句柄，而且绝大多数情况下是不一样的。

【解释】 无

【题目编号】 CSD-03-308-4002

【知 识 点】 引用和指针 - 指针与句柄的区别 - windows句柄和一般指针的区别

【难易程度】 70

【题目描述】

指针与句柄有什么区别？

【正确答案】

指针对应着数据在内存中的地址，利用指针可以自由地修改数据。Windows并不希望一般程序修改其内部数据结构，因为这样太不安全。所以Windows给每个使用GlobalAlloc等函数声明的内存区域指定一个句柄，即指向指针的指针。

句柄和指针都是地址，不同之处如下所示。

句柄所指可以是一个复杂的结构，并且可以与系统有关，例如线程的句柄，它可以指向一个类或者结构，而且和系统有很密切的关系，当一个线程由于不可预料的原因而终止时，系统就可以返回它所占用的资料，如CPU、内存等，反之、句柄中的某一些项是与系统进行交互的。

指针也可以指向一个复杂的结构，但是通常是由用户定义的，所以必要的工作要由用户完成，特别删除部分的工作。

【解释】 无

【题目编号】 CSD-03-301-4003

【知 识 点】 引用和指针 - 指针与句柄的区别 - Itearator和指针的区别

【难易程度】 80

【题目描述】

Itearator 和指针的区别。

【正确答案】

游标是指针，但不仅仅是指针。游标和指针很像，功能很像指针，但是实际上，游标是通过重载一元的”\*”和”->”来从容器中间接地返回一个值。将这些值存储在容器中并不是一个好主意，因为每当一个新值添加到容器中或者有一个值从容器中删除，这些值就会失效。在某种程度上，游标可以看作是句柄（handle）。通常情况下游标（iterator）的类型可以有所变化，这样容器也会有几种不同方式的转变。

iterator——对于除了 vector 以外的其他任何容器，你可以通过这种游标在一次操作中在容器中朝向前的方向走一步。这意味着对于这种游标你只能使用“++”操作符。而不能使用“--”或“+=”操作符。而对于 vector 这一种容器，你可以使用“+=”、“—”、“++”、“-=”中的任何一种操作符和“<”、“<=”、“>”、“>=”、“==”、“!=”等比较运算符。

【解释】 无

# 第4篇 字符串

在C/C++语言中没有专门的字符串变量，通常用字符数组来存放字符串。字符串是以“\0“作为结束符。C/C++提供了丰富的字符串处理函数，下面列出了几个最常用的函数。

字符串输出函数puts。

字符串输出函数gets。

字符串连接函数strcat。

字符串复制函数strcpy。

测字符串长度函数strlen。

字符串是面试的重点考查部分的相关知识，通过考查字符串的相关知识可以考察程序员的编程规范以及编程习惯。并且其中包括了许多知识点，例如内存越界、指针与数组操作等。许多公司在面试时会要求应聘者写一段复制字符串或字符串子串操作的程序。本章列举了一些与字符串相关的面试题，有些题目要求较高的编程技巧。

## 4.1 数字与字符串的转化

【题目编号】 CSD-04-401-1001

【知 识 点】 字符串 - 数字与字符串的转化 - 指针与字符串

【难易程度】 70

【题目描述】

下面程序的输出结果是( )。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | #include<iostream>  int fun(char \*s)  {  char \*p=s;  while(\*p!='\0')  p++;  return (p-s);  }  void main()  {  cout<<fun("ABCDEFGHIJKLMNIPQRST")<<endl;  } |

【选 项】

A、20 B、6 C、19 D、0

【正确答案】 A

【解释】

函数fun的作用时，传入一个带有’\0’的字符串，然后获取字符个数。

【题目编号】 CSD-04-401-4001

【知 识 点】 字符串 - 数字与字符串的转化 - C库函数中数字转换为字符串的使用

【难易程度】 60

【题目描述】

C语言提供了几个扩展函数，可以将任意类型（整型、长整型、浮点型等）的数字转换为字符串，下面列举了各函数的方法及其说明。

【正确答案】

itoa()：将整型值转换为字符串。

ltoa()：将长整型值转换为字符串。

ultoa()：将无符号长整型值转换为字符串。

gcvt()：将浮点型数转换为字符串，取四舍五入。

ecvt()：将双精度浮点型值转换为字符串，转换结果中不包含十进制小数点。

fcvt()：指定位数为转换精度，其余同ecvt()。

还可以使用sprintf系列函数把数字转换成字符串，其比itoa系列函数运行速度慢。

【解释】 无

【题目编号】 CSD-04-401-4002

【知 识 点】 字符串 - 数字与字符串的转化 - 数字转换为字符串，理解相关ASCII码

【难易程度】 70

【题目描述】

不使用库函数将整数转换为字符串。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | char \*IntToStr(int num, char str[])  {  int i = 0, j = 0;  char temp[100] = "";  while(num)  {  temp[i] = num % 10 + '0';  num = num / 10;  i++;  }  temp[i] = 0;    i = i - 1;  while(i >= 0)  {  str[j] = temp[i];  i--;  j++;  }  str[j] = 0;  return str;  } |

【解释】 无

【题目编号】 CSD-04-401-4003

【知 识 点】 字符串 - 数字与字符串的转化 - 库函数实现

【难易程度】 70

【题目描述】

C/C++语言提供了几个扩展函数，可以将字符串转换为任意类型（整型、长整型、浮点型等）。以下列举了各函数的方法及其说明。

atof()：将字符串转换为双精度浮点型值。

atoi()：将字符串转换为整型值。

atol()：将字符串转换为长整型值。

strtod()：将字符串转换为双精度浮点型值，并报告不能被转换的所有剩余数字。

strtol()：将字符串转换为长整值，并报告不能被转换的所有剩余数字。

strtoul()：将字符串转换为无符号长整型值，并报告不能被转换的所有剩余数字。

请实现atoi函数和atof函数。

【正确答案】

|  |  |
| --- | --- |
|  | 实现atoi函数 |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | int my\_atoi(char\* pstr)  {  int Ret\_Integer = 0;  int Integer\_sign = 1;    if(pstr == NULL)  {  printf("Pointer is NULL\n");  return 0;  }    while(isspace(\*pstr) == 0)  {  pstr++;  }    if(\*pstr == '-')  {  Integer\_sign = -1;  }  if(\*pstr == '-' || \*pstr == '+')  {  pstr++;  }    while(\*pstr >= '0' && \*pstr <= '9')  {  Ret\_Integer = Ret\_Integer \* 10 + \*pstr - '0';  pstr++;  }  Ret\_Integer = Integer\_sign \* Ret\_Integer;    return Ret\_Integer;  } |
|  | 实现atof函数 |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | double atof(char \*s)  {  int i,j,k=0;  float total,head = 0.0,tail = 0.0;  for(i = 0;i<strlen(s);i++)  {  k++;  if(\*(s+i)=='.')  break;  }  for(j=0;j<k-1;j++)  {  head = head\*10 + \*(s+j)-'0';  }  for(i=strlen(s)-1;i>=k;i--)  {  tail = tail/10.0 + (\*(s+i)-'0')/10.0;  }  total = head + tail;  printf("S after atof:%f",total);  } |

【解释】 无

## 4.2 字符串与数组

【题目编号】 CSD-04-402-4001

【知 识 点】 字符串 - 字符串与数组 - 字符串复制函数的实现

【难易程度】 70

【题目描述】

编程实现strcpy函数。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | char \*strcpy(char \*dest, const char \*src)  {  char \*tmp = dest;  while ((\*dest++ = \*src++) != '\0')  /\* nothing \*/;  return tmp;  } |

【解释】 无

【题目编号】 CSD-04-402-4002

【知 识 点】 字符串 - 字符串与数组 - 内存复制函数的实现

【难易程度】 70

【题目描述】

编程实现memcpy函数。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | void \*memcpy(void \*\_\_dest, \_\_const void \*\_\_src, size\_t \_\_n)  {  int i;  char \*d = (char \*)\_\_dest, \*s = (char \*)\_\_src;  for (i = 0; i < \_\_n; i++)  d[i] = s[i];  return \_\_dest;  } |

【解释】 无

【题目编号】 CSD-04-402-4003

【知 识 点】 字符串 - 字符串与数组 - 字符串复制与内存复制之间的区别

【难易程度】 60

【题目描述】

strcpy与memcpy的区别。

【正确答案】

strcpy和memcpy主要有以下3方面的区别。

复制的内容不同。strcpy只能复制字符串，而memcpy可以复制任意内容，例如字符数组、整型、结构体、类等。

复制的方法不同。strcpy不需要指定长度，它遇到字符串结束符“\0”便结束。memcpy

则是根据其第3个参数决定复制的长度。

用途不同。通常在复制字符串时用strcpy，而需要复制其他类型数据时则一般用memcpy()。

【解释】 无

【题目编号】 CSD-04-402-4004

【知 识 点】 字符串 - 字符串与数组 - 数组越界出现的问题

【难易程度】 70

【题目描述】

改错——数组越界。

|  |  |
| --- | --- |
|  | 题1 |
| 1  2  3  4  5  6 | void test1()  {  char string[10];  char \*str1 = "0123456789";  strcpy(string, str1);  } |
|  | 题2 |
| 1  2  3  4  5  6  7  8  9  10 | void test2()  {  char string[10], str1[10];  int i;  for(i = 0; i < 10; i++)  {  str1[i] = 'a';  }  strcpy(string, str1);  } |
|  | 题3 |
| 1  2  3  4  5  6  7  8 | void test3(char \*str1)  {  char string[10];  if(strlen(str1) <= 10)  {  strcpy(string, str1);  }  } |

【正确答案】

|  |  |
| --- | --- |
|  | 题1 |
| 1  2  3  4  5  6 | void test1()  {  **char string[11];**  char \*str1 = "0123456789";  strcpy(string, str1);  } |
|  | 题2 |
| 1  2  3  4  5  6  7  8  9  10  11 | void test2()  {  **char string[11], str1[11];**  int i;  for(i = 0; i < 10; i++)  {  str1[i] = 'a';  }  **str1[10] = '\0';**  strcpy(string, str1);  } |
|  | 题3 |
| 1  2  3  4  5  6  7  8 | void test3(char \*str1)  {  char string[10];  **if(strlen(str1) < 10)**  {  strcpy(string, str1);  }  } |

【解释】

题1中，string是一个含有10个元素的字符数组，str1指向的字符串长度为10，在进行strcpy调用时，会将str1的结束符也复制到string数组里，即复制的字符个数为11，这样会导致string出现数组越界。程序不一定会因此而崩溃，但这是一个潜在的危险。解决办法是将string的元素个数定义为11。

题2中，str1和string都是含有10个元素的字符数组，并且str1的元素全部被赋为字符“a”，然后再调用strcpy。这里会出现以下两个问题：一个问题是str1表示的字符数组没有“\0”结束，在随后调用strcpy时无法判断什么时候复制结束；另一个问题是string的数组长度不够，出现数组越界的现象。解决办法是将string和str1的元素个数都定义为11个，并在调用strcpy之前加入一条语句把str1[10]赋为“\0“。

题3中，if语句使用小于等于进行比较，如果str1的长度等于10，也会出现数组越界的情况。解决办法是把小于等于换成小于。

【题目编号】 CSD-04-402-4005

【知 识 点】 字符串 - 字符串与数组 - 打印操作时可能产生的数组越界问题。

【难易程度】 60

【题目描述】

分析程序——打印操作可能产生数组越界。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | #include <stdio.h>  int main()  {  int a[5] = {0, 1, 2, 3, 4}, \*p;  p = a;  printf("%d\n", \*(p + 4 \* sizeof(int)));    return 0;  } |

【正确答案】

这个程序存在着越界的问题。代码第6行，p指向a的第1个元素，所以p+4指向a的最后一个元素即4,p + 4 \* sizeof(int)即p + 16，此时指向的是数组a的第17个元素，显然已经越界了，因此打印结果是个随机数。

【解释】 无

【题目编号】 CSD-04-402-4006

【知 识 点】 字符串 - 字符串与数组 - strlen库函数的实现细节

【难易程度】 60

【题目描述】

编程实现字符串的长度检测。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | size\_t strlen(const char \*s)  {  const char \*sc = s;  while (\*sc != '\0')  sc++;  return sc - s;  } |

【解释】 无

【题目编号】 CSD-04-402-4007

【知 识 点】 字符串 - 字符串与数组 - strstr库函数的实现细节

【难易程度】 70

【题目描述】

编程实现字符串中子串的查找。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | char \*strstr(const char \*s1, const char \*s2)  {  size\_t l1, l2;  l2 = strlen(s2);  if (!l2)  return (char \*)s1;  l1 = strlen(s1);  while (l1 >= l2) {  l1--;  if (!memcmp(s1, s2, l2))  return (char \*)s1;  s1++;  }  return NULL;  } |

【解释】 无

【题目编号】 CSD-04-402-4008

【知 识 点】 字符串 - 字符串与数组 - 库函数strcmp的实现细节

【难易程度】 70

【题目描述】

编程实现strcmp库函数。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | int strcmp(const char \*cs, const char \*ct)  {  unsigned char c1, c2;  while (1) {  c1 = \*cs++;  c2 = \*ct++;  if (c1 != c2)  return c1 < c2 ? -1 : 1;  if (!c1)  break;  }  return 0;  } |

【解释】 无

【题目编号】 CSD-04-402-4009

【知 识 点】 字符串 - 字符串与数组 - 字符串相关的综合编程能力

【难易程度】 80

【题目描述】

编写字符串反转函数。

【正确答案】

|  |  |
| --- | --- |
|  | 方法1： |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | char\* strrev1(const char \*str)  {  int len = strlen(str);  char \*tmp = new char[len + 1];    strcpy(tmp, str);  for(int i = 0; i < len / 2; ++i)  {  char c = tmp[i];  tmp[i] = tmp[len - i - 1];  tmp[len - i - 1] = c;  }    return tmp;  } |
|  | 方法2： |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | char\* strrev2(const char \*str)  {  char \*tmp = new char[strlen(str) + 1];  strcpy(tmp, str);  char \*ret = tmp;  char \*p = tmp + strlen(str) - 1;    while(p > tmp)  {  char t = \*tmp;  \*tmp = \*p;  \*p = t;  --p;  ++tmp;  }    return ret;  } |
|  | 方法3： |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | char\*strrev3(const char \*str)  {  char \*tmp = new char[strlen(str) + 1];  strcpy(tmp, str);  char \*ret = tmp;  char \*p = tmp + strlen(str) - 1;    while(p > tmp)  {  \*p ^= \*tmp;  \*tmp ^= \*p;  \*p ^= \*tmp;    --p;  ++tmp;  }    return ret;  } |
|  | 方法4： |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | char\* strrev4(const char \*str)  {  char \*tmp = new char[strlen(str) + 1];  strcpy(tmp, str);  char \*ret = tmp;    char \*p = tmp + strlen(str) - 1;    while(p > tmp)  {  \*p = \*p + \*tmp;  \*tmp = \*p - \*tmp;  \*p = \*p - \*tmp;    --p;  ++tmp;  }    return ret;  } |
|  | 方法5 |
| 1  2  3  4  5  6  7  8  9  10  11 | char\* strrev5(char \*str, int len)  {  if(len <= 1)  return str;    char t = \*str;  \*str = \*(str + len - 1);  \*(str + len - 1) = t;    return (strrev5(str + 1, len - 2) - 1);  } |

【解释】 无

【题目编号】 CSD-04-402-4010

【知 识 点】 字符串 - 字符串与数组 - 字符串综合编程能力

【难易程度】 80

【题目描述】

编程实现任意长度的两个正整数相加。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | char \*addBigInt(char \*num1, char \*num2)  {  int c = 0; /\* 开始最低进位为0 \*/  int i = strlen(num1) - 1; /\* 指向第一个加数的最低位 \*/  int j = strlen(num2) - 1; /\* 指向第二个加数的最低位 \*/  int k;/\* 指向结果的最低位 \*/    debug("i = %d, j = %d\n", i, j);  debug("num1 = %s, num2 = %s\n", num1, num2);    /\* 得到两个数中较大数的位数,并加1位作为预留进位空间 \*/  int maxLength = strlen(num1) >= strlen(num2) ? strlen(num1) + 1 :  strlen(num2) + 1;  debug("maxLength = %d\n", maxLength);    char \*rst = (char \*)malloc(sizeof(char) \* (maxLength + 1));  if (rst == NULL)  {  printf("malloc error\n");  exit(1);  }    rst[maxLength] = '\0';  k = maxLength - 1; /\* 指向结果的最低位 \*/  debug("k = %d\n", k);    while ((i >= 0) && (j >= 0))  {  /\* 进算本位的值 \*/  rst[k] = ((num1[i] - '0') + (num2[j] - '0') + c)%10 + '0';  /\* 向高位进位的值 \*/  c = ((num1[i] - '0') + (num2[j] - '0') + c) / 10;  --i;  --j;  --k;  }  while (i >= 0)  {  rst[k] = ((num1[i] - '0') + c)%10 + '0';  c = ((num1[i] - '0') + c) / 10;  --i;  --k;  }  while (j >= 0)  {  rst[k] = ((num2[j] - '0') + c)%10 + '0';  c = ((num2[j] - '0') + c) / 10;  --j;  --k;  }    rst[0] = c + '0';    return rst;  } |

【解释】

在C/C++中可以用int、float、double等类型表示数字，但是它们的长度都是有限的，而本题要求数字是任意长度，因而需要用字符串表示数字，同样用字符串表示结果。因此所要做的就是做一个类似整数加法的字符串转换。

【题目编号】 CSD-04-402-4011

【知 识 点】 字符串 - 字符串与数组 - 字符串综合编程能力

【难易程度】 80

【题目描述】

编程实现字符串循环右移n位。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | void loopMove(char \*pStr, int steps)  {  int n = strlen(pStr) - steps;  char tmp[strlen(pStr)];  strcpy(tmp, pStr + n);  strcpy(tmp + steps, pStr);  \*(tmp + strlen (pStr)) = '\0';  strcpy(pStr, tmp);  } |

【解释】

根据题意，编写的函数能把字符串循环右移n位。例如字符串“abcdefghi"，如果n=2，移位后是“hiabcdefgh”。

【题目编号】 CSD-04-402-4012

【知 识 点】 字符串 - 字符串与数组 - 字符串综合编程能力

【难易程度】 80

【题目描述】

编程实现删除字符串中所有指定的字符。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | char\* deleteChar(char \*str, char c)  {  char \*head = NULL;  char \*p = NULL;    if(str = NULL)  {  return NULL;  }    head = p = str;    while(\*p++)  {  if(\*p != c)  {  \*str++ = \*p;  }  }  \*str = '\0';    return head;  } |

【解释】

根据题意，假设字符串为“cabodefegchci”，把该字符串中所有的字符“c”删除掉后，那么结果应该是“abodefeghi”。

【题目编号】 CSD-04-402-4013

【知 识 点】 字符串 - 字符串与数组 - 字符相关库函数实现

【难易程度】 70

【题目描述】

编程实现库函数strcat。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | char \*strcat(char \*dest, const char \*src)  {  char \*tmp = dest;  while (\*dest)  dest++;  while ((\*dest++ = \*src++) != '\0')  ;  return tmp;  } |

【解释】

库函数strcat把字符串内容连接到目标字符串的后面，所以应该从目标字符串的末尾，也就是结束符“\0“的位置开始插入另一个字符串的内容。

# 第5篇 位运算

## 5.1 位制转换与位运算

【题目编号】 CSD-05-501-4001

【知 识 点】 位运算 - 位制转换与位运算 - 使用prinif输出不同类型变量。

【难易程度】 60

【题目描述】

分析代码写出结果——位制转换。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #include <stdio.h>  int main(int argc, char \*argv[])  {  int i = 5.01;  float f = 5;  printf("%f\n", 5);  printf("%lf\n", 5.01);  printf("%f\n", f);  printf("%d\n", 5.01);  printf("%d\n", i);  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5 | 0.000000  5.010000  5.000000  1889785610  5 |

【解释】 无

【题目编号】 CSD-05-501-4002

【知 识 点】 位运算 - 位制转换与位运算 - 使用位操作符>>或<<

【难易程度】 60

【题目描述】

分析代码写出结果——位运算。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | #include <stdio.h>  int main(int argc, char \*argv[])  {  unsigned short int i = 0;  unsigned char ii = 255;  int j = 8, p, q;    p = j << 1;  q = j >> 1;  i = i - 1;  ii = ii + 1;    printf("i = %d\n", i);  printf("ii = %d\n", ii);  printf("p = %d\n", p);  printf("q = %d\n", q);    return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4 | i = 65535  ii = 0  p = 16  q = 4 |

【解释】 无

【题目编号】 CSD-05-501-4003

【知 识 点】 位运算 - 位制转换与位运算 - 使用&和|位操作符

【难易程度】 70

【题目描述】

嵌入式系统要求用户对变量或寄存器进行位操作。给定一个核型变量a，写两段代码，第1个设置a的BIT3，第2个清除a的BIT3。在以上两个操作中，要保持其他位不变。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | #define BIT3 (0x1 << 3)  static int a;  void set\_bit3(void)  {  a |= BIT3;  }  void clear\_bit3(void)  {  a &= ~BIT3;  } |

【解释】

【题目编号】 CSD-05-501-4004

【知 识 点】 位运算 - 位制转换与位运算 - 运算符转化

【难易程度】 70

【题目描述】

列举并解释C++中的四种运算符转化，说明它们的不同点。

【正确答案】

（1）const\_cast操作符：即供程序设计师在特殊情况下将限制为const成员函数的const定义解除，使其可以更改特定属性。

（2）dynamic\_cast操作符：如果启动了支持运行时间类型信息（RTTI)，dynamic\_cast有助于判断在运行时所指向对象的确定类型。它与typeid运算符有关，可以将一个基类的指针指向不同的子类型（派生类），然后将被转型作为基础类的对象还原成原来的类。不过，仅限于对象指针的类型转换。

（3）reinterpret\_cast操作符：将一个指针转换成其他类型的指针，新类型的指针与旧指针可以毫不相干。通常用于非标准的指针数据类型转换，例如将void＊转换为char＊。它也可以用于指针和整形数之间的类型转换。注意：它存在潜在的危险，除非有使用它的充分理由，否则就不要使用它。例如，将一个ini＊类型的指针转换为float＊类型的指针，会很容易造成整数数据不能被正确地读取。

（4）static\_cast操作符：在相关的对象和指针类型之间进行类型转换。有关的类之间必须通过继承构造函数或者转换函数进行联系。static\_cast操作符还能在数字（原始的）类型之间进行类型转换．通常情况下，static\_cast操作符用在将数域较大的类型转换为较小的类型。当转换的类型是原始数据类型时，这种操作可以有效地禁止编译器发出警告。

【解释】 无

## 5.2 嵌入式编程

【题目编号】 CSD-05-502-4001

【知 识 点】 位运算 - 嵌入式编程 - #define的使用规范

【难易程度】 60

【题目描述】

用#define声明一个常数，用以表明一年中有多少秒。

【正确答案】

#define SECONDS\_PER\_YEAR (60 \* 60 \* 24 \* 365)UL

【解释】

这里想考查以下几个方面。

#define语法的基本知识（例如不能以分号结束，括号的使用规则等）。

会用预处理器计算常数表达式的值，因此，直接写出计算一年中有多少秒而不是计算出实际的值，是清晰而没有代价的。

理解这个表达式会使16位机的整型数溢出，因此需要用长整型符号L告诉编译器这个

常数是长整型数。

如果在表达式中用到UL（表示无符号长整型），那么可以有一个好的起点。

【题目编号】 CSD-05-502-4002

【知 识 点】 位运算 - 嵌入式编程 - 合理编写代码访问特定内存

【难易程度】 70

【题目描述】

嵌入式系统经常要求程序员去访问特定的内存位置。在某工程中，一个整形变量的绝对地址0x67a9，请将其值设为0xaa55，并且己知编译器是一个纯粹的ANSI编译器，请编写程序代码。

【正确答案】

|  |  |
| --- | --- |
|  | 方案1 |
| 1  2  3 | int \*ptr;  ptr = (int\*)0x67a9;  \*ptr = 0xaa55; |
|  | 方案2 |
| 1 | \*(int \* const)(0x67a9) = 0xaa55; |

【解释】 无

【题目编号】 CSD-05-502-4003

【知 识 点】 位运算 - 嵌入式编程 - 对C语言中整数自动转换原则的理解

【难易程度】 70

【题目描述】

分析代码写结果―整数的自动转换。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | void foo(void)  {  unsigned int a = 6;  int b = -20;  if(a + b > 6)  puts(">6");  else  puts("<=6");  } |

【正确答案】 >6

【解释】

这个问题重点测试应聘者是否懂得C语言中的整数自动转换原则。这无符号整型问题的答案输出是“>6”。原因是当表达式中存在有符号类型和无符号类型时所有的操作数都自动转换为无符号类型。因此“-20”变成了一个非常大的正整数，所以该表达式计算出的结果是“>6”。这一点对于频繁用到无符号数据类型的嵌入式系统来说是非常重要的。

【题目编号】 CSD-05-502-4004

【知 识 点】 位运算 - 嵌入式编程 - 对C语言中static关键字的理解

【难易程度】 60

【题目描述】

关键字static的作用是什么？

【正确答案】

在C语言中，关键字static有3个明显的作用。

在函数体，被声明为静态的变量在这一函数被调用的过程中维持其值不变。

在模块内（但在函数体外），被声明为静态的变量可以被模块内所有函数访问，但不能被模块外其他函数访问。它是一个本地的全局变量。

在模块内，一个被声明为静态的函数只可被这一模块内的其他函数调用。即这个函数被限制在声明它的本地范围内。

大多数应聘者能正确回答第1部分，另一部分应聘者能正确回答第2部分，同时很少的人能懂得第3部分。这是应聘者的一个严重的知识缺陷，因为显然应聘者不懂得本地化数据和代码范围的好处及重要性。

【解释】 无

【题目编号】 CSD-05-502-4005

【知 识 点】 位运算 - 嵌入式编程 - 对C语言中volatile关键字作用的理解

【难易程度】 70

【题目描述】

关键字volatile有什么含义？

【正确答案】

定义为volatile的变量可能会被意想不到地改变，这样，编译器就不会去假设这个变量的值。准确地说，优化器在用到volatile变最时必须小心地重新读取该变量的值，而不是使用保存在寄存器里的备份。下面是使用volatile变量的3个例子。

并行设备的硬件寄存器（如：状态寄存器）。

一个中断服务子程序中会访问到的非自动变量。

多线程应用中被几个任务共享的变量。

【解释】 无

【题目编号】 CSD-05-502-4006

【知 识 点】 位运算 - 嵌入式编程 - 处理器的字节序以及union的作用

【难易程度】 80

【题目描述】

编写一个函数，若处理器使用大端模式存储数据，则返回0；若是用小端模式存储数据，则返回1。

【正确答案】

嵌入式系统开发者应该对小端和大端模式非常了解。采用小端模式的CPU对操作数的存储方式是从低字节到高字节，而大端模式对操作数的存储方式是从高字节到低字节。例如：

|  |  |
| --- | --- |
|  | 32位宽的数0x12345678在小端CPU内存中的存放方式（假设地址从0x4000开始） |
| 1  2  3  4 | 0x4000: 0x78  0x4001: 0x56  0x4002: 0x34  0x4003: 0x12 |
|  | 32位宽的数0x12345678在大端CPU内存中的存放方式（假设地址从0x4000开始） |
| 1  2  3  4 | 0x4000: 0x12  0x4001: 0x34  0x4002: 0x56  0x4003: 0x78 |

【解释】 无

【题目编号】 CSD-05-502-4007

【知 识 点】 位运算 - 嵌入式编程 - 中断

【难易程度】 70

【题目描述】

什么是中断？中断发生时CPU做什么工作？

【正确答案】

中断是CPU响应外设需求的一种模式，在外设需要CPU时，会向中断控制器发送中断请求，这时CPU要保护现场，即把正在运行的程序保存起来，一般是把状态压入堆栈，然后读中断号，启动相应的中断服务程序，服务完成后，载入保护现场，即把堆栈的数据弹出，继续运行之前的程序。

中断：是指当主机接到外界硬件(如I/O设备)发来的信号时，马上停止原来的工作，转去处理这一事件，在处理完了以后，主机又回到原来的工作继续工作。中断是机器BIOS中最重要的概念，当系统外部设备有请求时，比如鼠标，串口数据同步请求等，都会产生中断，这在一个单CPU的环境下，系统会保存当前机器状态，响应这个请求，在请求完成后，就恢复设备状态；“NGNc”就是一个中断请求，将我这个单CPU大脑中断，待响应结束IRET后，“NGNc”就呈现在了我的面前。

【解释】 无

【题目编号】 CSD-05-502-4008

【知 识 点】 位运算 - 嵌入式编程 - 系统启动顺序

【难易程度】 60

【题目描述】

cpu在上电后，进入操作系统的main()之前必须做什么工作？

【正确答案】

过程如下：

bios自举：检查硬件等

读取MBR

转到MBR执行它的代码,它会检测活动分区

把活动分区的引导扇区的引导代码装入内存

运行引导代码

引导代码装入该分区的操作系统，也就是进入main()（当然不一定叫main，如linux下叫start\_kernel）执行一系列的初始化，然后最终启动登录界面，实现启动过程。

【解释】 无

【题目编号】 CSD-05-502-4009

【知 识 点】 位运算 - 嵌入式编程 - 中断

【难易程度】 80

【题目描述】

中断是嵌入式系统中重要的组成部分，这导致了很多编译开发商提供一种扩展—让标准C支持中断。具代表事实是，产生了一个新的关键字 \_\_interrupt。下面的代码就使用了\_\_interrupt关键字去定义了一个中 断服务子程序(ISR)，请评论一下这段代码的。

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | \_\_interrupt double compute\_area (double radius)  {  double area = PI \* radius \* radius;  printf("\nArea = %f", area);  return area;  } |

【正确答案】

这个函数有太多的错误了，以至让人不知从何说起了：

（1）ISR 不能返回一个值。如果你不懂这个，那么你不会被雇用的。

（2）ISR 不能传递参数。如果你没有看到这一点，你被雇用的机会等同第一项。

（3）在许多的处理器/编译器中，浮点一般都是不可重入的。有些处理器/编译器需要让额处的寄存器入栈，有些处理器/编译器就是不允许在ISR中做浮点运算。此外，ISR应该是短而有效率的，在ISR中做浮点运算是不明智的。

（4）与第三点一脉相承，printf()经常有重入和性能上的问题。如果你丢掉了第三和第四点，我不会太为难你 的。不用说，如果你能得到后两点，那么你的被雇用前景越来越光明了。

【解释】 无

【题目编号】 CSD-05-502-4010

【知 识 点】 位运算 - 嵌入式编程 - 死循环

【难易程度】 60

【题目描述】

嵌入式系统中经常要用到无限循环，你怎么样用C编写死循环呢？

【正确答案】

这个问题用几个解决方案。我首选的方案是：

while(1){ }

一些程序员更喜欢如下方案：

for(;;){ }

这个实现方式让我为难，因为这个语法没有确切表达到底怎么回事。如果一个应试者给出这个作为方案，我将用这个作为一个机会去探究他们这样做的基本原理。如果他们的基本答案是：―我被教着这样做，但从没有想到过为什么。‖这会给我留下一个坏印象。

第三个方案是用 goto

Loop:

...

goto Loop;

应试者如给出上面的方案，这说明或者他是一个汇编语言程序员（这也许是好事）或者他是一个想进入新领域的BASIC/FORTRAN程序员。

【解释】 无

# 第6篇 C++面向对象

C语言是面向过程的，而C++作为C语言的超集支持，它是面向对象的编程。面向对象是当前计算机界关心的重点，它是当今软件开发方法的主流。因此也是笔试时的重要考点。在面试过程中，应聘者应该对面向对象的基本概念、类、对象构造函数以及多态性等有清晰的认识，具备面向对象编程的能力。

## 6.1 new/delete

【题目编号】 CSD-06-601-1001

【知 识 点】 C++面向对象 - new/delete - delete运算符

【难易程度】 60

【题目描述】

关于delete 运算符的下列描述中，（ ）是错误的。

【选 项】

A、它必须用语new返回的指针

B、它也适用于空指针

C、对一个指针可以使用多次该运算符

D、指针名前只用一队方括号符，不管所删除数组的维数

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-601-1002

【知 识 点】 C++面向对象 - new/delete - delete运算符

【难易程度】 50

【题目描述】

运算符delete删除一个动态对象时( )。

【选 项】

A、首先为该动态对象调用构造函数,在释放其占用的内存

B、首先释放该动态对象占用的内存,在为其调用构造函数

C、首先为该动态对象调用析构函数,在释放其占用的内存

D、首先释放动态对象占用的内存,在为其调用析构函数

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-601-1003

【知 识 点】 C++面向对象 - new/delete - new运算符

【难易程度】 60

【题目描述】

关于new运算符的下列描述中,( )是错误的。

【选 项】

A、它可以用来动态创建对象和对象数组

B、使用它创建对象或对象数组,可以使用运算符DELETE删除

C、使用它创建对象时要调用构造函数

D、使用它调用对象数组时不充许指定初始值

【正确答案】 D

【解释】 无

【题目编号】 CSD-06-601-4001

【知 识 点】 C++面向对象 - new/delete - new/delete和malloc/free的区别与联系

【难易程度】 70

【题目描述】

new/delete和malloc/free的区别与联系？

【正确答案】

面试中可以这样回答：

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 会调用对象的析构函数，而free 不会调用对象的析构函数。

详细讲解：

（1）malloc与free是C++/C语言的标准库函数，new/delete是C++的运算符。它们都用于在堆(heap)上进行动态的内存操作（申请动态内存和释放内存）。

（2）delete不仅会释放空间,在释放前会调用析构函数,和new对应，new调用构造函数，free只会释放内存。

（3）对于非内部数据类型的对象而言，光用malloc/free无法满足动态对象的要求。对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数。由于malloc/free是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于malloc/free。因此C++语言需要一个能完成动态内存分配和初始化工作的运算符new，以及一个能完成清理与释放内存工作的运算符delete。注意new/delete不是库函数。

（4）C++程序经常要调用C函数，其实new/delete内部实现也调用了malloc/free。

而C程序只能用malloc/free管理动态内存

（5）new是强制类型的，不需要考虑类型，而malloc不是，它返回的指针是void\*型，必须要强转成需要的类型。（注意）

（6）new 可以调用构造函数在声明的时候初始化，malloc只是分配空间,需要在其他地方初始化。而且malloc需要指定分配空间大小, 而new是自动计算的。

（7）new/delete是可以重载的，而重载之后，就成为了函数。当new/delete在类中被重载的时候，可以自定义申请过程，比如记录所申请内存的总长度，以及跟踪每个对象的指针。

（8）free和delete可以释放NULL指针。

【解释】 无

## 6.2 访控属性

【题目编号】 CSD-06-602-1001

【知 识 点】 C++面向对象 - new/delete - 类的访控权限

【难易程度】 50

【题目描述】

类定义的内容允许被其他对象无限制地存取是( )。

【选 项】

A、private 部分

B、protected 部分

C、public 部分

D、以上都不对

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-602-1002

【知 识 点】 C++面向对象 - new/delete - 类的访控权限

【难易程度】 60

【题目描述】

下列关于成员访问权限的描述中，不正确的是（ ）。

【选 项】

A、公有数据成员和公有成员函数都可以被类对象直接处理

B、类的私有数据成员只能被公有成员函数以及该类的任何友元类或友元函数访问

C、保护成员在派生类中可以被访问，而私有成员不可以

D、只有类或派生类的成员函数和友元类或友元函数可以访问保护成员

【正确答案】 B

【解释】 无

【题目编号】 CSD-06-602-1003

【知 识 点】 C++面向对象 - new/delete - 类的成员变量初始化

【难易程度】 60

【题目描述】

下面程序段中，说法正确的是（ ）。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | class location{  public :  int GetX();  };  class rectangle:private location {  public:  int GetW();  }; |

【选 项】

A、类rectangle私有派生类 location

B、类rectangle中的public成员在类 location中是私有的

C、类location中的public成员在类 rectangle中是不可访问的

D、viod f( ) { rectangle r; int X=r .GetX( ); } 其中int X = r .GetX( );是非法语句。

【正确答案】 D

【解释】

类rectangle私有派生类 location错误，是类location私有派生类rectangle类rectangle中的public成员在类 location中是私有的错误，类location内部没有类rectangle中的成员。类location中的public成员在类rectangle中是不可访问的错误，是不可以在类外部公开访问的。

【题目编号】 CSD-06-602-1004

【知 识 点】 C++面向对象 - new/delete - 类的访控权限

【难易程度】 50

【题目描述】

若派类的成员函数不能直接访问基类中继承来的某个成员，则该成员一定是基类中的（ ）。

【选 项】

A、私有成员

B、公有成员

C、保护成员

D、保护成员或私有成员

【正确答案】 A

【解释】 无

【题目编号】 CSD-06-602-1005

【知 识 点】 C++面向对象 - new/delete - 类的访问控制

【难易程度】 50

【题目描述】

对于公有派生，它的“访问控制“是（ ）。

【选 项】

A、private

B、public

C、protected

D、inaccessible

【正确答案】 B

【解释】 无

【题目编号】 CSD-06-602-1006

【知 识 点】 C++面向对象 - new/delete - 类的访控权限

【难易程度】 60

【题目描述】

下面叙述错误的是（ ）。

【选 项】

A、派生类可以使用private派生

B、对基类成员的访问必须是无二义性的

C、基类成员的访问能力在派生类中维持不变

D、赋值兼容规则也适用于多继承的组合

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-602-1007

【知 识 点】 C++面向对象 - new/delete - 类的访控权限

【难易程度】 60

【题目描述】

关于保护成员的说法正确的是（ ）。

【选 项】

A．在派生类中仍然是保护的

B．具有私有成员和公有成员的双重角色

C．在派生类中是私有的

D．在派生类中是公有的

【正确答案】 B

【解释】 无

【题目编号】 CSD-06-602-1008

【知 识 点】 C++面向对象 - new/delete - 类的访控权限

【难易程度】 60

【题目描述】

下面叙述不正确的是（ ）。

【选 项】

A．基类的保护成员在派生类中仍然是保护的

B．基类的保护成员在公有派生类中仍然是保护的

C．基类的保护成员在私有派生类中仍然是私有的

D．对基类成员的访问必须是无两义性

【正确答案】 A

【解释】 无

【题目编号】 CSD-06-602-1009

【知 识 点】 C++面向对象 - new/delete - 类的访控权限

【难易程度】 50

【题目描述】

下面不属于类的访控权限的是( )。

【选 项】

A、public

B、static

C、protected

D、private

【正确答案】 B

【解释】 无

## 6.3 模板和容器

【题目编号】 CSD-06-603-1001

【知 识 点】 C++面向对象 - 模板和窗口 - 模板

【难易程度】 60

【题目描述】

类模板的使用实际上是类模板实例化成一个具体的( )。

【选 项】

A、类

B、对象

C、函数

D、模板类

【正确答案】 A

【解释】 无

【题目编号】 CSD-06-603-1002

【知 识 点】 C++面向对象 - 模板和窗口 - 模板

【难易程度】 60

【题目描述】

类模板的模板参数( )。

【选 项】

A、只可作为数据成员的类型

B、只可作为成员函数的返回类型

C、只可作为成员函数的参数类型

D、以上三者皆可

【正确答案】 D

【解释】 无

【题目编号】 CSD-06-603-1003

【知 识 点】 C++面向对象 - 模板和窗口 - 模板

【难易程度】 70

【题目描述】

类模板template<class T>class X{…..}; 其中友元函数f对特定类型T（如 int），使函数f(X<int>&)成为X<int>模板类的友元，则其说明应为（）。

【选 项】

A、friend void f();

B、friend void f(X<T>&);

C、friend void A::f();

D、friend void C<T>::f(X<T>&);

【正确答案】 B

【解释】 无

【题目编号】 CSD-06-603-1004

【知 识 点】 C++面向对象 - 模板和窗口 - 模板

【难易程度】 60

【题目描述】

下面对模板的声明，正确的是（ ）。

【选 项】

A、template<T>

B、template<class T1,T2>

C、template<class T1,class T2>

D、template<class T1;class T2>

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-603-1005

【知 识 点】 C++面向对象 - 模板和窗口 - 模板

【难易程度】 60

【题目描述】

下面类模板的定义中，不正确的是（ ）

【选 项】

A、template<class T>

class A{T x,y;};

B、template<class T>

class A{private:T x;};

C、template<class T>

class A{public:T x;};

D、template<class T>

class A{protected:T x;};

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-603-1006

【知 识 点】 C++面向对象 - 模板和窗口 - 模板

【难易程度】 60

【题目描述】

对于类模板tanytemp,执行语句tanytemp<int>iobject(23,32);后( )。

【选 项】

A、产生的模板类为 tanytemp

B、产生的模板类为 iobject

C、产生的模板类为 tanytemp<int>

D、类模板实例化对象

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-603-4001

【知 识 点】 C++面向对象 - 模板和窗口 - 模板

【难易程度】 70

【题目描述】

请解释下面代码采用了何种C++特性（C语言不具备），作用是什么？

|  |  |
| --- | --- |
| 1  2  3  4  5 | template<typename T>  T sum(T a, T b)  {  return (a + b);  } |

【正确答案】

表示函数模板，用于将一类功能相同，参数类型和返回值不同的函数抽象为一个模板，方便模板函数调用。

【解释】 无

【题目编号】 CSD-06-603-4001

【知 识 点】 C++面向对象 - 模板和窗口 - 容器

【难易程度】 70

【题目描述】

C++容器的分类及各自的特性？

【正确答案】

C++容器分为顺序容器和关联容器。二者的本质区别在于：关联容器通过键（key）存储和读取元素，而顺序容器的元素排列次序与元素值无关，是由元素添加到容器的顺序决定的，通过元素在容器中的位置顺序存储和读取元素。

标准库定义了三种顺序容器类型，包括：vector、list、deque。

容器内的元素类型必须至少满足2个条件：可复制和可赋值。

vector是连续存储的，能支持快速随机访问，list不需要连续存储，能在中间快速插入和删除，deque是双端队列，具有后进先出的特性。

所有的迭代器范围都是左闭合区间，［begin，end）包括begin，但不包括end

list迭代器不支持算术运算，不支持关系运算，也没有下标操作，只有最简单的自增 自减 相等 不等 运算。

标准库还提供了三种容器适配器：后进先出的stack、先进先出的queue和有优先级管理的priority\_queue。stack可以建立在vector、list、deque上，queue要能提供push\_front操作，不能建立在vector上，priority\_queue要求提供随机访问能力，只能建立在vector或deque上。

标准库提供的关联容器有map、set、multimap和multiset。

map存储键值对，set存储单个键，multimap和multiset支持同一个键多次出现。

set与map不同的地方在于：set仅有key\_type类型，它的value\_type 也就是key\_type;而且set不提供下标操作。map支持下标操作，而且与下标访问数组或者vector的行为截然不同：用下标访问不存在的元素将导致在map容器中添加一个新元素。

【解释】 无

## 6.4 数据成员指针

【题目编号】 CSD-06-604-1001

【知 识 点】 C++面向对象 - 数据成员指针 - 类成员指针

【难易程度】 70

【题目描述】

已知一个类X, ( )是定义指向类X成员函数的指针,假设类有3个公有成员: void f1(int), void f2(int)和int a.

【选 项】

A、X\*P

B、int X::\*pc=&X::a

C、void(X::\*pA)()

D、X\*p[10]

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-604-1002

【知 识 点】 C++面向对象 - 数据成员指针 - 类成员指针

【难易程度】 70

【题目描述】

指向类X中类型为type的数据成员的指针的说明形式为( )。

【选 项】

A、type X::\*pointer

B、x::type\*pointer

C、type \*X::pointer

D、X::\*tyoe pointer

【正确答案】 A

【解释】 无

## 6.5 类的数据成员

【题目编号】 CSD-06-605-1001

【知 识 点】 C++面向对象 - 类的数据成员 - 成员变量初始化

【难易程度】 60

【题目描述】

有下面类的说明，有错误的语句是（ ）。

【选 项】

class X

{

A、int a=2;

B、X();

public:

C、X(int val):

D、~X();

};

【正确答案】 A

【解释】

类定义中不能给成员变量赋值。

【题目编号】 CSD-06-605-1002

【知 识 点】 C++面向对象 - 类的数据成员 - 静态数据成员

【难易程度】 60

【题目描述】

下列静态数据成员的特性中，( )是错误的。

【选 项】

A、说明静态数据成员时前边要加关键字static来修饰

B、静态数据成员在类体外进行初始化

C、引用静态数据成员时,要在静态数据成员名前加<类名>和作用域运算符

D、静态数据成员不是所有对象所共有的

【正确答案】 D

【解释】 无

【题目编号】 CSD-06-605-1003

【知 识 点】 C++面向对象 - 类的数据成员 - 常数据成员

【难易程度】 60

【题目描述】

关于常数据成员的说法不正确的是( )。

【选 项】

A、常数据成员的定义形式与一般常量的定义形式相同，只不过常数据成员的定义必须出现在类体中.

B、常数据成员必须进行初始化,并且不能被更新

C、常数据成员通过构造函数的成员初始化列表进行初始化

D、常数据成员可以在定义时直接初始化

【正确答案】 D

【解释】 无

【题目编号】 CSD-06-605-1004

【知 识 点】 C++面向对象 - 类的数据成员 - 类的静态成员

【难易程度】 60

【题目描述】

（ ）只能访问静态成员变量，静态成员函数和类以外的函数和数据不能访问类中的非静态成员变量。

【选 项】

A、静态函数

B、虚函数

C、构造函数

D、析构函数

【正确答案】 A

【解释】 无

【题目编号】 CSD-06-605-1005

【知 识 点】 C++面向对象 - 类的数据成员 - 常成员函数

【难易程度】 60

【题目描述】

对于常成员函数，下列描述正确的是（ ）。

【选 项】

A、常成员函数只能修改常数据成员

B、常成员函数只能修改一般数据成员

C、常成员函数不能修改任何数据成员

D、成员函数只能通过常对象调用

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-605-1006

【知 识 点】 C++面向对象 - 类的数据成员 - 访控权限

【难易程度】 60

【题目描述】

由于数据隐藏的需要，静态，数据成员通常被说明为（ ）。

【选 项】

A、私有的

B、公有的

C、保护的

D、不可访问的

【正确答案】 A

【解释】 无

【题目编号】 CSD-06-605-1007

【知 识 点】 C++面向对象 - 类的数据成员 - 成员函数

【难易程度】 60

【题目描述】

关于成员函数的特征的下述描述中，错误的是（ ）。

【选 项】

A、成员函数一定是内联函数

B、成员函数可以重载

C、成员函数可以设置参数的默认值

D、成员函数可以是静态的

【正确答案】 A

【解释】 无

【题目编号】 CSD-06-605-1008

【知 识 点】 C++面向对象 - 类的数据成员 - 静态成员

【难易程度】 60

【题目描述】

下面对静态数据成员的描述中，正确的是（ ）

【选 项】

A、静态数据成员是类的所有对象共享的数据

B、类的没个对象都有自己的静态数据成员

C、类的不同对象有不同的静态数据成员值

D、静态数据成员不能通过类的对象调用

【正确答案】 A

【解释】 无

【题目编号】 CSD-06-605-1009

【知 识 点】 C++面向对象 - 类的数据成员 - 常成员函数

【难易程度】 60

【题目描述】

已知:print()函数是一个类的常成员函数,它无返回值,下列表示中,( )是正确的。

【选 项】

A、void print() const

B、const coid print()

C、viod const print()

D、void print(const)

【正确答案】 A

【解释】 无

【题目编号】 CSD-06-605-1010

【知 识 点】 C++面向对象 - 类的数据成员 - 类的静态成员

【难易程度】 60

【题目描述】

类的静态成员( )。

【选 项】

A、是指静态数据成员

B、是指静态函数成员

C、为该类的所有对象共享

D、遵循类的其他成员做遵循的所有访问权限

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-605-1011

【知 识 点】 C++面向对象 - 类的数据成员 - 类的成员函数声明和定义

【难易程度】 60

【题目描述】

下列说法正确的是( )。

【选 项】

A、类定义中只能说明函数成员的函数头，不能定义函数体

B、类中的函数成员只能可以在类体内定义,不可能在类体外定义

C、类中的函数成员在类体外定义时必须要与类声明在同一文件中

D、在类体外定义的函数成员不能操作该类的私有数据成员

【正确答案】 B

【解释】 无

【题目编号】 CSD-06-605-1012

【知 识 点】 C++面向对象 - 类的数据成员 - 静态成员

【难易程度】 60

【题目描述】

静态成员遵循类的其他成员所遵循的访问限制，除了（ ）

【选 项】

A、静态成员函数

B、静态数据成员初始化

C、私有静态数据成员

D、公有静态成员函数

【正确答案】 B

【解释】 无

【题目编号】 CSD-06-605-1013

【知 识 点】 C++面向对象 - 类的数据成员 - 类数据成员的指针

【难易程度】 70

【题目描述】

可以用“px->\*p”来访问类A的对象a 的数据成员，则下面正确的是（ ）。

【选 项】

A、px是指向类数据成员的指针

B、p是指向类数据成员的指针

C、p是指向类的对象的指针

D、无法确定指针类型

【正确答案】 B

【解释】 无

【题目编号】 CSD-06-605-1014

【知 识 点】 C++面向对象 - 类的数据成员 - 类数据成员生存期

【难易程度】 70

【题目描述】

关于类中数据成员的生存期的说法正确的是（ ）。

【选 项】

A、与对象的生存期无关

B、比对象的生存期长

C、比对象的生存期短

D、有对象的生存期决定

【正确答案】 D

【解释】 无

【题目编号】 CSD-06-605-4001

【知 识 点】 C++面向对象 - 类的数据成员 - 静态成员与非静态成员区别

【难易程度】 70

【题目描述】

类的静态成员和非静态成员有什么区别？

【正确答案】

类的静态成员每个类只有一个，即是属于本类的；类的非静态成员每个对象都有一份。

【解释】 无

【题目编号】 CSD-06-605-4002

【知 识 点】 C++面向对象 - 类的数据成员 - 类成员函数的重载、覆盖和隐藏的区别

【难易程度】 80

【题目描述】

类成员函数的重载、覆盖和隐藏的区别？

【正确答案】

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。

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。

“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）。

【解释】 无

## 6.6 面向对象基础

【题目编号】 CSD-06-606-4001

【知 识 点】 C++面向对象 - 面向对象基础 - 面向对象特征

【难易程度】 60

【题目描述】

面向对象的三个基本特征，并简单叙述之？

【正确答案】

（1）继承：继承（inheritance）机制是面向对象程序设计使代码可以复用的最重要的手段，它提供了一种明确表述共性的方法。对象的一个新类可以从现有的类中派生，这个过程称为类继承。新类继承了原始类的特性，新类称为原始类的派生类（子类），而原始类称为新类的基类（父类）。派生类可以从它的基类那里继承方法和实例变量，并它允许程序员在保持原有类特性的基础上进行扩展，增加功能，使之更适合特殊的需要。

（2）封装：封装（encapsulation）是面向对象程序设计最基本的特性，把数据（属性）和函数（方法）合成一个整体，对数据的访问只能通过公有的接口访问，封装的对象，这些对象通过一个受保护的接口访问其他对象。

（3）多态性：多态性（polymorphism）多态性是指允许不同类的对象对同一消息作出响应。多态性语言具有灵活、抽象、行为共享、代码共享的优势，很好的解决了应用程序函数同名问题，函数的重载，运算符的重载，属于多态性中的编译时的多态性，虚函数属于运行时多态。

【解释】 无

## 6.7 友元

【题目编号】 CSD-06-607-1001

【知 识 点】 C++面向对象 - 友元 - 友元函数

【难易程度】 60

【题目描述】

关于类的友元函数的下述描述中，（ ）是正确的。

【选 项】

A、友元函数可以被继承。

B、一个类的友元函数可以访问这个类的所有成员。

C、一个类的友元函数必须在这个类中定义成内联函数。

D、一个类的友元函数的返回类型必须是这个类的类型标识符。

【正确答案】 B

【解释】 无

【题目编号】 CSD-06-607-1002

【知 识 点】 C++面向对象 - 友元 - 友元

【难易程度】 60

【题目描述】

类A是类B的友元,类B是类C的友元,则( )是正确的。

【选 项】

A、类B是类A的友元

B、类C是类A的友元

C、类A是类C的友元

D、以上都不对

【正确答案】 D

【解释】 无

【题目编号】 CSD-06-607-1003

【知 识 点】 C++面向对象 - 友元 - 友元

【难易程度】 60

【题目描述】

下列的各类函数中，( )不是类的成员函数

【选 项】

A、构造函数

B、析构函数

C、友元函数

D、拷贝构造函数

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-607-1004

【知 识 点】 C++面向对象 - 友元 - 友元

【难易程度】 70

【题目描述】

一个类的友元函数能够访问该类的( )。

【选 项】

A、私有成员

B、保护成员

C、公有成员

D、所有成员

【正确答案】 D

【解释】 无

【题目编号】 CSD-06-607-1005

【知 识 点】 C++面向对象 - 友元 - 友元

【难易程度】 70

【题目描述】

下面关于友元的描述中，错误的是( )。

【选 项】

A、友元函数可以访问该类的私有数据成员。

B、一个类的友元类中的成员函数都是这个类的友元函数。

C、友元可以提高程序的运行效率。

D、类与类之间的友元关系可以继承。

【正确答案】 D

【解释】 无

【题目编号】 CSD-06-607-1006

【知 识 点】 C++面向对象 - 友元 - 友元

【难易程度】 60

【题目描述】

友元关系不能（ ）。

【选 项】

A、继承

B、是类与类之间是关系

C、是一个类的成员函数与另一个类的关系

D、提高程序的运行效率

【正确答案】 A

【解释】

继承是类与类之间的关系。

【题目编号】 CSD-06-607-1007

【知 识 点】 C++面向对象 - 友元 - 友元

【难易程度】 60

【题目描述】

允许访问类的所有对象的私有成员,公有成员和保护成员的是( )。

【选 项】

A、友元函数

B、成员函数

C、内联函数

D、重载函数

【正确答案】 A

【解释】 无

【题目编号】 CSD-06-607-1008

【知 识 点】 C++面向对象 - 友元 - 友元

【难易程度】 60

【题目描述】

友元的作用是（ ）。

【选 项】

A、提高程序是运行效率

B、加强类的封装性

C、实现数据的隐藏性

D、增加成员函数的种类

【正确答案】 A

【解释】 无

【题目编号】 CSD-06-607-1009

【知 识 点】 C++面向对象 - 友元 - 友元

【难易程度】 60

【题目描述】

一个类的友元函数或友元类能够通过成员操作符访问该类的（ ）。

【选 项】

A、私有成员

B、公有成员

C、保护成员

D、公有成员、保护成员和私有成员

【正确答案】 D

【解释】 无

## 6.8 class和struct的区别

【题目编号】 CSD-06-608-4001

【知 识 点】 C++面向对象 - class和struct的区别 - 结构与联合的区别

【难易程度】 70

【题目描述】

结构与联合有何区别？

【正确答案】

（1）struct和union都是由多个不同的数据类型成员组成，但在任何同一时刻，union中只存放了一个被选中的成员,而struct的所有成员都存在。在struct中，各成员都占有自己的内存空间，它们是同时存在的。一个struct变量的总长度等于所有成员长度之和。在Union中，所有成员不能同时占用它的内存空间，它们不能同时存在。union变量的长度等于最长的成员的长度。

（2）对于union的不同成员赋值,将会对其它成员重写,原来成员的值就不存在了,而对于struct的不同成员赋值是互不影响的。

结构体是多个数据的集合，可以保存多个数据；而联合union只保存一个数据，但是可以按照不同类型来读取。

【解释】 无

【题目编号】 CSD-06-608-4002

【知 识 点】 C++面向对象 - class和struct的区别 - class和struct

【难易程度】 70

【题目描述】

class和struct有什么区别？

【正确答案】

分两种情况进行分析。

C的struct与C++的class的区别：struct只是作为一种复杂数据类型定义，不能用于面向对象编程。

C++中的struct和class的区别：对于成员访问权限以及继承方式，class中默认的是private，而struct中则是public。class还可以用于表示模板类型，struct则不行。

【解释】 无

## 6.9 成员变量

【题目编号】 CSD-06-609-1001

【知 识 点】 C++面向对象 - 成员变量 - 静态数据成员的理解和使用

【难易程度】 60

【题目描述】

下面对静态数据成员的描述中，正确的选项是哪一个？

【选 项】

A．静态数据成员可以在类体内进行初始化；

B．静态数据成员不可以被类的对象调用：

C．静态数据成员不受Private控制符作用；

D．静态数据成员可以直接用类名调用。

【正确答案】 CD

【解释】

A错误。静态数据成员必须在类外面初始化，以示与普通数据成员的区别。

B错误。静态数据成员是类的成员，它为类的所有的对象所共享。

【题目编号】 CSD-06-609-4001

【知 识 点】 C++面向对象 - 成员变量 - 静态成员变量的理解和使用

【难易程度】 60

【题目描述】

分析代码写结果―静态成员变量的使用。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | #include <iostream>  using namespace std;  class Myclass  {  public:  Myclass(int a, int b, int c);  void GetNumber();  void GetSum();  private:  int A;  int B;  int C;  int Num;  static int Sum;  };  int Myclass::Sum = 0;  Myclass::Myclass(int a, int b, int c)  {  A = a;  B = b;  C = c;  Num = A + B + C;  Sum = A + B + C;  }  void Myclass::GetNumber()  {  cout << "Number = " << Num << endl;  }  void Myclass::GetSum()  {  cout << "Sum = " << Sum << endl;  }  int main()  {  Myclass M(3, 7, 10), N(14, 9, 11);  M.GetNumber();  N.GetNumber();  M.GetSum();  N.GetSum();  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4 | Number = 20  Number = 34  Sum = 34  Sum = 34 |

【解释】

本题考查的是静态成员与非静态成员的区别。静态成员是该类类型的全局变量。对于非静态成员，每个类对象都有自己的复制品，而静态成员对每个类的类型只有一个复制品。静态成员只有一个，由该类类型的所有对象共享访问。

Myclass类有GetNumber()和GetSum()两种方法，它们分别输出成员变量Num和Sum的值。

main()函数中定义了两个Myclass的对象，并分别调用它们的GetNumber()和GetSum()。

Num成员为普通类型，它被Myclass类的对象所有。因此两次打印出来的值不一样。Sum成员为静态类型，它被Myclass类所有，也被Myclass类的所有对象所共享。因此两次打印出来的值是相同的。

【题目编号】 CSD-06-609-4002

【知 识 点】 C++面向对象 - 成员变量 - 对静态成员变量的理解

【难易程度】 60

【题目描述】

与全局对象相比，使用静态数据成员有什么优势？

【正确答案】

主要有以下所述两种优势。

静态数据成员没有进入程序的全局名字空间，因此不存在程序中其他全局名字冲突的

问题。

使用静态数据成员可以隐藏信息。因为静态成员可以是Private成员，而全局对象不能。

【解释】 无

【题目编号】 CSD-06-609-4003

【知 识 点】 C++面向对象 - 成员变量 - 初始化列表和赋值的区别

【难易程度】 70

【题目描述】

有哪几种情况只能用初始化列表，而不能用赋值？

【正确答案】

当类中含有const、引用成员变量以及基类的构造函数都需要初始化列表。

【解释】 无

## 6.10 构造函数和析构函数

【题目编号】 CSD-06-610-1001

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 构造函数和析构函数

【难易程度】 50

【题目描述】

下面说法中正确的是（ ）。

【选 项】

A、一个类只能定义一个构造函数，但可以定义多个析构函数

B、一个类只能定义一个析构函数，但可以定义多个构造函数

C、构造函数与析构函数同名，只是名字前加了一个求反符号（~）

D、构造函数可以指定返回类型，而析构函数不能指定任何返回类型，即使是void类型也不可以

【正确答案】 B

【解释】 无

【题目编号】 CSD-06-610-1002

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 析构函数

【难易程度】 50

【题目描述】

有关析构函数的说法不正确的是( )。

【选 项】

A、析构函数有且仅有一个

B、析构函数和构造函数一样可以有形参

C、析构函数的功能是用来释放一个对象

D、析构函数无任何函数类型

【正确答案】 B

【解释】

析构函数不能有形参。

【题目编号】 CSD-06-610-1003

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 析构函数

【难易程度】 50

【题目描述】

有关析构函数的错误的说法是（ ）。

【选 项】

A、一个类的析构函数有且只有一个。

B、析构函数无函数类型。

C、析构函数和构造函数一样，可以有形参。

D、析构函数的作用是在对象生命期结束时，收回分配给其的内存空间。

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-610-1004

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 构造函数

【难易程度】 60

【题目描述】

已知X类.则当程序执行到语句 X array[3];时,调用了（ ）次构造函数。

【选 项】

A、0

B、1

C、2

D、3

【正确答案】 D

【解释】

每一个数组元素都会调用类的构造函数。

【题目编号】 CSD-06-610-1005

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 构造函数

【难易程度】 60

【题目描述】

下列关于构造函数的描述中,错误的是( )。

【选 项】

A、构造函数可以设置默认参数

B、构造函数在说明类变量时自动执行

C、构造函数可以对静态数据成员进行初始化

D、构造函数可以重载

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-610-1006

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 拷贝构造函数

【难易程度】 60

【题目描述】

假定X为一个类，则该类的拷贝构造函数的声明语句为（ ）。

【选 项】

A、My Class(My Class x)

B、My Class&(My Class x)

C、My Class(My Class &x)

D、My Class(My Class \*x)

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-610-1007

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 缺省构造函数

【难易程度】 50

【题目描述】

缺省析构函数的函数体是（ ）。

【选 项】

A、不存在

B、随机产生的

C、空的

D、无法确定的

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-610-1008

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 构造函数

【难易程度】 50

【题目描述】

（ ）的功能是对对象进行初始化。

【选 项】

A、析构函数

B、数据成员

C、构造函数

D、静态数据成员

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-610-1009

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 析构函数

【难易程度】 50

【题目描述】

对于任意一个类，析构函数的个数最多为（ ）。

【选 项】

A、0

B、1

C、2

D、3

【正确答案】 B

【解释】 无

【题目编号】 CSD-06-610-1010

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 析构函数

【难易程度】 60

【题目描述】

（ ）是析构函数的特征。

【选 项】

A、析构函数可以有一个或多个参数

B、析构函数定义只能在类体内

C、析构函数名与类名不同

D、一个类中只能定义一个析构函数

【正确答案】 D

【解释】 无

【题目编号】 CSD-06-610-1011

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 构造函数

【难易程度】 60

【题目描述】

有关构造函数说法不正确的是( )。

【选 项】

A、构造函数的名字和类的名字一样。

B、在创建对象时，系统自动调用构造函数

C、构造函数无任何函数类型

D、构造函数有且只有一个

【正确答案】 D

【解释】 无

【题目编号】 CSD-06-610-1012

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 构造函数

【难易程度】 70

【题目描述】

关于对象成员的构造函数的调用顺序，说法正确的是( )。

【选 项】

A、与他们在成员初始化列表中给出的顺序相同

B、与折构函数的调用顺序相同

C、与他们在类中声明顺序相同

D、以上说法都不对

【正确答案】 C

【解释】 无

【题目编号】 CSD-06-610-1013

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 构造函数

【难易程度】 60

【题目描述】

下列（ ）不是构造函数的特征。

【选 项】

A、构造函数的函数名与类名相同

B、构造函数可以重载

C、构造函数可以设置默认参数

D、构造函数必须指定类型说明

【正确答案】 D

【解释】 无

【题目编号】 CSD-06-610-1014

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 拷贝构造函数

【难易程度】 50

【题目描述】

通常拷贝构造函数的参数表是( )。

【选 项】

A、某个对象名

B、某个对象的成员名

C、某个对象的引用名

D、某个对象的指针名

【正确答案】 C

【解释】

防止递归调用死循环。

【题目编号】 CSD-06-610-4001

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 构造函数

【难易程度】 70

【题目描述】

有哪几种情况只能用初始化列表而不能使用赋值。

【正确答案】

当类中含有const成员变量和引用成员变量时，基类的构造函数都需要初始化表。

【解释】 无

【题目编号】 CSD-06-610-4002

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 构造函数

【难易程度】 70

【题目描述】

main函数执行以前，会执行什么代码？

【正确答案】

全局对象的构造函数在main函数之前执行。

【解释】 无

【题目编号】 CSD-06-610-4003

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 类默认成员函数

【难易程度】 70

【题目描述】

C++中的空类，默认情况下会产生哪些类成员函数？

【正确答案】

C++的空类中，默认情况下会产生默认构造函数、复制构造函数、析构函数、赋值函数以及取值运算。

【解释】 无

【题目编号】 CSD-06-610-4004

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 构造函数和析构函数

【难易程度】 60

【题目描述】

构造函数和析构函数是否可以被重载？

【正确答案】

构造函数可以被重载，因为构造函数可以有多个，而且可以带参数。

析构函数不可以被重载。因为析构函数只能有一个，并且不能带参数。

【解释】 无

【题目编号】 CSD-06-610-4005

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 构造函数

【难易程度】 70

【题目描述】

构造函数explicit与普通构造函数的区别。

【正确答案】

普通构造函数能够被隐式调用，而explicit构造函数只能被显示调用。

【解释】 无

【题目编号】 CSD-06-610-4006

【知 识 点】 C++面向对象 - 构造函数和析构函数 - 析构函数的执行顺序

【难易程度】 60

【题目描述】

分析代码写结果——析构函数的执行顺序。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | #include <iostream>  using namespace std;  class A  {  private:  int a;  public:  A(int aa){ a = aa; }  ~A() { cout << "Destructor A!" << a << endl; }  };  class B : public A  {  private:  int b;  public:  B(int aa = 0, int bb = 0) : A(aa) { b = bb; }  ~B() { cout << "Destructor B!" << b << endl; }  };  int main()  {  B obj1(5), obj2(6, 7);  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4 | Destructor B!7  Destructor A!6  Destructor B!0  Destructor A!5 |

【解释】 无

## 6.11 复制构造函数和赋值函数

【题目编号】 CSD-06-611-1001

【知 识 点】 C++面向对象 - 复制构造函数和赋值函数 - 构造函数

【难易程度】 60

【题目描述】

形如X::X(X&)的构造函数称为( )。

【选 项】

A、拷贝函数

B、复制函数

C、初始化函数

D、拷贝初始化构造函数

【正确答案】 D

【解释】 无

【题目编号】 CSD-06-611-4001

【知 识 点】 C++面向对象 - 复制构造函数和赋值函数 - 拷贝构造函数

【难易程度】 60

【题目描述】

那些情况需要调用拷贝构造函数，请举例？

【正确答案】

1) 对象作函数参数值传递方式传入函数体;

2) 对象作函数返回值值传递方式从函数返回;

3) 对象用于给另外对象进行初始化(常称赋值初始化)。

【解释】 无

【题目编号】 CSD-06-611-4002

【知 识 点】 C++面向对象 - 复制构造函数和赋值函数 - 复制构造函数

【难易程度】 70

【题目描述】

复制构造函数是什么？什么情况下会用到它？什么是深复制和浅复制？

【正确答案】

复制构造函数是一种特殊的构造函数，它由编译器调用来完成一些基于同一类的其他对象的构造及初始化。

如果在类中没有显式声明一个复制构造函数，那么，编译器会私下里制定一个函数来进行对象之间的位复制。这个隐含的复制构造函数简单地关联了所有的类成员。

在C++中，3种对象需要复制，此时复制构造函数将会被调用：（1）一个对象以值传递的方式传入函数体。（2）一个对象以值传递的方式从函数返回。（3）一个对象需要通过另外一个对象进行初始化。

浅复制是让新旧两个对象指向同一个外部的内容，而深复制是指为新对象制作了外部对象的独立复制。如果复制的对象中引用了某个外部的内容（例如分配在堆上的数据），那么在复制这个对象的时候，让新旧两个对象指向同·个外部的内容，就是浅复制；如果在复制这个对象的时候为新对象制作了外部对象的独立复制，就是深复制。

【解释】 无

【题目编号】 CSD-06-611-4003

【知 识 点】 C++面向对象 - 复制构造函数和赋值函数 - 构造函数与赋值函数的区别

【难易程度】 70

【题目描述】

复制构造函数与赋值函数有什么区别？

【正确答案】

构造函数与赋值函数有以下3个区别。

复制构造是一个对象初始化一块内存区域，这块内存就是新对象的内存区；而赋值函数是对于一个已经被初始化的对象来进行赋值操作。

一般来说在数据成员包含指针对象的时候，应考虑两种不同的处理需求：一种是复制指针对象，另一种是引用指针对象。复制构造函数大多数情况下是复制，赋值函数则是引用对象。

实现不一样。复制构造函数首先是一个构造函数，它调用的时候是通过参数的对象初始化产生一个对象。赋值函数则是把一个新的对象赋值给一个原有的对象，所以如果原来的对象中有内存分配要先把内存释放掉，而且还要检查一下两个对象是不是同一个对象，如果是，则不做任何操作。

【解释】 无

【题目编号】 CSD-06-611-4004

【知 识 点】 C++面向对象 - 复制构造函数和赋值函数 - 构造函数、析构函数和赋值函数的编写方法

【难易程度】 60

【题目描述】

编写类string的构造函数、析构函数和赋值函数。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120 | class CMyString  {  friend std::ostream& operator<<( std::ostream& os, const CMyString& str);  private:  char\* m\_pData; // 私有变量保存字符串  public:  CMyString( const char\* str = NULL ); // 构造函数  CMyString( const CMyString& str ); // 拷贝构造函数  ~CMyString( void ); // 析构函数  CMyString& operator=( const CMyString& str ); // 赋值运算符  CMyString operator+( const CMyString& str ); // 字符串连接  bool operator==( const CMyString& str ); // 判断相等  char operator[]( int idx ); // 数组索引  int getLength(); // 返回长度  };    CMyString::CMyString( const char\* str )  {  if ( !str )  {  this->m\_pData = 0;  }  else  {  this->m\_pData = new char[ strlen( str ) + 1 ];  strcpy( this->m\_pData, str );  }  }    CMyString::CMyString( const CMyString& str )  {  if ( !str.m\_pData )  {  this->m\_pData = 0;  }  else  {  this->m\_pData = new char[ strlen( str.m\_pData ) + 1 ];  strcpy( this->m\_pData, str.m\_pData );  }  }    CMyString::~CMyString( void )  {  if ( this->m\_pData)  {  delete[] this->m\_pData;  this->m\_pData = 0;  }  }    CMyString& CMyString::operator=( const CMyString& str)  {  if ( this != &str )  {  delete[] this->m\_pData;  if ( !str.m\_pData )  {  this->m\_pData = 0;  }  else  {  this->m\_pData = new char[ strlen( str.m\_pData ) + 1 ];  strcpy( this->m\_pData, str.m\_pData );  }  }  return \*this;  }    CMyString CMyString::operator+( const CMyString& str )  {  CMyString newString;  if ( !str.m\_pData )  {  newString = \*this;  }  else if ( !this->m\_pData )  {  newString = str;  }  else  {  newString.m\_pData = new char[ strlen( this->m\_pData ) +  strlen( str.m\_pData ) + 1 ];  strcpy( newString.m\_pData, this->m\_pData );  strcat( newString.m\_pData, str.m\_pData );  }  return newString;  }    bool CMyString::operator==( const CMyString& str )  {  if ( strlen(this->m\_pData) != strlen( str.m\_pData ) )  {  return false;  }  else  {  return strcmp( this->m\_pData, str.m\_pData ) ? false : true;  }  }  char CMyString::operator[]( int idx)  {  if ( idx > 0 && idx < strlen( this->m\_pData ) )  return this->m\_pData[idx];  }  int CMyString::getLength()  {  return strlen(this->m\_pData);  }    std::ostream& operator<<( std::ostream& os,  const CMyString& str )  {  os << str.m\_pData;  return os;  } |

【解释】 无

## 6.12 函数重载和运算符重载

【题目编号】 CSD-06-612-1001

【知 识 点】 C++面向对象 - 函数重载和运算符重载 - 运算符重载

【难易程度】 60

【题目描述】

下列运算符中，（ ）运算符在C++中不能重载。

【选 项】

A、？：

B、+

C、-

D、〈=

【正确答案】 A

【解释】

C++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。

【题目编号】 CSD-06-612-1002

【知 识 点】 C++面向对象 - 函数重载和运算符重载 - 运算符重载

【难易程度】 60

【题目描述】

下列关于运算符重载的描述中， （ ）是正确的。

【选 项】

A、运算符重载可以改变操作数的个数

B、运算符重载可以改变优先级

C、运算符重载可以改变结合性

D、运算符重载不可以改变语法结构

【正确答案】 D

【解释】

运算符重载不可以改变语法结构。语法结构是语言本身决定的，用户无法进行改变。

【题目编号】 CSD-06-612-1003

【知 识 点】 C++面向对象 - 函数重载和运算符重载 - 友元运算符重载

【难易程度】 70

【题目描述】

友元运算符obj1〉obj2被C++编译器解释为（ ）。

【选 项】

A、operator>（obj1,obj2）

B、>(obj1,obj2)

C、jobj2.operator>(obj1)

D、obj1.operator(obj2)

【正确答案】 A

【解释】

友元解释为operator > (obj1, obj2)，成员解释为obj1.operator>(obj2)

【题目编号】 CSD-06-612-1004

【知 识 点】 C++面向对象 - 函数重载和运算符重载 - 函数重载

【难易程度】 50

【题目描述】

采用函数重载的目的在于。

【选 项】

A、实现共享 B、减少空间 C、提高速度 D、使用方便，提高可靠性

【正确答案】 D

【解释】 无

【题目编号】 CSD-06-612-1005

【知 识 点】 C++面向对象 - 函数重载和运算符重载 - 重载函数

【难易程度】 60

【题目描述】

下列对重载函数的描述中，( )是错误的。

【选 项】

A、重载函数中不允许使用默认参数

B、重载函数中编译时根据参数表进行选择

C、不要使用重载函数来描述毫无相干的函数

D、构造函数重载将会给初始化带来多种方式

【正确答案】 A

【解释】 无

【题目编号】 CSD-06-612-1006

【知 识 点】 C++面向对象 - 函数重载和运算符重载 - 函数重载

【难易程度】 60

【题目描述】

系统在调用重载函数时往往根据一些条件确定哪个重载函数被调用，在下列选项中，不能作为依据的是（ ）。

【选 项】

A、参数个数 B、参数的类型 C、函数名称 D、函数的类型

【正确答案】 D

【解释】 无

【题目编号】 CSD-06-612-1007

【知 识 点】 C++面向对象 - 函数重载和运算符重载 - 函数重载

【难易程度】 70

【题目描述】

对定义重载函数的下列要求中，( )是错误的。

【选 项】

A、要求参数的个数不同

B、要求参数中至少有一个类型不同

C、要求参数个数相同时，参数类型不同

D、要求函数的返回值不同

【正确答案】 D

【解释】

重载规则：

1）必须在同一作用域中

2）基于形参表不同而构成重载。

3）基于指针（或引用）形参是否为const也可以构成重载

4）在类中，基于成员函数是否为const可以构成重载（成员函数的const就是把隐含的this指针const了）

【题目编号】 CSD-06-612-4001

【知 识 点】 C++面向对象 - 函数重载和运算符重载 - 重载概念

【难易程度】 50

【题目描述】

什么是函数重载？为什么C不支持函数重载，而C++能支持函数重载？

【正确答案】

函数重载是用来描述同名函数具有相同或者相似功能，但数据类型或者是参数不同的函数管理操作。

函数名经过C++编译器处理后包含了原函数名、函数参数数量及返回类型信息，而C不会对函数名进行处理。

【解释】 无

【题目编号】 CSD-06-612-4002

【知 识 点】 C++面向对象 - 函数重载和运算符重载 - 重载和覆写之间区别的理解

【难易程度】 70

【题目描述】

重载和覆写有什么区别？

【正确答案】

覆写（overriding）是指子类改写父类的方法；重载（overloading）是指同一个函数的不同版本之间参数不同。

重载是编写一个与己有函数同名但是参数表不同（参数数量或参数类型不同）的方法，它具有如一卜所示的特征：（1）方法名必须相同。（2）参数列表必须不相同，与参数列表的顺序无关。（3）返回值类型可以不相同。

覆写是派生类重写基类的虚函数，它具有如下所示的特征。（1）只有虚方法和抽象方法才能够被覆写。（2）具有相同的函数名。（3）具有相同的参数列表。（4）具有相同的返回值类型。

重载是一种语法规则，由编译器在编译阶段完成，不属于面向对象的编程；而覆写是由运行阶段决定的，是面向对象编程的重要特征。

【解释】 无

【题目编号】 CSD-06-612-4003

【知 识 点】 C++面向对象 - 函数重载和运算符重载 - 运算符重载

【难易程度】 70

【题目描述】

请问下述代码中: int operator+(）起什么作用？this是什么？ccc的值最终为多少？

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | class Fruit  {  public:  Fruit()  {  weight = 2;  }  Fruit(int w)  {  weight = w;  }  int operator+(Fruit f)  {  return this->weight \* f.weight;  }  private:  int weight;  };  Fruit aaa;  Fruit bbb(4);  int ccc = aaa + bbb; |

【正确答案】

int operator+(）表示重载类的“+”号运算符，this表示对象本身的指针，本例中它指向类的对象aaa；ccc最终的结果为8(8 = 2 \* 4)。

【解释】 无

# 第7篇 C++继承和多态

## 7.1 继承的概念

【题目编号】 CSD-07-701-1001

【知 识 点】 C++继承和多态 - 继承的概念 - 公有派生

【难易程度】 60

【题目描述】

在公有派生情况下，有关派生类对象和基类对象的关系，不正确的叙述是（ ）。

【选 项】

A、派生类的对象可以赋给基类对象

B、派生类的对象可以初始化基类的引用

C、派生类的对象可以直接访问基类的成员

D、派生类的对象的地址可以赋给指向基类的指针

【正确答案】 C

【解释】 无

【题目编号】 CSD-07-701-1002

【知 识 点】 C++继承和多态 - 继承的概念 - 基类和派生类

【难易程度】 60

【题目描述】

对基类和派生类的描述中错误的是（ ）。

【选 项】

A、派生类是基类的具体化

B、基类继承了派生类的属性

C、派生类是基类定义的延续

D、派生类是基类的特殊化

【正确答案】 B

【解释】 无

【题目编号】 CSD-07-701-1003

【知 识 点】 C++继承和多态 - 继承的概念 - 动态联编

【难易程度】 70

【题目描述】

关于动态联编的下列叙述中，（ ）是错误的。

【选 项】

A、动态联编是以虚函数为基础的

B、动态联编调用虚函数操作是指向对象的指针或引用

C、动态联编是在运行时确定所调用的函数代码的

D、动态联编是在编译时确定操作函数的

【正确答案】 D

【解释】 无

【题目编号】 CSD-07-701-1004

【知 识 点】 C++继承和多态 - 继承的概念 - 构造函数的初始化列表

【难易程度】 70

【题目描述】

派生类的构造函数的成员初始化列表中，不能包含（ ）。

【选 项】

A、基类的构造函数

B、派生类中子对象的初始化

C、基类的子对象初始化

D、派生类中一般数据成员的初始化

【正确答案】 C

【解释】 无

【题目编号】 CSD-07-701-1005

【知 识 点】 C++继承和多态 - 继承的概念 - 虚函数

【难易程度】 60

【题目描述】

关于函数的描述中，（ ）是正确的。

【选 项】

A、派生类的虚函数与基类的虚函数具有不同的参数个数和类型

B、基类中说明了虚函数后，派生类中其对应的函数一定要说明为虚函数

C、虚函数是一个成员函数

D、虚函数是一个static类型的成员函数

【正确答案】 C

【解释】 无

【题目编号】 CSD-07-701-1006

【知 识 点】 C++继承和多态 - 继承的概念 - 派生类

【难易程度】 60

【题目描述】

派生类的对象对它的基类成员中（ ）是可以访问的。

【选 项】

A、公有继承的公有成员

B、公有继承的私有成员

C、公有继承的保护成员

D、私有继承的公有成员

【正确答案】 A

【解释】 无

【题目编号】 CSD-07-701-1007

【知 识 点】 C++继承和多态 - 继承的概念 - 派生类

【难易程度】 60

【题目描述】

下列对派生类的描述中，（ ）是错误的。

【选 项】

A、一个派生类可以作另一个派生类的基类

B、派生类至少有一个基类

C、派生类的成员除了它自己的成员外，包含了它的基类的成员

D、派生类中继承的基类成员的访问权限到派生类保持不变

【正确答案】 D

【解释】 无

【题目编号】 CSD-07-701-1008

【知 识 点】 C++继承和多态 - 继承的概念 - 虚基类

【难易程度】 50

【题目描述】

在继承关系中，虚基类的作用是（ ）。

【选 项】

A、实现重载

B、构造对象

C、消除二义性

D、使得派生类可以再也不必考虑二义性问题

【正确答案】 C

【解释】 无

【题目编号】 CSD-07-701-4001

【知 识 点】 C++继承和多态 - 继承的概念 - 继承的优缺点

【难易程度】 70

【题目描述】

继承的优缺点。

【正确答案】

类继承是在编译时刻静态定义的，且可直接使用，类继承可以较方便地改变父类的实现。但是类继承也有一些不足之处。首先，因为继承在编译时刻就定义了，所以无法在运行时刻改变从父类继承的实现。更糟的是，父类通常至少定义了子类的部分行为，父类的任何改变都可能影响子类的行为。如果继承下来的实现不适合解决新的问题，则父类必须重写或被其他更适合的类替换。这种依赖关系限制了灵活性并最终限制了复用性。

【解释】 无

【题目编号】 CSD-07-701-4002

【知 识 点】 C++继承和多态 - 继承的概念 - 对C++类继承的3种关系的理解

【难易程度】 60

【题目描述】

C++类继承的3种关系。

【正确答案】

C++中继承主要有3种关系public、protected和private。

（1）public继承。public继承是一种接口继承，子类可以代替父类完成父类接口所声明的行为。此时子类可以自动转换成为父类的接口，完成接口转换。从语法角度上来说，public继承会保留父类中成员（包括函数和变量等）的可见性，也就是说，如果父类中的某个函数是public，那么被子类继承后仍然是public。

（2）protected继承。protected继承是一种实现继承，子类不能代替父类完成父类接口所声明的行为，此时子类不能自动转换成父类的接口．从语法角度上来说，protected继承会将父类中的public可见性的成员修改成为protected可见性，这样在子类中同样可以调用父类的protected和public成员，子类的子类也可以调用被protected继承的父类的protected和public成员。

（3）private继承。private继承是一种实现继承，子类不能代替父类完成父类接口所声明的行为，此时子类不能自动转换成为父类的接口。从语法角度上来说，private继承会将父类中的public和protected可见性的成员修改成为Private可见性，这样虽然子类中同样还是可以调用父类的protected和public成员，但是在子类的子类就不可以再调用被private继承的父类的成员了。

【解释】 无

【题目编号】 CSD-07-701-4003

【知 识 点】 C++继承和多态 - 继承的概念 - 私有继承

【难易程度】 50

【题目描述】

私有继承有什么作用？

【正确答案】

和公有继承相反，如果两个类之间的继承关系为私有，编译器一般不会将派生类对象转换成基类对象；从私有基类继承而来的成员成为派生类的私有成员，即使它们在基类中是保护或公有成员。

【解释】 无

## 7.2 多态的概念

当不同的对象调用相同的名称的成员函数时，可能引起不同的行为（即执行不同的代码）。这种现象称为多态性。将函数调用链接相应函数体的代码的过程称为函数联编（简称联编）。在C++中，根据联编时刻不同，分为静态联编和动态联编。

静态联编，不同的类可以有相同名称的成员函数（甚至还可以有相同的参数，编译器在编译时就对它们进行函数联编，这种在编译时刻进行的联编称为静态联编。静态联编所支持的多态性称为编译时多态性。函数重载就属于编译时多态性。

动态联编，在动态联编中，在程序运行时才能确定调用哪个函数。这种在运行时的函数联编称为动态联编。动态联编所支持的多态性称为运行时多态性。在C++中，只有虚函数才可能是动态联编的。可以通过定义类的虚函数和创建派生类，然后在派生类中重新实现虚函数，实现具有运行时的多态性。

【题目编号】 CSD-07-702-1001

【知 识 点】 C++继承和多态 - 多态的概念 - 动态联编

【难易程度】 70

【题目描述】

动态联编所支持的多态性称为（ ）。

【选 项】

A、虚函数

B、继承

C、编译时的多态性

D、运行时的多态性

【正确答案】 D

【解释】

编译程序在编译阶段并不能确切知道将要调用的函数，只有在程序运行时才能确定将要调用的函数，为此要确切知道该调用的函数，要求联编工作要在程序运行时进行，这种在程序运行时进行联编工作被称为动态联编。

【题目编号】 CSD-07-702-1002

【知 识 点】 C++继承和多态 - 多态的概念 - 虚函数

【难易程度】 60

【题目描述】

关于虚函数，下列表述正确的是（ ）。。

【选 项】

A、如果在重定义虚函数时使用virtual，则该重定义函数仍然是虚函数

B、虚函数不得声明为静态函数

C、虚函数不得声明为另一个类的友元函数

D、派生类必须重新定义基类的虚函数

【正确答案】 B

【解释】 无

【题目编号】 CSD-07-702-1003

【知 识 点】 C++继承和多态 - 多态的概念 - 虚析构

【难易程度】 70

【题目描述】

若析构函数是虚函数，则delete对析构函数的调用采用（ ）。

【选 项】

A、静态联编

B、动态联编

C、重载函数

D、多重函数

【正确答案】 B

【解释】 无

【题目编号】 CSD-07-702-1004

【知 识 点】 C++继承和多态 - 多态的概念 - 多态概念

【难易程度】 50

【题目描述】

不同的对象可以调用相同名称的函数，并可导致完全不同的行为的现象称为（ ）。

【选 项】

A、多态性

B、抽象

C、继承

D、封装

【正确答案】 A

【解释】 无

【题目编号】 CSD-07-702-1005

【知 识 点】 C++继承和多态 - 多态的概念 - 虚函数

【难易程度】 50

【题目描述】

虚函数必须是类的（ ）。

【选 项】

A、成员函数

B、友元函数

C、构造函数

D、析构函数

【正确答案】 A

【解释】 无

【题目编号】 CSD-07-702-1006

【知 识 点】 C++继承和多态 - 多态的概念 - 虚函数

【难易程度】 60

【题目描述】

不能说明为虚函数的是（ ）。

【选 项】

A、析构函数

B、构造函数

C、类的成员函数

D、以上都不对

【正确答案】 B

【解释】 无

【题目编号】 CSD-07-702-1007

【知 识 点】 C++继承和多态 - 多态的概念 - 赋值兼容规则

【难易程度】 70

【题目描述】

下面（ ）的叙述不符合赋值兼容规则。

【选 项】

A、派生类的对象可以赋值给基类的对象

B、基类的对象可以赋值给派生类的对象

C、派生类的对象可以初始化基类的对象

D、派生类的对象的地址可以赋值给指向基类的指针

【正确答案】 B

【解释】 无

【题目编号】 CSD-07-702-1008

【知 识 点】 C++继承和多态 - 多态的概念 - C++联编

【难易程度】 70

【题目描述】

C++有（ ）种联编。

【选 项】

A、1

B、2

C、3

D、4

【正确答案】 B

【解释】

静态联编和动态联编。

【题目编号】 CSD-07-702-1009

【知 识 点】 C++继承和多态 - 多态的概念 - 静态联编

【难易程度】 70

【题目描述】

若要强制C++对虚函数的调用使用静态联编，则在调用中对该函数使用（ ）。

【选 项】

A．成员名限定 B． 指针 C．引用 D．virtual关键字

【正确答案】 A

【解释】 无

【题目编号】 CSD-07-702-1010

【知 识 点】 C++继承和多态 - 多态的概念 - virtual关键字

【难易程度】 60

【题目描述】

用于类中虚成员函数说明的关键字（ ）。

【选 项】

A．virtual B．public C．protected D．private

【正确答案】 A

【解释】 无

【题目编号】 CSD-07-702-1011

【知 识 点】 C++继承和多态 - 多态的概念 - 虚函数

【难易程度】 60

【题目描述】

实现运行时的多态性要使用（ ）。

【选 项】

A、构造函数

B、析构函数

C、重载函数

D、虚函数

【正确答案】 D

【解释】 无

【题目编号】 CSD-07-702-4001

【知 识 点】 C++继承和多态 - 多态的概念 - 多态、虚函数和纯虚函数概念

【难易程度】 70

【题目描述】

多态，虚函数，纯虚函数。

【正确答案】

多态：是对于不同对象接收相同消息时产生不同的动作。C++的多态性具体体现在运行和编译两个方面：在程序运行时的多态性通过继承和虚函数来体现；在程序编译时多态性体现在函数和运算符的重载上。

虚函数：在基类中冠以关键字virtual的成员函数。它提供了一种接口界面。允许在派生类中对基类的虚函数重新定义。

纯虚函数的作用：在基类中为其派生类保留一个函数的名字，以便派生类根据需要对它进行定义。作为接口而存在纯虚函数不具备函数的功能，一般不能直接被调用。从基类继承来的纯虚函数，在派生类中仍是虚函数。如果一个类中至少有一个纯虚函数，那么这个类被称为抽象类（abstract class）。

抽象类中不仅包括纯虚函数，也可包括虚函数。抽象类必须用作派生其他类的基类，而不能用于直接创建对象实例。但仍可使用指向抽象类的指针支持运行时多态性。

【解释】 无

【题目编号】 CSD-07-702-4002

【知 识 点】 C++继承和多态 - 多态的概念 - 多态的实现方法

【难易程度】 60

【题目描述】

实现多态的方法？

【正确答案】

（1）一个基类的引用可以指向它的派生类实例；（2）一个基类的指针可以指向它的派生类实例。

【解释】 无

【题目编号】 CSD-07-702-4003

【知 识 点】 C++继承和多态 - 多态的概念 - 多态作用

【难易程度】 60

【题目描述】

多态的作用？

【正确答案】

主要是两个：（1）隐藏实现细节，使得代码能够模块化；扩展代码模块，实现代码重用；（2）接口重用：为了类在继承和派生的时候，保证使用家族中任一类的实例的某一属性时的正确调用。

【解释】 无

【题目编号】 CSD-07-702-4004

【知 识 点】 C++继承和多态 - 多态的概念 - 构造函数和析构函数

【难易程度】 60

【题目描述】

构造函数可否是虚函数，为什么？析构函数呢，可否是纯虚的呢？

【正确答案】

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。

【解释】 无

【题目编号】 CSD-07-702-4005

【知 识 点】 C++继承和多态 - 多态的概念 - 多态

【难易程度】 70

【题目描述】

请阅读下述代码，写出程序执行的结果。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | #include <iostream>  using namespace std;  class CBase  {  public:  virtual void print()  {  cout<< "base" << endl;  }  void DoPrint()  {  print();  }  };  class CChild1: public CBase  {  public:  virtual void print()  {  cout<< "child1" << endl;  }  };  class CChild2: public CBase  {  public:  virtual void print()  {  cout<< "child2" << endl;  }  };  void DoPrint(CBase \*base)  {  base->DoPrint();  }  int main()  {  CBase\* base = new CBase();  CChild1\* child1 = new CChild1();  CChild2\* child2 = new CChild2();  DoPrint(child1);  DoPrint(child2);  DoPrint(base);  delete base;  base = child1;  base->print();  delete child1;  delete child2;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4 | child1  child2  base  child1 |

【解释】

这是C++中类的虚函数实现多态的一个实例。

【题目编号】 CSD-07-702-4006

【知 识 点】 C++继承和多态 - 多态的概念 - 多态概念

【难易程度】 60

【题目描述】

多态是什么？

【正确答案】

多态、封装和继承是面向对象的三大特征。而多态性的定义是：同一操作作用于不同的对象，产生不同的执行结果。有两种类型的多态性：（1）编译时的多态性。编译时的多态性是通过重载来实现的。对于非虚成员来说，系统在编译时，根据传递的参数、返回的类型等信息决定实现何种操作。（2）运行时的多态性。运行时的多态性是指直到系统运行时，才根据实际情况决定实现何种操作。C++中，运行时的多态性通过虚成员实现。

【解释】 无

【题目编号】 CSD-07-702-4007

【知 识 点】 C++继承和多态 - 多态的概念 - 虚函数

【难易程度】 70

【题目描述】

虚函数是怎么实现的？

【正确答案】

简单地说，虚函数是通过虚函数表实现的。事实上，如果一个类中含有虚函数，则系统会为这个类分配一个指针成员指向一张虚函数表（vtbl)，表中每一项指向一个虚函数的地址，实现上就是一个函数指针的数组。

【解释】 无

【题目编号】 CSD-07-702-4008

【知 识 点】 C++继承和多态 - 多态的概念 - C++类虚拟机制的理解

【难易程度】 60

【题目描述】

分析代码写输出——虚函数的作用。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | #include <iostream>  using namespace std;  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 A  {  public:  void print(void)  {  cout << "C::print()" << endl;  }  };  void print(A a)  {  a.print();  }  int 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);  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | A::print()  B::print()  C::print()  A::print()  B::print()  C::print()  A::print()  A::print()  A::print() |

【解释】 无

【题目编号】 CSD-07-702-4009

【知 识 点】 C++继承和多态 - 多态的概念 - C++类虚拟机制的理解

【难易程度】 60

【题目描述】

分析代码写结果——虚函数。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | #include <iostream>  #include <string>  using namespace std;  void print(const std::string &msg)  {  cout << msg << endl;;  }  class Base  {  public:  Base()  {  print("Base::Base()");  virt();  }  void f()  {  print("Base::f()");  virt();  }  virtual void virt()  {  print("Base::virt()");  }  };  class Derived : public Base  {  public:  Derived()  {  print("Derived::Derived()");  virt();  }  virtual void virt()  {  print("Derived::virt()");  }  };  int main(int argc, char \*argv[])  {  Derived d;  Base \*pB = &d;  pB->f();  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Base::Base()  Base::virt()  Derived::Derived()  Derived::virt()  Base::f()  Derived::virt() |

【解释】 无

【题目编号】 CSD-07-702-4010

【知 识 点】 C++继承和多态 - 多态的概念 - C++多重继承的理解

【难易程度】 70

【题目描述】

分析代码找错——多重继承中的二义性。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | #include <iostream>  class cat  {  public:  void show()  {  std::cout << "cat" << std::endl;  }  };  class fish  {  public:  void show()  {  std::cout << "fish" << std::endl;  }  };  class catfish : public cat, public fish  {  };  int main()  {  catfish obj;  obj.show(); // error -> obj.cat::show();  return 0;  } |

【正确答案】

代码第27行出现编译错误是因为obj.show()无法区分执行哪个基类的show()方法。可以改成obj.car::show()访问cat的show()成员。

【解释】

程序中catfish类多重继承cat类和fish类，因此继承了cat的show(）方法和fish的show()方法。由于这两个方法同名，代码第27行直接用obj.show()时，无法区别执行哪个基类的show()方法，因此会出现编译错误。

【题目编号】 CSD-07-702-4011

【知 识 点】 C++继承和多态 - 多态的概念 - 多重继承和虚拟继承的理解

【难易程度】 70

【题目描述】

下面的代码输出结果是什么？如果类Child1和Child2都改为虚拟继承Parent，输出结果又是什么？

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | #include <iostream>  using namespace std;  class Parent  {  public:  Parent() : num(0) { cout << "Parent" << endl; }  Parent(int n) : num(n) { cout << "Parent(int)" << endl; }  private:  int num;  };  class Child1 : public Parent  {  public:  Child1() { cout << "Child1()" << endl; }  Child1(int num) : Parent(num) { cout << "Child1(int)" << endl; }  };  class Child2 : public Parent  {  public:  Child2() { cout << "Child2()" << endl; }  Child2(int num) : Parent(num) { cout << "Child2(int)" << endl; }  };  class Derived : public Child1, public Child2  {  public:  Derived() : Child1(0), Child2(1) {}  Derived(int num) : Child1(num), Child2(num + 1) {}  };  int main()  {  Derived d(4);  return 0;  } |

【正确答案】

|  |  |
| --- | --- |
|  | 不存在虚继承时的输出结果。 |
| 1  2  3  4 | Parent(int)  Child1(int)  Parent(int)  Child2(int) |
|  | 存在虚继承时的输出结果。即Child1和Child2虚继承自Parent类。 |
| 1  2  3 | Parent  Child1(int)  Child2(int) |

【解释】

多继承中的构造函数顺序如下：

（1）任何虚拟基类的构造函数按照它们被继承的顺序构造。

（2）任何非虚拟基类的构造函数按照它们被构造的顺序构造。

（3）任何成员对象的构造函数按照它们声明的顺序调用。

（4）类自身的构造函数。

## 7.3 多重继承和虚拟继承

【题目编号】 CSD-07-703-1001

【知 识 点】 C++继承和多态 - 多重继承和虚拟继承 - 多重继承

【难易程度】 60

【题目描述】

对于多重继承，有（ ）。

【选 项】

A、一个派生类只能有一个基类

B、一个基类只能产生一个派生类

C、一个基类必须产生多个派生类

D、一个派生类可有多个基类

【正确答案】 D

【解释】 无

【题目编号】 CSD-07-703-1002

【知 识 点】 C++继承和多态 - 多重继承和虚拟继承 - 虚基类

【难易程度】 60

【题目描述】

下列虚基类的声明中正确的是（ ）。

【选 项】

A．class virtual B:public A B．virtual class B:public

C．class B:public A virtual D．class B: virtual public A

【正确答案】 D

【解释】 无

【题目编号】 CSD-07-703-1003

【知 识 点】 C++继承和多态 - 多重继承和虚拟继承 - 多继承二义性

【难易程度】 60

【题目描述】

关于多继承二义性的描述中，错误的是（ ）。

【选 项】

A、一个派生类的基类中都有某个同名成员，在派生类中对这个成员的访问可能出现二义性

B、解决二义性的最常用的方法是对成员名的限定法

C、基类和派生类同时出现的同名函数，也存在二义性问题

D、一个派生类是从两个基类派生出来的，而这两个基类又有一个共同的基类，对该基类成员进行访问时，可能出现二义性

【正确答案】 C

【解释】 无

【题目编号】 CSD-07-703-1004

【知 识 点】 C++继承和多态 - 多重继承和虚拟继承 - 多重继承

【难易程度】 60

【题目描述】

在多继承中，公有私生和私有派生对于基类成员的派生类中的可访问性与单继承的规则（ ）。

【选 项】

A、完全相同

B、完全不同

C、部分相同，部分不同

D、以上都不对

【正确答案】 A

【解释】 无

【题目编号】 CSD-07-703-4001

【知 识 点】 C++继承和多态 - 多重继承和虚拟继承 - 抽象基类和虚函数

【难易程度】 80

【题目描述】

为什么要引入抽象基类和纯虚函数？

【正确答案】

纯虚函数在基类中没有定义，必须在子类中加以实现。如果基类含有一个或多个纯虚函数，那么它就属于抽象基类，不能被实例化。

引入抽象基类和纯虚函数的原因有以下两点。（1）为了方便使用多态特性。（2）在很多情况下，基类本身生成对象是不合情理的。例如，动物作为一个基类可以派生出老虎、狮子等子类，但动物本身生成对象明显不合常理。抽象基类不能够被实例化，它定义的纯虚函数相当于接口，能把派生类的共同行为提取出来。

【解释】 无

【题目编号】 CSD-07-703-4002

【知 识 点】 C++继承和多态 - 多重继承和虚拟继承 - 虚函数和纯虚函数区别

【难易程度】 70

【题目描述】

虚函数与纯虚函数有什么区别？

【正确答案】

虚函数和纯虚函数有以下区别。（1）类里声明虚函数的作用是为了能让这个函数在它的子类里面被覆盖，这样编译器就可以使用后期绑定来达到多态了。纯虚函数只是一个接口，是个函数的声明而已，它要留到子类里去实现。（2）虚函数在子类里面也可以不重载，但是纯虚函数必须在子类去实现。通常，很多函数加上virtual修辞，虽然牺牲掉一些性能，但是增加了面向对象的多态性，可以阻止父类里的这个函数在子类里被修改实现。（3）虚函数的类用于“实作继承”，也就是说继承接口的同时也继承了父类的实现。当然大家也可以完成自己的实现。纯虚函数的类用于“介面继承”，即纯虚函数关注的是接口的统一性，实现由子类完成。（4）带纯虚函数的类叫虚基类。这种基类不能直接生成对象，只能被继承，并重写其虚函数后才能使用，这样的类也叫抽象类。

【解释】 无

## 7.4 纯虚函数和抽象基类

纯虚函数就是基类只定义了函数体，没有实现过程。如果基类含有一个或多个纯虚函数，那么它就属于抽象基类，不能被实例化。

【题目编号】 CSD-07-704-1001

【知 识 点】 C++继承和多态 - 纯虚函数和抽象基类 - 抽象类

【难易程度】 60

【题目描述】

关于抽象类，下列表述正确的是（ ）。

【选 项】

A、抽象类的成员函数中至少有一个没有实现的函数（即无函数体定义的函数）

B、派生类必须实现作为基类的抽象类中的纯虚函数

C、派生类不可能成为抽象类

D、抽象类不能用于定义对象

【正确答案】 D

【解释】 无

【题目编号】 CSD-07-704-1002

【知 识 点】 C++继承和多态 - 纯虚函数和抽象基类 - 纯虚函数

【难易程度】 60

【题目描述】

关于纯虚函数，下列表述正确的是（ ）。

【选 项】

A、纯虚函数是给出实现版本（即无函数体定义）的虚函数

B、纯虚函数的声明总是“=0”结束

C、派生类必须实现基类的纯虚函数

D、含有纯虚函数的类一定是派生类

【正确答案】 B

【解释】 无

【题目编号】 CSD-07-704-1003

【知 识 点】 C++继承和多态 - 纯虚函数和抽象基类 - 虚基类

【难易程度】 60

【题目描述】

带有虚基类的多层派生类构造函数的成员初始化列表中都要列出虚基类构造函数，这样将对虚基类的子对象初始化（ ）。

【选 项】

A．与虚基类下面的派生类个数有关 B．多次 C．二次 D．一次

【正确答案】 D

【解释】 无

【题目编号】 CSD-07-704-1004

【知 识 点】 C++继承和多态 - 纯虚函数和抽象基类 - 纯虚函数

【难易程度】 50

【题目描述】

下面哪个基类中的成员函数表示纯虚函数（ ）。

【选 项】

A． virtual void vf(int)

B．void vf(int)=0

C．virtual void vf( )=0

D．virtual void vf(int)

【正确答案】 C

【解释】 无

【题目编号】 CSD-07-704-1005

【知 识 点】 C++继承和多态 - 纯虚函数和抽象基类 - 虚基类

【难易程度】 60

【题目描述】

设置虚基类的目的是（ ）。

【选 项】

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

【解释】 无

【题目编号】 CSD-07-704-1006

【知 识 点】 C++继承和多态 - 纯虚函数和抽象基类 - 抽象类

【难易程度】 60

【题目描述】

如果一个类至少有一个纯虚函数，那么就称该类为（ ）。

【选 项】

A、抽象类

B、派生类

C、纯基类

D、以上都不对

【正确答案】 A

【解释】 无

【题目编号】 CSD-07-704-1007

【知 识 点】 C++继承和多态 - 纯虚函数和抽象基类 - 纯虚函数

【难易程度】 60

【题目描述】

（ ）是一个在基类中说明虚函数，它在该基类中没有定义，但要求任何派生类中的可访问性的规定是相同的。

【选 项】

A、纯虚函数

B、虚析构函数

C、虚构造函数

D、静态成员函数

【正确答案】 A

【解释】 无

【题目编号】 CSD-07-704-1008

【知 识 点】 C++继承和多态 - 纯虚函数和抽象基类 - 抽象类

【难易程度】 60

【题目描述】

下列描述中，（ ）是抽象类的特性。

【选 项】

A、可以说明虚函数

B、可以定义友元函数

C、可以进行构造函数重载

D、不能说明其对象

【正确答案】 D

【解释】 无

【题目编号】 CSD-07-704-1009

【知 识 点】 C++继承和多态 - 纯虚函数和抽象基类 - 纯虚函数和抽象类

【难易程度】 70

【题目描述】

下面关于纯虚函数和抽象类的描述中，（ ）是错误的。

【选 项】

A、纯虚函数是一种特殊的虚函数，它没有具体意义

B、一个基类中说明有纯虚函数，该基类的派生类一定不再是抽象类

C、抽象类只能作为基类来使用，其纯虚函数的定义由派生类给出

D、抽象类是指具有纯虚函数的类

【正确答案】 B

【解释】 无

【题目编号】 CSD-07-704-1010

【知 识 点】 C++继承和多态 - 纯虚函数和抽象基类 - 虚函数的调用

【难易程度】 70

【题目描述】

对虚函数的调用（ ）。

【选 项】

A、一定使用动态联编

B、一定使用静态联编

C、必须使用动态联编

D、不一定使用动态联编

【正确答案】 D

【解释】 无

# 第8篇 数据结构

数据结构主要研究数据的组织方式以及相应的操作方法。它除了描述数据本身之外，还描述数据之间的相互关系。它不仅是一般程序设计的基础，而且是设计编译程序、操作系统、数据库、人工智能及其他大型应用程序的基础。如今数据结构在计算机科学中占有重要的地位，对于相当多的程序设计来说，认清数据的内在关系，可获得对问题的正确认识，看清问题的结构甚至解法。在一定意义上，程序所描述的就是在数据结构上实现的算法。算法的设计依赖于数据的逻辑结构，算法的实现依赖于数据的存储结构，所以数据结构选择的好坏，对程序的质量影响甚大。掌握基本的数据结构知识，是程序设计水平提高的必要条件。

## 8.1 单链表

【题目编号】 CSD-08-801-4001

【知 识 点】 数据结构 - 单链表 - 单链表的建立

【难易程度】 70

【题目描述】

编程实现单链表的建立。

【正确答案】

|  |  |
| --- | --- |
|  | 单链表结点定义 |
| 1  2  3  4  5 | typedef struct node  {  int data;  struct node \*next;  } node; |
|  | 单链表的创建 |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | node\* create()  {  int i = 0; // 链表中数据的个数  node \*head, \*p, \*q;  int x = 0;  head = (node\*)malloc(sizeof(node)); // 创建头节点    while(1)  {  printf("Please input the data: ");  scanf("%d", &x);  if(x == 0) // data为0时创建结束  break;  p = (node\*)malloc(sizeof(node));  p->data = x;  if(++i == 1)  { // 链表只有一个元素  head->next = p; // 连接到head的后面  }  else  {  q->next = p; //连接到链表尾端  }  q = p; // q指向末节点  }  q->next = NULL; //链表的最后一个指针为NULL  return head;  } |

【解释】 无

【题目编号】 CSD-08-801-4002

【知 识 点】 数据结构 - 单链表 - 单链表的测长

【难易程度】 70

【题目描述】

编程实现单链表的测长。

【正确答案】

|  |  |
| --- | --- |
|  | 返回单链表长度。 |
| 1  2  3  4  5  6  7  8  9  10  11  12 | int length(node \*head)  {  int len = 0;  node \*p;  p = head->next;  while(p != NULL) //遍历链表  {  len++;  p = p->next;  }  return len;  } |

【解释】 无

【题目编号】 CSD-08-801-4003

【知 识 点】 数据结构 - 单链表 - 单链表的打印

【难易程度】 70

【题目描述】

编程实现单链表的打印。

【正确答案】

|  |  |
| --- | --- |
|  | 打印单链表。 |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | void print(node \*head)  {  node \*p; int index = 0;  if(head->next == NULL) //链表为空  {  printf("Link is empty!\n");  return;  }  p = head->next;  while(p != NULL) //遍历链表  {  printf("The %dth node is: %d\n", ++index, p->data);  p = p->next;  }  } |

【解释】 无

【题目编号】 CSD-08-801-4004

【知 识 点】 数据结构 - 单链表 - 单链表的节点查找

【难易程度】 70

【题目描述】

编程实现单链表的节点查找。

【正确答案】

|  |  |
| --- | --- |
|  | 查找单链表pos位置的节点，返回节点指针。 |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | //pos从0开始，0返回head节点  node\* search\_node(node \*head, int pos)  {  node \*p = head->next;  if(pos < 0) // pos位置无效  {  printf("incorrect position to search node!\n");  return NULL;  }  if(pos == 0) // 在head位置，返回head  {  return head;  }  if( p == NULL)  {  printf("Link is empty!\n"); // 链表为空  return NULL;  }  while(--pos)  {  if((p = p->next) == NULL)  {  printf("incorrect position to search node!\n");  break; // 超出链表返回  }  }  return p;  } |

【解释】 无

【题目编号】 CSD-08-801-4005

【知 识 点】 数据结构 - 单链表 - 单链表的节点插入

【难易程度】 70

【题目描述】

编程实现单链表的节点插入。

【正确答案】

|  |  |
| --- | --- |
|  | 在单链表pos位置处插入节点，返回链表头指针。 |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | //pos从0开始计算，0表示插入到head节点后面  node\* insert\_node(node \*head, int pos, int data)  {  node \*item = NULL;  node \*p;  item = (node\*)malloc(sizeof(node));  item->data = data;  if(pos == 0) // 插入链表头后面  {  head->next = item;  return head;  }  p = search\_node(head, pos); // 获得位置pos的节点指针  if(p != NULL)  {  item->next = p->next;  p->next = item;  }  return head;  } |

【解释】 无

【题目编号】 CSD-08-801-4006

【知 识 点】 数据结构 - 单链表 - 单链表的节点删除

【难易程度】 70

【题目描述】

编程实现单链表的节点删除。

【正确答案】

|  |  |
| --- | --- |
|  | 删除单链表的pos位置的节点，返回链表头指针。 |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | //pos从1开始计算，1表示删除head后的第一个节点  node\* delete\_node(node \*head, int pos)  {  node \*item = NULL;  node \*p = head->next;  if(p == NULL) // 链表为空  {  printf("Link is empty!\n");  return NULL;  }  p = search\_node(head, pos - 1); // 获得位置pos的节点指针  if(p != NULL && p->next != NULL)  {  item = p->next;  p->next = item->next;  delete item;  }  return head;  } |

【解释】 无

【题目编号】 CSD-08-801-4007

【知 识 点】 数据结构 - 单链表 - 单链表的逆置

【难易程度】 80

【题目描述】

编程实现单链表的逆置。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | node\* reverse(node \*head)  {  node \*p, \*q, \*r;  if(head->next == NULL) // 链表为空  {  return head;  }  p = head->next;  q = p->next; // 保存源第二个节点  p->next = NULL; // 遍历，各个节点的next指针反转  while(q != NULL)  {  r = q->next;  q->next = p;  p = q;  q = r;  }  head->next = p; // 新的第一个节点为原末节点  return head;  } |

【解释】 无

【题目编号】 CSD-08-801-4008

【知 识 点】 数据结构 - 单链表 - 单链表的中间元素

【难易程度】 70

【题目描述】

寻找单链表的中间元素。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | node\* search\_midnode(node \*head)  {  int i = 0, j = 0;  node \*current = NULL;  node \*middle = NULL;    current = middle = head->next;  while(current != NULL)  {  if(i / 2 > j)  {  j++;  middle = middle->next;  }  i++;  current = current->next;  }  return middle;  } |

【解释】 无

【题目编号】 CSD-08-801-4009

【知 识 点】 数据结构 - 单链表 - 单链表的正向排序

【难易程度】 80

【题目描述】

单链表的正向排序。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | node\* insert\_sort(void)  {  int data = 0;  node \*head = NULL, \*New, \*Cur, \*Pre;  while(1)  {  printf("please input the data: \n");  scanf("%d", &data);  if(data == 0) break;  New = (node\*)malloc(sizeof(node));  New->data = data; // 新分配一个node节点  New->next = NULL;  if(head == NULL) // 第一次循环时对头节点赋值  {  head = New;  continue;  }  if(New->data <= head->data)  {  New->next = head;  head = New;  continue;  }  Cur = head;  while(New->data > Cur->data && Cur->next != NULL)  { // 找到需要插入的位置  Pre = Cur;  Cur = Cur->next;  }  if(Cur->data >= New->data) // 位置在中间  {  Pre->next = New; // 把New节点插入到Pre和Cur之间  New->next = Cur;  }  else  {  Cur->next = New;  }  }  return head;  } |

【解释】 无

【题目编号】 CSD-08-801-4010

【知 识 点】 数据结构 - 单链表 - 单链表是否存在环形链表

【难易程度】 80

【题目描述】

判断单链表是否存在环形链表问题。

【正确答案】

|  |  |
| --- | --- |
|  | 判断是否存在环，如果存在，start存放回环开始的节点。 |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | bool is\_loop(node \*head, node \*\*start)  {  node \*p1 = head, \*p2 = head;    if(head == NULL || head->next = NULL)  {  return false; // head为空或链表为空时返回false  }  do {  p1 = p1->next; // p1走一步  p2 = p2->next->next; // p2走两步  } while(p2 && p2->next && p1 != p2);    if(p1 == p2)  {  \*start = p1; // p1为回环开始节点  return true;  }  else  {  return false;  }  } |

【解释】

这里有一种比较简单的方法。假设两个指针分别为p1和p2。每循环一次p1向前走一步，p2向前走两步，直到p2碰到NULL指针或者两个指针相等时循环结束。如果两个指针相等则说明存在环。

【题目编号】 CSD-08-801-4011

【知 识 点】 数据结构 - 单链表 - 有序单链表的合并

【难易程度】 80

【题目描述】

有序单链表的合并。

【正确答案】

|  |  |
| --- | --- |
|  | 有序节点的插入 |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | node\* insert\_node(node \*head, node \*item) // head != NULL  {  node \*p = head;  node \*q = NULL; // 始终指向p之前的节点  while(p->data < item->data && p != NULL)  {  q = p;  p = p->next;  }  if(p == head) // 插入到原头节点之前  {  item->next = p;  return item;  }  // 插入到p与q之间  q->next = item;  item->next = p;  return head;  } |
|  | 非递归方法 |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | node\* merge(node \*head1, node \*head2)  {  node \*head; // 合并后的头指针  node \*p;  node \*nextP; // 指向p之后    if(head1 == NULL) // 有一个链表为空，直接返回另一个链表  {  return head2;  }  else if(head2 == NULL)  {  return head1;  }    // 两个链表都不为空  if(length(head1) >= length(head2))  { // 选取较短的链表，这样进行的插入次数要少些  head = head1;  p = head2;  }  else  {  head = head2;  p = head1;  }    while(p != NULL)  {  nextP = p->next; // 保存p的下一个节点  head = insert\_node(head, p); // 把p插入到目标链表中  p = nextP; // 指向将要插入的下一个节点  }  return head;  } |
|  | 递归方法 |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | node\* merge\_recursive(node \*head1, node \*head2)  {  node \*head = NULL;  if(head1 == NULL)  {  return head2;  }  if(head2 == NULL)  {  return head1;  }  if(head1->data < head2->data)  {  head = head1;  head->next = merge\_recursive(head1->next, head2);  }  else  {  head = head2;  head->next = merge\_recursive(head1, head2->next);  }  return head;  } |

【解释】

递归方法的步骤如下。

（1）比较链表1和链表2的第1个节点数据，由于1<2，因此把结果链表头节点指向链表1中的第1个节点，即数据1所在的节点。

（2）对剩余的链表1（3->5）和链表2再调用本过程，比较得到结果链表的第2个节点，即2与3比较得到2。此时合并后的链表节点为1->2。如此递归直到两个链表的节点都被加到结果链表中。

## 8.2 循环链表

循环链表与单链表一样，是一种链式的存储结构。所不同的是，循环链表的最后一个节点的指针指向该循环链表的第1个节点或者表头节点，从而构成一个环形的链。循环链表的运算与单链表的运算基本一致。所不同的有以下几点。

（1）在建立一个循环链表时，必须使其最后一个节点的指针指向表头节点，而不是像单链表那样置为NULL。此种情况适用于在最后一个节点后插入一个新的节点。

（2）判断是否到表尾采用判断该节点链域的值是否是表头节点的方法，当链域值等于表头指针时，说明已到表尾，而不是像单链表那样判断链域值是否为NULL。

【题目编号】 CSD-08-802-4001

【知 识 点】 数据结构 - 循环链表 - 约瑟夫问题

【难易程度】 80

【题目描述】

编一号为1~N的N个人按顺时针方向围坐一圈，梅人持有一个密码（正整数），开始任选一个正整数作为报数上限值M，从第1个人按顺时针方向自1开始顺序报数，报到M时停止报数。报M的人出列，将他的密码作为新的M值，从他顺时针方向上的下一个人开始从1报数，如此下去，直至所有人全部出列为止。试设计一个程序求出出列顺序。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71 | #include <stdio.h>  #include <stdlib.h>  typedef struct node{  int data;  struct node \*next;  } node;  node\* node\_create(int n)  {  node \*pRet = NULL;  if(0 != n)  {  int n\_idx = 1;  node \*p\_node = NULL;  // 创建n个node  p\_node = (node\*)malloc(sizeof(node) \* n);  if(NULL == p\_node) // 申请内存失败，返回NULL  {  return NULL;  }  else  {  memset(p\_node, 0, n \* sizeof(node)); // 初始化内存  }  pRet = p\_node;  while(n\_idx < n) // 初始化循环链表  {  p\_node->data = n\_idx;  p\_node->next = p\_node + 1;  p\_node = p\_node->next;  n\_idx++;  }  p\_node->data = n;  p\_node->next = pRet;  }  return pRet;  }  int main()  {  node \*pList = NULL;  node \*pIter = NULL;  int n = 20;  int m = 6;  pList = node\_create(n);  // 约瑟夫 循环取数  pIter = pList;  m %= n;  while(pIter != pIter->next)  {  int i = 1;  // 取到第m-1个节点  for(; i < m - 1; i++)  {  pIter = pIter->next;  }  // 输出第m个节点的值  printf("%d ", pIter->next->data);  // 从链表中删除第m个节点  pIter->next = pIter->next->next;  pIter = pIter->next;  }  printf("%d\n", pIter->data);  //释放申请的空间  free(pList);  return 0;  } |

【解释】

显然当有人退出圆圈后，报数的工作要从下一个人开始继续，而其剩的人仍然围成一个圆圈，因此可以使用循环单链表。退出圆圈的工作对应着表中节点的删除操作，对于这种删除操作频繁的情况，选用效率较高的链表结构，为了程序指针每一次都指向一个具体的代表一个人的节点而不需要判断，链表不带头节点。因此，对于所有人围成的圆圈所对应的数据结构采用一个不带头节点的循环链表来描述。设头指针为p，并根据具体情况移动。为了记录退出的人的先后顺序，采用一个顺序表进行存储。程序结束后再输出依次退出人的编号顺序。由于只记录各个节点的数据值就可以，所以定义一个整型一维数组。如：在int quite[int]中n为根据实际问题定义的足够大的整数。

## 8.3 双向链表

【题目编号】 CSD-08-803-4001

【知 识 点】 数据结构 - 双向链表 - 创建双向链表

【难易程度】 70

【题目描述】

建立一个双向链表。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | // 双向链表结构定义  typedef struct DbNode  {  int data; // 节点数据  struct DbNode \*left; // 前驱节点指针  struct DbNode \*right; // 后继节点指针  } DbNode;  // 创建一个新节点  DbNode\* CreateNode(int data)  {  DbNode \*pnode = (DbNode\*)malloc(sizeof(DbNode));  pnode->data = data;  // 创建新节点时，让前驱和后继都指向自身  pnode->left = pnode->right = NULL;  return pnode;  }  // 创建链表表头，返回表头节点  // 参数给出表头节点数据，表头节点不作为存放有意义数据的节点  DbNode\* CreateList(int head)  {  DbNode \*pnode = (DbNode\*)malloc(sizeof(DbNode));  pnode->data = head;  pnode->left = pnode->right = NULL;  return pnode;  }  // 在链表尾部插入新节点，返回表头节点  DbNode \*AppendNode(DbNode \*head, int data)  {  DbNode \*node = CreateNode(data); // 创建数据为data的新节点  DbNode \*p = head, \*q;  while(p != NULL)  {  q = p;  p = p->right;  }  q->right = node;  node->left = q;  return head;  }  // 生成从节点0~9的循环链表  DbNode \*head = CreateList(0); // 生成表头，表头数据为0  for(int i = 1; i < 10; i++)  {  head = AppendNode(head, i); // 添加9个节点  } |

【解释】

【题目编号】 CSD-08-803-4002

【知 识 点】 数据结构 - 双向链表 - 双向链表的测长

【难易程度】 70

【题目描述】

双向链表的测长。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | int GetLength(DbNode \*head) // 参数为链表的表头节点  {  int count = 1;  DbNode \*pnode = NULL;  if(head == NULL) // head为NULL表示链表空  return 0;  pnode = head->right;  while(pnode != NULL)  {  pnode = pnode->right; // 使用right指针遍历  count++;  }  return count;  } |

【解释】

为了得到双向链表的长度，需要使用right指针进行遍历，直到得到NULL为止。

【题目编号】 CSD-08-803-4003

【知 识 点】 数据结构 - 双向链表 - 双向链表的打印

【难易程度】 70

【题目描述】

双向链表的打印。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | void PrintList(DbNode \*head)  {  DbNode \*pnode = NULL;  if(head == NULL)  return;  pnode = head;  while(pnode != NULL)  {  printf("%d ", pnode->data);  pnode = pnode->right;  }  } |

【解释】 无

【题目编号】 CSD-08-803-4004

【知 识 点】 数据结构 - 双向链表 - 双向链表的节点查找

【难易程度】 70

【题目描述】

双向链表的节点查找。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | // 查找节点，成功返回满足条件的节点指针，否则返回NULL  DbNode\* FindNode(DbNode \*head, int data)  {  DbNode \*pnode = head;  if(head == NULL)  return NULL;  // 找到数据或者到达链表末尾退出while循环  while(pnode->right != NULL && pnode->data != data)  {  pnode = pnode->right;  }  // 没有找到数据为data的节点，返回NULL  if(pnode->right == NULL)  return NULL;  return pnode;  } |

【解释】 无

【题目编号】 CSD-08-803-4005

【知 识 点】 数据结构 - 双向链表 - 双向链表的节点插入

【难易程度】 70

【题目描述】

双向链表的节点插入。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | // 在node节点之后插入新节点  void InsertNode(DbNode \*node, int data)  {  DbNode \*newNode = CreateNode(data);  DbNode \*p = NULL;  if(node == NULL) // node为NULL时返回NULL  return NULL;  if(node->right == NULL) // node为最后一个节点  {  node->right = newNode;  newNode->left = node;  }  else // node为中间节点  {  newNode->right = node->right; // newNode向右连接  node->right->left = newNode;  node->right = newNode; // newNode向左连接  newNode->left = node;  }  } |

【解释】 无

【题目编号】 CSD-08-803-4006

【知 识 点】 数据结构 - 双向链表 - 双向链表的节点删除

【难易程度】 70

【题目描述】

双向链表的节点删除。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | // 删除满足指定条件的节点，返回表头节点，删除失败返回NULL  DbNode\* DeleteNode(DbNode \*head, int data)  {  DbNode \*ptmp = NULL;  DbNode \*pnode = FindNode(head, data); // 查找节点  if(NULL == pnode)  return NULL; // 节点不存在，返回NULL  else if(pnode->left == NULL) // node为第一个节点  {  head = pnode->right;  if(head != NULL) // 链表不为空  head->left = NULL;  }  else if(pnode->right == NULL) // node为最后一个节点  pnode->left->right = NULL;  else  {  pnode->left->right = pnode->right;  pnode->right->left = pnode->left;  }  free(pnode);  return head;  } |

【解释】 无

## 8.4 队列

【题目编号】 CSD-08-804-4001

【知 识 点】 数据结构 - 队列 - 队列的入队、出队、测长和打印

【难易程度】 70

【题目描述】

编程实现队列的入队、出队、测长和打印。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120 | #include <stdio.h>  typedef struct Node{  int data;  struct Node \*next;  }node;  typedef struct Queue  {  // node表示队列中的每个节点元素  node \*front; // 队头  node \*rear; // 队尾  }MyQueue;  //创建一个空的队列  MyQueue\* CreateMyQueue()  {  MyQueue \*q = (MyQueue\*)malloc(sizeof(MyQueue));  q->front = NULL; // 把队首指针置空  q->rear = NULL; // 把队尾指针置空  return q;  }  // 入队，从队尾一端插入节点  MyQueue\* enqueue(MyQueue \*q, int data)  {  node \*newP = NULL;  newP = (node\*)malloc(sizeof(node)); // 新建节点  newP->data = data;  newP->next = NULL;  if(q->rear == NULL)  {  // 若队列为空，新节点既是队首又是队尾  q->front = q->rear = newP;  }  else  {  // 若队列不为空，新节点放到队尾，队尾指针指向新节点  q->rear->next = newP;  q->rear = newP;  }  return q;  }  // 出队，从队头一端删除节点  MyQueue\* dequeue(MyQueue \*q)  {  node \*pnode = NULL;  pnode = q->front; // 指向队头  if(pnode == NULL) // 队列为空  {  printf("Empty queue!\n");  }  else  {  q->front = q->front->next; // 新队头  if(q->front == NULL) // 若删除后队列为空时，对rear置空  {  q->rear = NULL;  }  free(pnode);  }  return q;  }  // 队列的测长  int GetLength(MyQueue \*q)  {  int nlen = 0;  node \*pnode = q->front; // 指向队头  if(pnode != NULL) // 队列不为空  {  nlen = 1;  }  while(pnode != q->rear) // 遍历队列  {  pnode = pnode->next;  nlen++; // 循环一次nlen递增1  }  return nlen;  }  // 打印队列  void PrintMyQueue(MyQueue \*q)  {  node \*pnode = q->front;  if(pnode == NULL)  {  printf("Empty Queue!\n");  return;  }  printf("data: ");  while(pnode != q->rear) // 遍历队列  {  printf("%d ", pnode->data);  pnode = pnode->next;  }  printf("%d ", pnode->data);  }  int main()  {  int nlen = 0;  MyQueue \*hp = CreateMyQueue();  enqueue(hp, 1);  enqueue(hp, 2);  enqueue(hp, 3);  enqueue(hp, 4);  nlen = GetLength(hp);  printf("nlen = %d\n", nlen);  PrintMyQueue(hp);  dequeue(hp);  dequeue(hp);  nlen = GetLength(hp);  printf("\nnlen = %d\n", nlen);  PrintMyQueue(hp);  return 0;  } |

【解释】 无

## 8.5 栈

【题目编号】 CSD-08-805-4001

【知 识 点】 数据结构 - 栈 - 队列和栈的区别

【难易程度】 60

【题目描述】

队列和栈有什么区别？

【正确答案】

操作的名称不同。队列的插入称为入队，队列的删除称为出队。栈的插入称为进栈，栈的删除称为出栈。

可操作的方向不同。队列是在队尾入队，队头出队，即两边都可操作。而栈的进栈和出栈都是在栈顶进行的，无法对栈底直接进行操作。

操作的方法不同。队列是先进先出（FIFO)，即队列的修改是依据先进先出的原则进行的。新来的成员总是加入队尾（即不能中间插入），每次离开的成员总是队头上的成员（不允许中途离队）。而栈为后进先出（LIFO)，即每次删除（退栈）的总是当前栈中“最新的”元素，即最后插入（进栈）的元素，而最先插入的元素是被放在栈的底部，要到最后才被删除。

【解释】 无

【题目编号】 CSD-08-805-4002

【知 识 点】 数据结构 - 栈 - 队列实现栈

【难易程度】 70

【题目描述】

使用队列实现栈。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | #include <iostream>  using namespace std;  class MyData  {  public:  MyData() : data(0), next(NULL) {} // 默认构造函数  MyData(int value):data(value),next(NULL){} //带参数的构造函数  int data; // 数据域  MyData \*next; // 下一个节点  };  class MyStatic  {  public:  MyStatic() : top(NULL) {} // 默认构造函数  void push(MyData data); // 进栈  void pop(MyData \*pData); // 退栈  bool IsEmpty(); // 是否为空栈  MyData \*top; // 栈顶  };  //进栈  void MyStatic::push(MyData data)  {  MyData \*pData = NULL;  pData = new MyData(data.data); // 生成新节点  pData->next = top; // 与原来的栈顶节点相连  top = pData; // 栈顶节点为新加入的节点  }  // 出栈，返回栈顶节点内容  void MyStatic::pop(MyData \*data)  {  if(IsEmpty()) // 如果栈为空，则直接返回  {  return;  }  data->data = top->data; // 给传出的参数赋值  MyData \*p = top; // 临时保存原栈顶节点  top = top->next; // 移动栈顶，指向下一个节点  delete p; // 释放原栈顶节点内存  }  // 判断栈是否为空栈  bool MyStatic::IsEmpty()  {  return (top == NULL); // 如果top为空返回1，否则返回0  }  // 测试  int main()  {  MyData data(0); // 定义一个节点  MyStatic s; // 定义一个栈结构  s.push(MyData(1));  s.push(MyData(2));  s.push(MyData(3));  s.pop(&data); // 第1次退栈  cout << "pop" << data.data << endl;  s.pop(&data);  cout << "pop" << data.data << endl;  s.pop(&data);  cout << "pop" << data.data << endl;  cout << "Empty = " << s.IsEmpty() << endl; // 打印判空  return 0;  } |

【解释】 无

## 8.6 二叉树

【题目编号】 CSD-08-806-4001

【知 识 点】 数据结构 - 二叉树 - 二叉排序树的创建、插入、删除、查找

【难易程度】 70

【题目描述】

用C++实现一个二叉排序树，完成创建节点、插入节点、删除节点、查找节点等功能。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151 | // 有序二叉树  #include <iostream>  using namespace std;  // 有序二叉树类  class Tree {  public:  // 构造过程中初始化为空树  Tree (void) : m\_root (NULL), m\_size (0) {}  // 析构过程中释放剩余节点  ~Tree (void) {  clear ();  }  // 插入数据  void insert (int data) {  insert (new Node (data), m\_root);  m\_size++;  }  // 删除找到的第一个匹配数据，没找到返回false  bool remove (int data) {  Node\*& f = find (data, m\_root);  if (f) {  // 将f的左子树插入右子树  insert (f -> m\_left, f -> m\_right);  // 将f的右子树提升到f处  Node\* node = f;  f = f -> m\_right;  // 删除f  delete node;  m\_size--;  return true;  }  return false;  }  // 删除全部匹配数据  void removeall (int data) {  while (remove (data));  }  // 清空树  void clear (void) {  clear (m\_root);  m\_size = 0;  }  // 将所有的old替换为val  void update (int old, int val) {  while (remove (old))  insert (val);  }  // 是否存在data  bool find (int data) {  return find (data, m\_root) != NULL;  }  // 中序遍历  void travel (void) {  cout << '{';  travel (m\_root);  cout << '}' << endl;  }  // 获取树大小  size\_t size (void) {  return m\_size;  }  // 获取树高度  size\_t height (void) {  return height (m\_root);  }  private:  // 节点类  class Node {  public:  Node (int data) : m\_data (data),  m\_left (NULL), m\_right (NULL) {}  int m\_data; // 数据  Node\* m\_left; // 左树  Node\* m\_right; // 右树  };  void insert (Node\* node, Node\*& tree) {  if (! tree)  tree = node;  else if (node) {  if (node -> m\_data < tree -> m\_data)  insert (node, tree -> m\_left);  else  insert (node, tree -> m\_right);  }  }  Node\*& find (int data, Node\*& tree) {  if (! tree)  return tree;  else  if (data == tree -> m\_data)  return tree;  else  if (data < tree -> m\_data)  return find (data, tree -> m\_left);  else  return find (data, tree -> m\_right);  }  void clear (Node\*& tree) {  if (tree) {  clear (tree -> m\_left);  clear (tree -> m\_right);  delete tree;  tree = NULL;  }  }  void travel (Node\*& tree) {  if (tree) {  travel (tree -> m\_left);  cout << '<' << tree -> m\_data << '>';  travel (tree -> m\_right);  }  }  size\_t height (Node\*& tree) {  if (tree) {  size\_t lh = height (tree -> m\_left);  size\_t rh = height (tree -> m\_right);  return (lh > rh ? lh : rh) + 1;  }  return 0;  }  Node\* m\_root; // 树根  size\_t m\_size; // 树大小  };  int main (void) {  Tree tree;  tree.insert (50);  tree.insert (70);  tree.insert (20);  tree.insert (60);  tree.insert (40);  tree.insert (30);  tree.insert (10);  tree.insert (90);  tree.insert (80);  tree.travel ();  cout << tree.height () << ',' << tree.size () << endl;  tree.remove (60);  tree.travel ();  tree.insert (30);  tree.insert (30);  tree.travel ();  tree.removeall (30);  tree.travel ();  tree.update (70, 15);  tree.travel ();  tree.clear ();  tree.travel ();  return 0;  } |

【解释】 无

# 第9篇 排序

## 9.1 插入排序

插入排序的基本思想是：每次将一个待排序的记录，按其关键字大小插入到前面已经排好序的子数组中的适当位置，直到全部记录插入完成为止。直接插入排序和希尔（Shell）排序都属于插入排序方法。

【题目编号】 CSD-09-901-4001

【知 识 点】 排序 - 插入排序 - 直接插入排序

【难易程度】 80

【题目描述】

编程实现直接插入排序。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | #include <iostream>  using namespace std;  // 直接插入排序  void insert\_sort(int a[], int n)  {  int i, j, temp;  for(i = 1; i < n; i++)  {  //暂存下标为i的数，下标从1开始，因为开始时下标为0的数，前面没有  // 任何数，此时认为它是排好顺序的  temp = a[i];  for(j = i - 1; j >= 0 && temp < a[j]; j--)  {  //如果满足条件就往后挪，最坏情况是temp比a[0]小，要放在最前面  a[j + 1] = a[j];  }  a[j + 1] = temp; // 找到下标为i的数的放置位置  }  }  void print\_array(int a[], int len)  {  for(int i = 0; i < len; i++)  {  cout << a[i] << " ";  }  cout << endl;  }  int main()  {  int a[] = {7, 3, 5, 8, 9, 1, 2, 4, 6};  cout << "before insert sort: ";  print\_array(a, 9);  insert\_sort(a, 9); // 进行直接插入排序  cout << "after insert sort: ";  print\_array(a, 9);  return 0;  } |

【解释】

直接插入排序的作法是：每次从无序表中取出第一个元素，把它插入到有序表的合适位置，使有序表仍然有序。第一趟比较前两个数,然后把第二个数按大小插入到有序表中；第二趟把第三个数据与前两个数从前向后扫描，把第三个数按大小插入到有序表中；依次进行下去，进行了(n-1)趟扫描以后就完成了整个排序过程。

直接插入排序属于稳定的排序，最坏时间复杂性为O(n^2)，空间复杂度为O(1)。

【题目编号】 CSD-09-901-4002

【知 识 点】 排序 - 插入排序 - 希尔排序

【难易程度】 80

【题目描述】

编程实现shell排序。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | #include <iostream>  using namespace std;  void shell\_sort(int a[], int len)  {  int h, i, j, temp;  for(h = len / 2; h > 0; h = h / 2)  {  for(i = h; i < len; i++) //这个for循环就是前面的直接插入排序  {  temp = a[i];  for(j = i - h; (j >= 0 && temp < a[j]); j -= h)  {  a[j + h] = a[j];  }  a[j + h] = temp;  }  }  }  void print\_array(int a[], int len)  {  for(int i = 0; i < len; i++)  {  cout << a[i] << " ";  }  cout <<endl;  }  int main()  {  int a[] = {7, 3, 5, 8, 9, 1, 2, 4, 6};  cout << "before shell sort: ";  print\_array(a, 9);  shell\_sort(a, 9);  cout << "after shell sort: ";  print\_array(a, 9);  return 0;  } |

【解释】

希尔排序是插入排序的一种。是针对直接插入排序算法的改进。该方法又称缩小增量排序，因DL．Shell于1959年提出而得名，属于不稳定排序方法。

在直接插入排序算法中，每次插入一个数，使有序序列只增加1个节点，并且对插入下一个数没有提供任何帮助。如果比较相隔较远距离（称为增量）的数，使得数移动时能跨过多个元素，则进行一次比较就可能消除多个元素交换。

希尔排序算法先将要排序的一组数按某个增量d分成若干组，对每组中全部元素进行排序，然后用一个较小的增量对它进行再次分组，并对每个新组进行排序。当增量减到1时，整个要排序的数被分成一组，排序完成。因此希尔排序实质上是一种分组插入方法。希尔排序的时间性能优于直接插入排序，其原因如下。

当数组初始状态基本有序时，直接插入排序所需的比较和移动次数均较少。

当n值较小时，n和n2的差别也较小，即直接插入排序的最好时间复杂度O(n）和最坏时间复杂度O(n2）差别不大。

在希尔排序开始时增量较大，分组较多，每组的记录数目少，故各组内直接插入较快，后来增量d逐渐缩小，分组数逐渐减少，而各组的记录数目逐渐增多，但由于已经按d-1作为距离完成排序，使数组较接近于有序状态，所以新的一次排序过程也较快。

因此，希尔排序在效率上比直接插入排序有较大的改进。

注意：由于分组的存在，相等的元素可能会分在不同组，导致它们的次序可能发生变化，因此希尔排序是不稳定的。

## 9.2 交换排序

【题目编号】 CSD-09-902-4001

【知 识 点】 排序 - 交换排序 - 冒泡排序

【难易程度】 80

【题目描述】

编程实现冒泡排序。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | #include <iostream>>  using namespace std;  int\* BubbleSort(int \*array, int len)  {  int length = len;  for (int i = 0; i <= length - 1; i++)  {  for (int j = length - 1; j > i; j--)  {  if (array[j] < array[j - 1])  {  int temp = array[j];  array[j] = array[j - 1];  array[j - 1] = temp;  }  }  }  return array;  }  int main()  {  int a[] = {2,5,8,1,3,9};  BubbleSort(a, 6);  for(int i = 0; i < 5; i++)  {  cout << a[i] << " ";  }  cout << endl;  return 0;  } |

【解释】

冒泡排序的基本概念是：依次比较相邻的两个数，将小数放在前面，大数放在后面。即在第一趟：首先比较第1个和第2个数，将小数放前，大数放后。然后比较第2个数和第3个数，将小数放前，大数放后，如此继续，直至比较最后两个数，将小数放前，大数放后。

至此第一趟结束，将最大的数放到了最后。在第二趟：仍从第一对数开始比较（因为可能由于第2个数和第3个数的交换，使得第1个数不再小于第2个数），将小数放前，大数放后，一直比较到倒数第二个数（倒数第一的位置上已经是最大的），第二趟结束，在倒数第二的位置上得到一个新的最大数（其实在整个数列中是第二大的数）。如此下去，重复以上过程，直至最终完成排序。

由于在排序过程中总是小数往前放，大数往后放，相当于气泡往上升，所以称作冒泡排序。

用二重循环实现，外循环变量设为i，内循环变量设为j。假如有10个数需要进行排序，则外循环重复9次，内循环依次重复9，8，...，1次。每次进行比较的两个元素都是与内循环j有关的，它们可以分别用a[j]和a[j+1]标识，i的值依次为1,2,...,9，对于每一个i,j的值依次为1,2,...10-i。

【题目编号】 CSD-09-902-4002

【知 识 点】 排序 - 交换排序 - 快速排序

【难易程度】 80

【题目描述】

编程实现快速排序。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | #include <iostream>>  using namespace std;  void quick\_sort(int s[], int l, int r)  {  if (l < r)  {  //Swap(s[l], s[(l + r) / 2]); //将中间的这个数和第一个数交换  int i = l, j = r, x = s[l];  while (i < j)  {  while(i < j && s[j] >= x) // 从右向左找第一个小于x的数  j--;  if(i < j)  s[i++] = s[j];  while(i < j && s[i] < x) // 从左向右找第一个大于等于x的数  i++;  if(i < j)  s[j--] = s[i];  }  s[i] = x;  quick\_sort(s, l, i - 1); // 递归调用  quick\_sort(s, i + 1, r);  }  }  int main()  {  int a[] = {2,5,8,1,3,9};  quick\_sort(a, 0, 5);  for(int i = 0; i < 5; i++)  {  cout << a[i] << " ";  }  cout << endl;  return 0;  } |

【解释】

快速排序是一种有效的排序算法。虽然算法在最坏的情况下运行时间为O(n^2)，但由于平均运行时间为O(nlogn)，并且在内存使用、程序实现复杂性上表现优秀，尤其是对快速排序算法进行随机化的可能，使得快速排序在一般情况下是最实用的排序方法之一。快速排序被认为是当前最优秀的内部排序方法。

它的基本思想是：通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

## 9.3 选择排序

【题目编号】 CSD-09-903-4001

【知 识 点】 排序 - 选择排序 - 直接选择排序

【难易程度】 80

【题目描述】

编程实现直接选择排序。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | #include <iostream>  using namespace std;  void Swap(int &a, int &b)  {  if (a != b)  {  a ^= b;  b ^= a;  a ^= b;  }  }  void Selectsort(int a[], int n)  {  int i, j, nMinIndex;  for (i = 0; i < n; i++)  {  nMinIndex = i; //找最小元素的位置  for (j = i + 1; j < n; j++)  if (a[j] < a[nMinIndex])  nMinIndex = j;  Swap(a[i], a[nMinIndex]); //将这个元素放到无序区的开头  }  }  int main()  {  int a[] = {3,2,1,5,7,4,6};  Selectsort(a, 7);  for(int i = 0; i < 7; i++)  {  cout << a[i] << " ";  }  cout << endl;  return 0;  } |

【解释】

直接选择排序也是一种简单的排序方法，它的基本思想是：第一次从R[0]~R[n-1]中选取最小值，与R[0]交换，第二次从R{1}~R[n-1]中选取最小值，与R[1]交换，....，第i次从R[i-1]~R[n-1]中选取最小值，与R[i-1]交换，.....，第n-1次从R[n-2]~R[n-1]中选取最小值，与R[n-2]交换,总共通过n-1次,得到一个按排序码从小到大排列的有序序列。

例如：给定n=8，数组R中的8个元素的排序码为(8,3,2,1,7,4,6,5),则直接选择排序的过程如下所示

初始状态 [ 8 3 2 1 7 4 6 5 ] 8 🡪 1

第一次 [ 1 3 2 8 7 4 6 5 ] 3 🡪 2

第二次 [ 1 2 3 8 7 4 6 5 ] 3 🡪 3

第三次 [ 1 2 3 8 7 4 6 5 ] 8 🡪 4

第四次 [ 1 2 3 4 7 8 6 5 ] 7 🡪 5

第五次 [ 1 2 3 4 5 8 6 7 ] 8 🡪 6

第六次 [ 1 2 3 4 5 6 8 7 ] 8 🡪 7

第七次 [ 1 2 3 4 5 6 7 8 ] 排序完成

【题目编号】 CSD-09-903-4002

【知 识 点】 排序 - 选择排序 - 堆排序

【难易程度】 80

【题目描述】

编程实现堆排序。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85 | #include <iostream>  using namespace std;  // 输出当前堆的排序状况  void PrintArray(int data[], int size)  {  for (int i=1; i<=size; ++i)  cout <<data[i]<<" ";  cout<<endl;  }  // 堆化，保持堆的性质  // MaxHeapify让a[i]在最大堆中"下降"，  // 使以i为根的子树成为最大堆  void MaxHeapify(int \*a, int i, int size)  {  int lt = 2\*i, rt = 2\*i+1;  int largest;  if(lt <= size && a[lt] > a[i])  largest = lt;  else  largest = i;  if(rt <= size && a[rt] > a[largest])  largest = rt;  if(largest != i)  {  int temp = a[i];  a[i] = a[largest];  a[largest] = temp;  MaxHeapify(a, largest, size);  }  }  // 建堆  // 自底而上地调用MaxHeapify来将一个数组a[1..size]变成一个最大堆  void BuildMaxHeap(int \*a, int size)  {  for(int i=size/2; i>=1; --i)  MaxHeapify(a, i, size);  }  // 堆排序  // 初始调用BuildMaxHeap将a[1..size]变成最大堆  // 因为数组最大元素在a[1]，则可以通过将a[1]与a[size]互换达到正确位置  // 现在新的根元素破坏了最大堆的性质，所以调用MaxHeapify调整，  // 使a[1..size-1]成为最大堆，a[1]又是a[1..size-1]中的最大元素，  // 将a[1]与a[size-1]互换达到正确位置。  // 反复调用Heapify，使整个数组成从小到大排序。  // 注意： 交换只是破坏了以a[1]为根的二叉树最大堆性质，它的左右子二叉树还是  // 具备最大堆性质。这也是为何在BuildMaxHeap时需要遍历size/2到1的结点才能  // 构成最大堆，而这里只需要堆化a[1]即可。  void HeapSort(int \*a, int size)  {  BuildMaxHeap(a, size);  PrintArray(a, size);  int len = size;  for(int i=size; i>=2; --i)  {  int temp = a[1];  a[1] = a[i];  a[i] = temp;  len--;  MaxHeapify(a, 1, len);  cout << "中间过程:";  PrintArray(a, size);  }  }  int main()  {  int size;  int arr[100];  cout << "Input the num of elements: ";  cin >> size;  cout << "Input the elements: ";  for(int i=1; i<=size; ++i)  cin >> arr[i];  cout << endl;  HeapSort(arr, size);  cout << "最后结果:";  PrintArray(arr, size);  } |

【解释】

堆排序是指利用堆这种数据结构所设计的一种排序算法。堆是一个近似完全二叉树的结构，并同时满足堆性质：即子结点的键值或索引总是小于（或者大于）它的父节点。

## 9.4 归并排序

【题目编号】 CSD-09-904-4001

【知 识 点】 排序 - 归并排序 - 归并排序

【难易程度】 80

【题目描述】

编程实现归并排序算法。

【正确答案】

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86 | #include <iostream>  #include <cstdlib>  using namespace std;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*函数名称：Merge  \*参数说明：pDataArray 无序数组；  \* int \*pTempArray 临时存储合并后的序列  \* bIndex 需要合并的序列1的起始位置  \* mIndex 需要合并的序列1的结束位置  并且作为序列2的起始位置  \* eIndex 需要合并的序列2的结束位置  \*说明： 将数组中连续的两个子序列合并为一个有序序列  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void Merge(int\* pDataArray, int \*pTempArray, int bIndex,  int mIndex, int eIndex)  {  int mLength = eIndex - bIndex; //合并后的序列长度  int i = 0; //记录合并后序列插入数据的偏移  int j = bIndex; //记录子序列1插入数据的偏移  int k = mIndex; //记录子序列2掺入数据的偏移  while (j < mIndex && k < eIndex)  {  if (pDataArray[j] <= pDataArray[k])  {  pTempArray[i++] = pDataArray[j];  j++;  }  else  {  pTempArray[i++] = pDataArray[k];  k++;  }  }  if (j == mIndex) //说明序列1已经插入完毕  while (k < eIndex)  pTempArray[i++] = pDataArray[k++];  else //说明序列2已经插入完毕  while (j < mIndex)  pTempArray[i++] = pDataArray[j++];  for (i = 0; i < mLength; i++) //将合并后序列重新放入pDataArray  pDataArray[bIndex + i] = pTempArray[i];  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*函数名称：BottomUpMergeSort  \*参数说明：pDataArray 无序数组；  \* iDataNum为无序数据个数  \*说明： 自底向上的归并排序  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void BottomUpMergeSort(int\* pDataArray, int iDataNum)  {  //临时存放合并后的序列  int \*pTempArray = (int \*)malloc(sizeof(int) \* iDataNum);  int length = 1; //初始有序子序列长度为1  while (length < iDataNum)  {  int i = 0;  for (; i + 2\*length < iDataNum; i += 2\*length)  Merge(pDataArray, pTempArray, i, i + length,  i + 2\*length);  if (i + length < iDataNum)  Merge(pDataArray, pTempArray, i, i + length, iDataNum);  length \*= 2; //有序子序列长度\*2  }  free(pTempArray);  }  int main()  {  int size;  int arr[] = {3, 2, 1, 89, 68, 100, 99};  BottomUpMergeSort(arr, 7);  cout << "最后结果是：";  for(int i = 0; i < 7; i++)  {  cout << arr[i] << " ";  }  cout << endl;  return 0;  } |

【解释】

归并排序是建立在归并操作上的一种有效的排序算法,该算法是采用分治法的一个非常典型的应用。

归并操作的工作原理如下：申请空间，使其大小为两个已经排序序列之和，该空间用来存放合并后的序列；设定两个指针，最初位置分别为两个已经排序序列的起始位置；比较两个指针所指向的元素，选择相对小的元素放入到合并空间，并移动指针到下一位置；重复步骤3直到某一指针达到序列尾；将另一序列剩下的所有元素直接复制到合并序列尾

## 9.5 各种排序方法比较

【题目编号】 CSD-09-905-1001

【知 识 点】 排序 - 各种排序方法比较 - 排序算法性能比较

【难易程度】 70

【题目描述】

下面哪种排序法对12354排序最快？

【选 项】

A. 快速排序法。 B. 冒泡排序法。 C. 归并排序法。

【正确答案】 B

【解释】

选择排序算法的时候，需要考虑以下几点：数据的规模；数据的类型；数据己有的顺序。

一般来说，当数据规模较小时，应选择直接插入排序法或冒泡排序法。任何排序算法在数据量小时基本体现不出来差距。考虑数据的类型，比如全部是正整数时，应该考虑使用桶排序法。

考虑数据己有顺序，快速排序是一种不稳定的排序（当然可以改进），对于大部分排好的数据，快速排序会浪费大量不必要的步骤。认为快速排序好，是指大量随机数据下，使用快速排序效果最理想。而不是指所有情况。

所以，根据题目分析，12354数据量极小，已经基本排好序。所以此时冒泡排序法是最佳选择。

【题目编号】 CSD-09-905-4001

【知 识 点】 排序 - 各种排序方法比较 - 各排序算法的时间复杂度比较

【难易程度】 70

【题目描述】

写出下列算法的时间复杂度。

冒泡排序法。

选择排序法。

插入排序法。

快速排序法。

堆排序法。

归并排序法。

【正确答案】

冒泡排序算法时间复杂度是O(n^2)。

选择排序算法复杂度是O(n^2)。

插入排序算法时间复杂度是O(n^2)。

快速排序法是不稳定的。最理想情况算法时间复杂度为O(nlog2n)，最坏为O(n^2)。

堆排序算法时间复杂度为O(nlogn)。

归并排序的时间复杂度是O(nlog2n)。

【解释】 无

# 第10篇 Oracle、Pro\*C、Pl/Sql

【题目编号】 CSD-10-1001-4001

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 触发器的作用

【难易程度】 60

【题目描述】

触发器的作用？

【正确答案】

触发器是一中特殊的存储过程，主要是通过事件来触发而被执行的。它可以强化约束，来维护数据的完整性和一致性，可以跟踪数据库内的操作从而不允许未经许可的更新和变化。可以联级运算。如，某表上的触发器上包含对另一个表的数据操作，而该操作又会导致该表触发器被触发。

触发器： 当满足触发器条件，则系统自动执行触发器的触发体。

触发时间：有before,after.触发事件：有insert,update,delete三种。触发类型：有行触发、语句触发

【解释】 无

【题目编号】 CSD-10-1001-4002

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 存储过程

【难易程度】 60

【题目描述】

什么是存储过程？用什么来调用？

【正确答案】

存储过程是一个预编译的SQL语句，优点是允许模块化的设计，就是说只需创建一次，以后在该程序中就可以调用多次。如果某次操作需要执行多次SQL，使用存储过程比单纯SQL语句执行要快。可以用一个命令对象来调用存储过程。

【解释】 无

【题目编号】 CSD-10-1001-4003

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 索引

【难易程度】 60

【题目描述】

索引的作用？它的优点缺点是什么？

【正确答案】

索引就一种特殊的查询表，数据库的搜索引擎可以利用它加速对数据的检索。它很类似与现实生活中书的目录，不需要查询整本书内容就可以找到想要的数据。索引可以是唯一的，创建索引允许指定单个列或者是多个列。缺点是它减慢了数据录入的速度，同时也增加了数据库的尺寸大小。

【解释】 无

【题目编号】 CSD-10-1001-4004

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 事务和锁

【难易程度】 60

【题目描述】

什么是事务？什么是锁？

【正确答案】

事务就是被绑定在一起作为一个逻辑工作单元的SQL语句分组，如果任何一个语句操作失败那么整个操作就被失败，以后操作就会回滚到操作前状态，或者是上有个节点。为了确保要么执行，要么不执行，就可以使用事务。要将有组语句作为事务考虑，就需要通过ACID测试，即原子性，一致性，隔离性和持久性。

锁：在所以的DBMS中，锁是实现事务的关键，锁可以保证事务的完整性和并发性。与现实生活中锁一样，它可以使某些数据的拥有者，在某段时间内不能使用某些数据或数据结构。当然锁还分级别的。

【解释】 无

【题目编号】 CSD-10-1001-4005

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 视图和游标

【难易程度】 60

【题目描述】

什么叫视图？游标是什么？

【正确答案】

视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，试图通常是有一个表或者多个表的行或列的子集。对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询。

游标：是对查询出来的结果集作为一个单元来有效的处理。游标可以定在该单元中的特定行，从结果集的当前行检索一行或多行。可以对结果集当前行做修改。一般不使用游标，但是需要逐条处理数据的时候，游标显得十分重要。

【解释】 无

【题目编号】 CSD-10-1001-4006

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 存储过程

【难易程度】 60

【题目描述】

存储过程：存储过程就是编译好了的一些sql语句。

【正确答案】

（1）存储过程因为SQL语句已经预编绎过了，因此运行的速度比较快。

（2）可保证数据的安全性和完整性。通过存储过程可以使没有权限的用户在控制之下间接地存取数据库，从而保证数据的安全。通过存储过程可以使相关的动作在一起发生，从而可以维护数据库的完整性。

（3）可以降低网络的通信量。存储过程主要是在服务器上运行，减少对客户机的压力。

（4）存储过程可以接受参数、输出参数、返回单个或多个结果集以及返回值。可以向程序返回错误原因。

（5）存储过程可以包含程序流、逻辑以及对数据库的查询。同时可以实体封装和隐藏了数据逻辑。

【解释】 无

【题目编号】 CSD-10-1001-4007

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 内联接和外联接区别

【难易程度】 60

【题目描述】

内联接,外联接区别？

【正确答案】

内连接是保证两个表中所有的行都要满足连接条件，而外连接则不然。在外连接中，某些不满条件的列也会显示出来，也就是说，只限制其中一个表的行，而不限制另一个表的行。分左连接、右连接、全连接三种。

【解释】 无

【题目编号】 CSD-10-1001-4008

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 表空间

【难易程度】 60

【题目描述】

TEMPORARY tablespace和PERMANENT tablespace 的区别是？

【正确答案】

temporary tablespace 用于临时对象例如排序结构而 permanent tablespaces用来存储那些'真实'的对象(例如表，回滚段等)。

【解释】 无

【题目编号】 CSD-10-1001-4009

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 表空间

【难易程度】 60

【题目描述】

创建数据库时自动建立的tablespace名称？

【正确答案】

SYSTEM tablespace

【解释】 无

【题目编号】 CSD-10-1001-4010

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 创建用户

【难易程度】 60

【题目描述】

创建用户时，需要赋予新用户什么权限才能使它联上数据库。

【正确答案】

CONNECT权限。

【解释】 无

【题目编号】 CSD-10-1001-4011

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 表空间

【难易程度】 60

【题目描述】

如何在tablespace里增加数据文件？

【正确答案】

ALTER TABLESPACE ADD DATAFILE SIZE

【解释】 无

【题目编号】 CSD-10-1001-4012

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 数据文件

【难易程度】 60

【题目描述】

如何变动数据文件的大小？

【正确答案】

ALTER DATABASE DATAFILE RESIZE

【解释】 无

【题目编号】 CSD-10-1001-4013

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 数据文件

【难易程度】 60

【题目描述】

哪个VIEW用来检查数据文件的大小？

【正确答案】

DBA\_DATA\_FILES

【解释】 无

【题目编号】 CSD-10-1001-4014

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 表空间

【难易程度】 60

【题目描述】

哪个VIEW用来判断tablespace的剩余空间？

【正确答案】

DBA\_FREE\_SPACE

【解释】 无

【题目编号】 CSD-10-1001-4015

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 索引

【难易程度】 60

【题目描述】

如何重构索引？

【正确答案】

ALTER INDEX REBUILD

【解释】 无

【题目编号】 CSD-10-1001-4016

【知 识 点】 Oracle、Pro\*C、Pl/Sql - 错误信息

【难易程度】 60

【题目描述】

刚刚编译了一个PL/SQL Package但是有错误报道，如何显示出错信息？

【正确答案】

SHOW ERRORS

【解释】 无

【题目编号】 CSD-10-1001-4017

【知 识 点】 Oracle、Pro\*C、Pl/Sql - oracle网络连接

【难易程度】 70

【题目描述】

oracle中，用于网络连接的2个文件？

【正确答案】

TNSNAMES.ORA and SQLNET.ORA

【解释】 无

# 第11篇 Win32、MFC

【题目编号】 CSD-11-1101-4001

【知 识 点】 Win32、MFC - 死锁的处理

【难易程度】 60

【题目描述】

死锁的处理。

【正确答案】

鸵鸟策略、预防策略、避免策略、检测与解除死锁。

【解释】 无

【题目编号】 CSD-11-1101-4002

【知 识 点】 Win32、MFC - TCP/IP 建立连接的过程

【难易程度】 70

【题目描述】

TCP/IP 建立连接的过程。

【正确答案】

在TCP/IP协议中，TCP协议提供可靠的连接服务，采用三次握手建立一个连接。

第一次握手：建立连接时，客户端发送连接请求到服务器，并进入SYN\_SEND状态，等待服务器确认；

第二次握手：服务器收到客户端连接请求，向客户端发送允许连接应答，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的允许连接应答，向服务器发送确认，客户端和服务器进入通信状态，完成三次握手。

【解释】 无

【题目编号】 CSD-11-1101-4003

【知 识 点】 Win32、MFC - CMemoryState主要功能

【难易程度】 60

【题目描述】

CMemoryState主要功能是什么？

【正确答案】

用于检测内存泄露。分配了内存而没有释放,逐渐耗尽内存资源，导致系统崩溃。内存泄露是指程序中间动态分配了内存，但是在程序结束时没有释放这部分内存，从而造成那一部分内存不可用的情况，重起计算机可以解决，但是也有可能再次发生内存泄露，内存泄露和硬件没有关系，它是由软件引起的。

【解释】 无

【题目编号】 CSD-11-1101-4004

【知 识 点】 Win32、MFC - 进程、线程创建

【难易程度】 60

【题目描述】

用什么函数开启新进程、线程？

【正确答案】

CreateProccess()创建进程；CreateThread()创建线程；MFC中还提供了\_beginthread()与\_beginthreadex()函数创建线程。

【解释】 无

【题目编号】 CSD-11-1101-4005

【知 识 点】 Win32、MFC - SendMessage和PostMessage的区别

【难易程度】 60

【题目描述】

SendMessage和PostMessage有什么区别？

【正确答案】

SendMessage：发送消息后，等待消息处理完毕后才继续执行自身的程序。PostMessage：发送消息后不等待消息处理即继续执行自身的程序。

【解释】 无

【题目编号】 CSD-11-1101-4006

【知 识 点】 Win32、MFC - WaitForSingleObject的作用

【难易程度】 60

【题目描述】

WaitForSingleObject 有何作用？

【正确答案】

WaitForSingleObject是表示等待线程的一个函数。参数为INFINITE表示一直等待线程CWinThread执行结束后，再继续处理自身程序。

【解释】 无

【题目编号】 CSD-11-1101-4007

【知 识 点】 Win32、MFC - \_\_stdcall、\_\_cdecl、\_\_pascal区别

【难易程度】 60

【题目描述】

\_\_stdcall、\_\_cdecl、\_\_pascal在什么方面有所不同？

【正确答案】

这些都是一些函数参数的调用约定，告诉编译器函数参数压栈的顺序，以及压入堆栈的内容由谁来清除，是调用者还是函数本身清除堆栈的内容。

【解释】 无

【题目编号】 CSD-11-1101-4008

【知 识 点】 Win32、MFC - 进程间通信

【难易程度】 60

【题目描述】

进程间通信的方式有？

【正确答案】

进程间通信的方式有 ：共享内存，管道（有名管道/无名管道），Socket，消息队列，信号，信号量，内存映射等。

【解释】

【题目编号】 CSD-11-1101-4009

【知 识 点】 Win32、MFC - 进程死锁

【难易程度】 60

【题目描述】

进程死锁的原因？

【正确答案】

资源竞争及进程推进顺序非法。

【解释】 无

【题目编号】 CSD-11-1101-4010

【知 识 点】 Win32、MFC - 死锁的必要条件

【难易程度】 60

【题目描述】

死锁的四个必要条件？

【正确答案】

互斥，请求保持，不可剥夺，环路。

【解释】 无

【题目编号】 CSD-11-1101-4011

【知 识 点】 Win32、MFC - Windows应用程序类型

【难易程度】 60

【题目描述】

windows应用程序的类型？

【正确答案】

控制台程序、窗口程序、库程序（静态/动态）。

【解释】 无

【题目编号】 CSD-11-1101-4012

【知 识 点】 Win32、MFC - 窗口程序的创建过程

【难易程度】 70

【题目描述】

窗口程序的创建过程？

【正确答案】

（1）定义WinMain入口函数

（2）定义窗口处理函数 WindowProc

（3）注册窗口类RegisterClass

（4）创建窗口CreateWindow

（5）显示窗口ShowWindow/UpdateWindow

（6）消息循环GetMessage

TranslateMessage

DisptachMessage

（7）消息处理

【解释】 无

【题目编号】 CSD-11-1101-4013

【知 识 点】 Win32、MFC - win32程序的执行机制

【难易程度】 60

【题目描述】

win32程序的执行机制？

【正确答案】

Win32窗口程序是采用事件驱动方式执行，也就是消息机制。(程序的执行机制分为过程驱动和事件驱动)

【解释】 无

【题目编号】 CSD-11-1101-4014

【知 识 点】 Win32、MFC - Win32中消息相关函数

【难易程度】 60

【题目描述】

列举win32中与消息相关的几个函数。

【正确答案】

GetMessage - 获取消息；TranslateMessage - 翻译消息。将按键消息，翻译成字符消息；DispatchMessage - 派发消息。将消息派发到该消息所属窗口的窗口处理函数上。

【解释】 无

【题目编号】 CSD-11-1101-4015

【知 识 点】 Win32、MFC - 窗口消息

【难易程度】 60

【题目描述】

列举几个常用的窗口消息？

【正确答案】

WM\_DESTROY、WM\_SYSCOMMAND、WM\_CREATE、WM\_SIZE、WM\_QUIT、WM\_PAINT等。

【解释】

【题目编号】 CSD-11-1101-4016

【知 识 点】 Win32、MFC - Windows消息分类

【难易程度】 60

【题目描述】

Windows中消息可以分为几类？

【正确答案】

系统消息，由系统定义好的消息，可以在程序中直接使用；用户自定义消息，由用户自己定义，满足用户自己的需求。由用户自己发出消息，并响应处理；应用程序消息，程序之间通讯时使用的消息；系统注册消息，在系统注册并生成相应消息，然后可以在各个程序中使用这个消息。

【解释】 无

【题目编号】 CSD-11-1101-4017

【知 识 点】 Win32、MFC - 消息队列

【难易程度】 60

【题目描述】

消息队列的类型？

【正确答案】

系统消息队列，由系统维护的消息队列。存放系统产生的消息，例如鼠标、键盘等；程序消息队列，属于每一个应用程序（线程）的消息队列。由应用程序（线程）维护。

【解释】 无

【题目编号】 CSD-11-1101-4018

【知 识 点】 Win32、MFC - windows程序入口和windows消息机制

【难易程度】 70

【题目描述】

Windows程序的入口是哪里？写出Windows消息机制的流程。

【正确答案】

入口在WinMain()函数。

Windows应用程序消息处理机制：

（1）操作系统接收应用程序的窗口消息，将消息投递到该应用程序的消息队列中

（2）应用程序在消息循环中调用GetMessage函数从消息队列中取出一条一条的消息，取出消息后，应用程序可以对消息进行一些预处理。

（3）应用程序调用DispatchMessage，将消息回传给操作系统。

（4）系统利用WNDCLASS结构体的lpfnWndProc成员保存的窗口过程函数的指针调用窗口过程，对消息进行处理。

【解释】 无

【题目编号】 CSD-11-1101-4019

【知 识 点】 Win32、MFC - assert函数的用法

【难易程度】 60

【题目描述】

函数assert的用法？

【正确答案】

断言assert是仅在debug版本起作用的宏，用于检查“不应该“发生的情况。程序员可以把assert看成一个在任何系统状态下都可以安全使用的无害测试手段。

【解释】 无

【题目编号】 CSD-11-1101-4020

【知 识 点】 Win32、MFC - Windows线程同步

【难易程度】 60

【题目描述】

Windows编程线程同步的几种方式？

【正确答案】

原子锁、临界区(段)、事件、互斥(体)、信号量、可等候定时器。

【解释】 无

【题目编号】 CSD-11-1101-4021

【知 识 点】 Win32、MFC - 同步机制

【难易程度】 70

【题目描述】

对几种同步机制的总结。

【正确答案】

（1）Critical Section（临界区）

A.速度快

B.不能用于不同进程

C.不能进行资源统计(每次只可以有一个线程对共享资源进行存取)

（2）Mutex（互斥量）

A.速度慢

B.可用于不同进程

C.不能进行资源统计

（3）Semaphore（信号量）

A.速度慢

B.可用于不同进程

C.可进行资源统计(可以让一个或超过一个线程对共享资源进行存取)

（4）Event

A.速度慢

B.可用于不同进程

C.可进行资源统计

【解释】 无

# 第12篇：ARM体系结构和接口编程

## 12.1

【题目编号】 CSD-12-1201-4022

【知 识 点】 ARM CISC 与 RISC

【难易程度】 65

【题目描述】

CISC体系与RISC体系分别指什么？

【正确答案】

RISC(精简指令集计算机)和CISC(复杂指令集计算机)是当前CPU的两种架构。他们的区别在于不同的CPU设计理念和方法。

CISC架构，它的设计目的是要用最少的机器语言指令来完成所需的计算任务。这种架构会增加CPU结构的复杂性和对CPU工艺的要求，但对于编译器的开发十分有利, 今天只有Intel及其兼容CPU还在使用CISC架构。精简指令集，RISC微处理器不仅精简了指令系统，采用超标量和超流水线结构；它们的指令数目只有几十条，却大大增强了并行处理

能力。

性能特点一：由于指令集简化后，流水线以及常用指令均可用硬件执行； 性能特点二：采用大量的寄存器，使大部分指令操作都在寄存器之间进行，提高了处理速度； 性能特点三：采用缓存—主机—外存三级存储结构，使取数与存数指令分开执行，使处理器可以完成尽可能多的工作，且不因从存储器存取信息而放慢处理速度。

【解释】 无

【题目编号】 CSD-12-1201-4023

【知 识 点】 ARM 访问内存的用法

【难易程度】 60

【题目描述】

在arm9模式下，对一个地址为0×80000000的寄存器，写入数值0×78

【正确答案】

\*((volatile unsigned int \*)0x80000000)=0x78

【解释】 无

【题目编号】 CSD-12-1201-4024

【知 识 点】 ARM 判断大端小端

【难易程度】 60

【题目描述】

对于整形变量A=0x12345678,请画出在little endian及big endian的方式下在内存中是如何存储的?编写一个c语言程序，判断大端模式和小端模式？

【正确答案】

a) 0x12345678

- little endian big endian 刚好反过来

- 高地址--〉 0x12 低地址--〉 0x12

- 0x34 0x34

- 0x56 0x56

- 低地址--〉 0x78 高地址--〉 0x78

{{{

Int CheckCpu()

{

union

{

int a;

char b;

}c;

c.a=1;

if(c.b==1)

printf(“小端”)；

else printf（“大端”）；

}

}}}

【解释】 无

【题目编号】 CSD-12-1201-4025

【知 识 点】 ARM 中断

【难易程度】 65

【题目描述】

什么是中断？中断发生时 CPU 做什么工作？

【正确答案】

中断是 CPU 响应外设需求的一种模式，在外设需要 CPU 时，会向中断控制器发送中断请求，这时 CPU 要保护现场，即把正在运行的程序保存起来，一般是把状态压入堆栈，然后读中断号，启动相应的中断服务程序，服务完成后，载入保护现场，即把堆栈的数据弹出，继续运行之前的程序。

- 中断：是指当主机接到外界硬件(如 I/O 设备)发来的信号时，马上停止原来的工作，转去处理这一事件，在处理完了以后，主机又回到原来的工作继续工作。中断是机器 BIOS 中最重要的概念，当系统外部设备有请求时，比如鼠标，串口数据同步请求等，都会产生中断，这在一个单 CPU 的环境下，系统会保存当前机器状态，响应这个请求，在请求完成后，就恢复设备状态；

【解释】 无

【题目编号】 CSD-12-1201-4026

【知 识 点】 ARM 死循环的使用

【难易程度】 60

【题目描述】

嵌入式系统中经常要用到无限循环，你怎么样用 C 编写死循环呢？

【正确答案】

While(1){}

For(;;){}

Loop:

…

goto loop

【解释】 无

【题目编号】 CSD-12-1201-4027

【知 识 点】 ARM 中断

【难易程度】 60

【题目描述】

中断(interrupt,如键盘中断)与异常(exception,如除零异常)有何区别?

【正确答案】

异常：在产生时必须考虑与处理器的时钟同步，实践上，异常也称为同步中断。在处理器执行到由于编程失误而导致的错误指令时，或者在执行期间出现特殊情况(如缺页)，必须靠内核处理的时候，处理器就会产生一个异常。

所谓中断应该是指外部硬件产生的一个电信号，从cpu的中断引脚进入，打断cpu当前的运行；所谓异常，是指软件运行中发生了一些必须作出处理的事件，cpu自动产生一个陷入来打断当前运行，转入异常处理流程。

【解释】 无

【题目编号】 CSD-12-1201-4028

【知 识 点】 ARM SPI传输

【难易程度】 90

【题目描述】

简述SPI传输方式

【正确答案】

- SPI：高速同步串行口，首发独立，可同步进行

- SPI接口主要应用在EEPROM，Flash，实时时钟，A/D转化器，数字信号处理，是一种全双工同步通讯总线，该接口一般使用四条线：串行时钟线（sck），主出从入线，主入从出线，低电平有效地的从机选择线。

- SPI总线（Serial Peripheral interface）

- SPI接口是Motorola 首先提出的全双工三线同步串行外围接口，采用主从模式（Master Slave）架构；支持多slave模式应用，一般仅支持单Master。时钟由Master控制，在时钟移位脉冲下，数据按位传输，高位在前，低位在后（MSB first）；SPI接口有2根单向数据线，为全双工通信，目前应用中的数据速率可达几Mbps的水平。SPI接口主要应用在 EEPROM，FLASH，实时时钟，AD转换器，还有数字信号处理器和数字信号解码器之间

- SPI接口共有4根信号线，事实上3根也可以（单向传输时）。分别是：设备选择线、时钟线、串行输出数据线、串行输入数据线。

- （1）MOSI：主器件数据输出，从器件数据输入

- （2）MISO：主器件数据输入，从器件数据输出

- （3）SCLK ：时钟信号，由主器件产生

- （4）nCS：从器件使能信号，由主器件控制

- 通讯是通过数据交换完成的，这里先要知道SPI是串行通讯协议，也就是说数据是一位一位的传输的。这就是SCK时钟线存在的原因，由SCK提供时钟脉冲，SDI，SDO则基于此脉冲完成数据传输。数据输出通过 SDO线，数据在时钟上升沿或下降沿时改变，在紧接着的下降沿或上升沿被读取（改变和读取恰好是在半个周期）。完成一位数据传输，输入也使用同样原理。这样，在至少8次时钟信号的改变（上沿和下沿为一次），就可以完成8位数据的传输。

- 要注意的是，SCK信号线只由主设备控制，从设备不能控制信号线。同样，在一个基于SPI的设备中，至少有一个主控设备。这样传输的特点：这样的传输方式有一个优点，与普通的串行通讯不同，普通的串行通讯一次连续传送至少8位数据，而SPI允许数据一位一位的传送，甚至允许暂停，因为SCK时钟线由主控设备控制，当没有时钟跳变时，从设备不采集或传送数据。也就是说，主设备通过对SCK时钟线的控制可以完成对通讯的控制。SPI还是一个数据交换协议：因为SPI的数据输入和输出线独立，所以允许同时完成数据的输入和输出。不同的SPI设备的实现方式不尽相同，主要是数据改变和采集的时间不同，在时钟信号上沿或下沿采集有不同定义，具体请参考相关器件的文档。

- AT91RM9200的SPI接口主要由4个引脚构成：SPICLK、MOSI、MISO及 /SS，其中SPICLK是整个SPI总线的公用时钟，MOSI、MISO作为主机，从机的输入输出的标志，MOSI是主机的输出，从机的输入，MISO 是主机的输入，从机的输出。/SS是从机的标志管脚，在互相通信的两个SPI总线的器件，/SS管脚的电平低的是从机，相反/SS管脚的电平高的是主机。在一个SPI通信系统中，必须有主机。SPI总线可以配置成单主单从，单主多从，互为主从。SPI的片选可以扩充选择16个外设,这时PCS输出=NPCS,说NPCS0~3接4-16译码器,这个译码器是需要外接4-16译码器，译码器的输入为NPCS0~3，输出用于16个外设的选择。

- 在SPI操作中，最重要的两项设置就是时钟极性（CPOL或UCCKPL）和时钟相位（CPHA或UCCKPH）。时钟极性设置时钟空闲时的电平，时钟相位设置读取数据和发送数据的时钟沿。主机和从机的发送数据是同时完成的，两者的接收数据也是同时完成的。所以为了保证主从机正确通信，应使得它们的SPI具有相同的时钟极性和时钟相位。

- SPI接口时钟配置心得：在主设备这边配置SPI接口时钟的时候一定要弄清楚从设备的时钟要求，因为主设备这边的时钟极性和相位都是以从设备为基准的。因此在时钟极性的配置上一定要搞清楚从设备是在时钟的上升沿还是下降沿接收数据，是在时钟的下降沿还是上升沿输出数据。如果 CPOL=0，串行同步时钟的空闲状态为低电平；如果CPOL=1，串行同步时钟的空闲状态为高电平。如果CPHA=0，在串行同步时钟的第一个跳变沿（上升或下降）数据被采样；如果CPHA=1，在串行同步时钟的第二个跳变沿（上升或下降）数据被采样

- 传输时序：SPI是一个环形总线结构，SPI接口在内部硬件实际上是两个简单的移位寄存器,传输的数据为8位，在主器件产生的从器件使能信号和移位脉冲下，按位传输，高位在前，低位在后。如下图所示，在SCLK的下降沿上数据改变，上升沿一位数据被存入移位寄存器。

- 数据传输：在一个SPI时钟周期内，会完成如下操作：1) 主机通过MOSI线发送1位数据，从机通过该线读取这1位数据；2) 从机通过MISO线发送1位数据，主机通过该线读取这1位数据。这是通过移位寄存器来实现的。如下图所示，主机和从机各有一个移位寄存器，且二者连接成环。随着时钟脉冲，数据按照从高位到低位的方式依次移出主机寄存器和从机寄存器，并且依次移入从机寄存器和主机寄存器。当寄存器中的内容全部移出时，相当于完成了两个寄存器内容的交换。

- 上文中最后一句话：SPI主模块和与之通信的外设备时钟相位和极性应该一致。个人理解这句话有2层意思：其一，主设备SPI时钟和极性的配置应该由外设来决定；其二，二者的配置应该保持一致，即主设备的SDO同从设备的SDO配置一致，主设备的SDI同从设备的SDI配置一致。因为主从设备是在SCLK的控制下，同时发送和接收数据，并通过2个双向移位寄存器来交换数据。工作原理演示如下图：

- 一个完整的传送周期是16位，即两个字节，因为，首先主机要发送命令过去，然后从机根据主机的命令准备数据，主机在下一个8位时钟周期才把数据读回来。

- 上升沿主机SDO发送数据1，同时从设备SDO发送数据0；紧接着在SCLK的下降沿的时候从设备的SDI接收到了主机发送过来的数据1，同时主机也接收到了从设备发送过来的数据0.

- SPI主要特点有:可以同时发出和接收串行数据;可以当作主机或从机工作;提供频率可编程时钟;发送结束 中断标志;写冲突保护;总线竞争保护等。下图示出SPI总线工作的四种方式，其中使用的最为广泛的是SPI0和SPI3方式 (实线表示):

- 优缺点

- SPI接口具有如下优点：

- 1) 支持全双工操作；

- 2) 操作简单；

- 3) 数据传输速率较高。

- 同时，它也具有如下缺点：

- 1) 需要占用主机较多的口线（每个从机都需要一根片选线）；

- 2) 只支持单个主机。

- 3) 没有指定的流控制，没有应答机制确认是否接收到数据。

- 在点对点的通信中，SPI接口不需要进行寻址操作，且为全双工通信，显得简单高效。在多个从设备的系统中，每个从设备需要独立的使能信号，硬件上比I2C系统要稍微复杂一些。

【解释】 无

【题目编号】 CSD-12-1201-4029

【知 识 点】 ARM I2C

【难易程度】 85

【题目描述】

简述I2C传输方式

【正确答案】

- I2C协议：是单片机与其他芯片进行通讯的协议：

- 1：只要求两条总线线路，一条是串行时钟线，一条是串行数据线；

- 2：通过软件设定地址

- 3：是一个多主机总线，如果两个或更多主机同时初始化数据传送可通过冲突检测和仲裁防止数据破坏；

- 4：I2C总线传输的是数据的总高位

- I2C(Inter－Integrated Circuit)总线是一种由PHILIPS公司开发的两线式串行总线，用于连接微控制器及其外围设备。从硬件上来说，只有两条线，十分简单，所以很多设备都使用这种协议。通过控制两条线的时序来传递数据。总线上可以挂接很多设备，那么主设备怎么找到对应的从设备呢？每一个从设备都有一个自己的地址，访问时遵守i2c总线协议，以eeprom为例：高4位固定，三位A0,A1,A2（由设备在板子上的连接确定的。）,最后一个读写位这个地址由从设备和开发板连线决定的。

{

1k/2K A0,A1,A2 板子上的连线决定

4K A2,A1,P0

8K A2,P1,P2

16K P2,P1,P0 分页访问

}

- 因为s5pv210处理器集成了i2c的控制器，所以我们只需要给几个寄存器赋值就可以使用了，但是有的cpu没有集成i2c控制器，那么怎么办？

- 1 外接一个控制器

- 2 用两个gpio模拟时序输出，根据图 可以看出在scl为低时改变sda的值。这样才有效，发送1要在scl为低电平的时候进行。

- I2C总线最主要的优点是其简单性和有效性。由于接口直接在组件之上，因此I2C总线占用的空间非常小，减少了电路板的空间和芯片管脚的数量，降低了互联成本。

i2c（IIC）通信协议

定义： I2C 两线式串行总线

SCL: 时钟线

SDA: 数据线

【解释】 无

【题目编号】 CSD-12-1201-4030

【知 识 点】 ARM 中断和轮询

【难易程度】 70

【题目描述】

中断和轮询哪个效率高？怎样决定是采用中断方式还是采用轮询方式去实现驱动？

【正确答案】

中断是CPU处于被中状态下来接受设备的信号，而轮询是CPU主动去查询该设备是否有请求。凡事都是两面性，所以，看效率不能简单的说那个效率高。如果是请求设备是一个频繁请求cpu的设备，或者有大量数据请求的网络设备，那么轮询的效率是比中断高。如果是一般设备，并且该设备请求cpu的频率比较底，则用中断效率要高一些。

【解释】 无

【题目编号】 CSD-12-1201-4031

【知 识 点】 ARM 工作模式

【难易程度】 65

【题目描述】

= ARM7、ARM9、ARM11有多少种工作模式（）。 =

- A 2

- B 3

- C 5

- D 7

【正确答案】 D

【解释】

ARM微处理器支持7种运行模式，分别为：

- 用户模式(usr)：ARM处理器正常的程序执行状态。

- 快速中断模式(fiq)：用于高速数据传输或通道处理。

- 外部中断模式(irq)：用于通用的中断处理。

- 管理模式(svc)：操作系统使用的保护模式。

- 数据访问终止模式(abt)：当数据或指令预取终止时进入该模式，可用于虚拟存储及存储保护。

- 系统模式(sys)：运行具有特权的操作系统任务。

- 定义指令中止模式(und)：当未定义的指令执行时进入该模式，可用于支持硬件协处理器的软件仿真。

【题目编号】 CSD-12-1201-4032

【知 识 点】 ARM 寄存器

【难易程度】 65

【题目描述】

= ARM7微处理器中私有寄存器最多与访问寄存器数量最少的模式分别是（）。 =

- A USER/SYS， SVC

- B IRQ, USER/SYS

- C FIQ， USER/SYS

- D SVC， IRQ

【正确答案】 C

【解释】

问题是 私有寄存器，强调私有，所以FIQ最多有r8到r14还有SPSR，能访问最少

的自然是基本最低的USER/SYS模式，所以答案是C

【题目编号】 CSD-12-1201-4033

【知 识 点】 ARM LR寄存器

【难易程度】 65

【题目描述】

ARM微处理器中有些寄存器在某些模式下是没有的，如：用户模式没有SPSR寄存器，请问没有LR寄存器的模式是（ ）。 =

- A SVC

- B IRQ

- C FIQ

- D 以上都不对

【正确答案】 D

【解释】

LD是连接寄存器，用于程序的返回地址，在每一种模式里面都需要该寄存器所以答案是D

【题目编号】 CSD-12-1201-4034

【知 识 点】 ARM 终止模式

【难易程度】 65

【题目描述】

= ARM7微处理器中会导致处理器进入中止模式的异常是（ ）。 =

- A 快速中断异常

- B 指令预取异常

- C 软件中断异常

- D 未定义指令异常

【正确答案】 B

【解释】

- 指令预取异常会导致进入终止模式，快速中断异常，软件中断异常，分别进入快速中断模式(fiq)和管理模式(svc)，未定义指令异常进入未定义指令中止模式，所以答案是B

【题目编号】 CSD-12-1201-4035

【知 识 点】 ARM 寄存器

【难易程度】 65

【题目描述】

= ARM微处理器的CPSR寄存器的BIT5表示的含义是（）。 =

- A Z 位

- B T 位

- C I 位

- D F位

【正确答案】 B

【解释】 - CPSR的BIT5是T位，答案为B

【题目编号】 CSD-12-1201-4036

【知 识 点】 ARM 汇编

【难易程度】 65

【题目描述】

= 以下ARM指令中不访问存储器的是（）。 =

- A ADD

- B LDR

- C LDMFD

- D STR

【正确答案】 A

【解释】 ADD是对寄存器的操作，不访问存储器，所以答案是A

【题目编号】 CSD-12-1201-4037

【知 识 点】 ARM SRAM和DRAM

【难易程度】 65

【题目描述】

= 以下哪项关于SRAM和DRAM的区别是不对 =

- A SRAM比DRAM慢

- B SRAM比DRAM耗电多

- C DRAM存储密度比SRAM高得多

- D DRM需要周期性刷新

【正确答案】 A

【解释】

- SRAM是英文Static RAM的缩写，它是一种具有静止存取功能的内存，不需要刷新电路即能保存它内部存储的数据。不像DRAM内存那样需要刷新电路，每隔一段时间，固定要对DRAM刷新充电一次，否则内部的数据即会消失，因此SRAM具有较高的性能，但是SRAM也有它的缺点，即它的集成度较低，相同容量的DRAM内存可以设计为较小的体积，但是SRAM却需要很大的体积，所以在主板上SRAM存储器要占用一部分面积

- 优点，速度快，不必配合内存刷新电路，可提高整体的工作效率。

- 缺点，集成度低，功耗较大，相同的容量体积较大，而且价格较高，少量用于关键性系统以提高效率。

- DRAM（Dynamic Random-Access Memory），即动态随机存储器最为常见的系统内存。DRAM 只能将数据保持很短的时间。为了保持数据，DRAM使用电容存储，所以 必须隔一段时间刷新（refresh）一次，如果存储单元没有被刷新，存储的信息就会丢失。 答案是A。

【题目编号】 CSD-12-1201-4038

【知 识 点】 ARM 文件系统格式

【难易程度】 65

【题目描述】

= 以下哪种方式不是文件系统的格式。 =

- A Yaffs2

- B DOS

- C Cramfs

- D EXT

【正确答案】 B

【解释】

- Yaffs2 ，Cramfs是嵌入式文件系统，EXT扩展文件系统是linux文件系统，只有DOS是操作系统，不是文件系统，答案是B。

【题目编号】 CSD-12-1201-4039

【知 识 点】 ARM UART

【难易程度】 65

【题目描述】= 串行通信中，根据数据传输方向不同，可以分成三种方式，不包括的方式 =

- A 单工

- B 双工

- C 半单工

- D 半双工

【正确答案】 C

【解释】

串行通信中有单工，双工，半双工，没有半单工的，答案是C。

【题目编号】 CSD-12-1201-4040

【知 识 点】 ARM 汇编

【难易程度】 65

【题目描述】

= 在汇编过程中，不产生机器码，只用来指示汇编程序如何汇编的指令是 =

- A 宏指令

- B 伪指令

- C 汇编指令

- D 机器指令

【正确答案】 B

【解释】

- 伪指令又称为伪操作，它是在对源程序汇编期间由汇编程序处理的操作，它们可以完成如处理器选择、定义程序模式、定义数据、分配存储区、指示程序结束等功能。

- 宏指令是由用户按照宏定义格式编写的一段程序，其中语句可以是指令、伪指令，甚至是已定义的宏指令，答案是B

【题目编号】 CSD-12-1201-4041

【知 识 点】 ARM

【难易程度】 65

【题目描述】

= 某嵌入式系统采用字节编址方式，内存由A4000H到BBFFFH 共有多少KB =

- A 32

- B 64

- C 96

- D 128

【正确答案】 C

【解释】

- BBFFFH – A4000H = 17FFFH 1KB的地址范围是：0～3FFH

- 17FFF/3FF = 96，即A4000H到BBFFFH共有96KB，答案是C

【题目编号】 CSD-12-1201-4042

【知 识 点】 ARM 同步通信和异步通信

【难易程度】 65

【题目描述】

= 同步通信比异步通信具有较高的传输频率，这是因为 =

- A 同步通信不需要应答信号

- B 同步通信方式的总线长度较短

- C 同步通信采用一个公共时钟信号进行同步

- D 同步通信中各部件存取时间比较短

【正确答案】 C

【解释】

同步通信中，接收端和发送端的收发时钟是严格同步的，也就是说通信过程中采用同一个公共时钟信号进行同步。由于收发时钟是严格同步，在接收端就不 会因接收时钟和发送时钟不一致所造成的时钟误差积累。正是这种收发时钟的严格同步，才允许在同步通信中，可以用很高的传输速率一次传送几十、几百甚至更多 字节的数据。异步通信中，接收端和发送端的收发时钟是不需要严格同步的，也就是说在通信过程中收发时钟是异步的，是有小的不同(允许在5%以内)。正是这 种收发时钟的异步，在接收端就会因接收时钟和发送时钟不一致所造成的时钟误差积累。这就使得异步通信的传输速率低，每次只能传送几位数据。答案是C。

【题目编号】 CSD-12-1201-4043

【知 识 点】 ARM 内存计算

【难易程度】 65

【题目描述】

= 某数码相机内置 128MB 的存储空间，拍摄分辨率设定为 1600×1200 像素，颜色深度为24位，若不采用压缩存储技术，使用内部存储器最多可以存储多少张照片 =

- A 12

- B 25

- C 13

- D 23

【正确答案】 D

【解释】

- 每个象素存储空间为：24/3＝8Byte

- 每张照片的存储空间为：1600×1200×3＝ 5760008Byte ＝ 5625KB 数码相机的存储空间为：128MB = 128×1204KB ＝ 131072KB 可以保存的照片数量为：131072/5625 = 23.301，答案是D。

【题目编号】 CSD-12-1201-4044

【知 识 点】 ARM 相对地址

【难易程度】 65

【题目描述】

= 编译程序对高级语言源程序进行翻译时，需要在该程序的地址空间中为变量指定地址，这种地址称为 =

- A 物理地址

- B 接口地址

- C 逻辑地址

- D 线性地址

【正确答案】 A

【解释】

逻辑地址也叫相对地址或虚地址。用户的程序经过汇编或编译后形成目标代码，而目标代码通常采用的就是相对地址的形式，其首地址为0，其余指令中的地址都是相对于这个首地址来编址的。

【题目编号】 CSD-12-1201-4045

【知 识 点】 中断方式

【难易程度】 65

【题目描述】

= 采用什么方式可以使得设备与主存间的数据块传输无需CPU的干预 =

- A 程序控制输入输出

- B 中断

- C DMA

- D 总线控制

【正确答案】 C

【解释】

- 与外设进行数据交换的的方法有程序控制方式，中断方式，DMA。其中DMA方式下，CPU只需要向DMA控制器下达指令，让DMA控制器管理数据的传输，传输完毕后把结果反馈给CPU，减少了CPU的参与度，节省CPU资源。

【题目编号】 CSD-12-1201-4046

【知 识 点】 ARM UART

【难易程度】 65

【题目描述】

= 某系统以RS232为接口，进行7位ASCII码字符传送，带有一位奇校验位和两位停止位，当波特率为9600时，字符传输率是多少 =

- A 960

- B 873

- C 840

- D 480

【正确答案】 A

【解释】

所谓字符传输率就是指每一秒传输多少个字符，根据条件可知传送一个字符共需要10位二进制位，当波特率为9600时，字符传输率等于波特率除以传输一个字符的位数，结果为960，答案是A

【题目编号】 CSD-12-1201-4047

【知 识 点】 ARM 内存计算

【难易程度】 65

【题目描述】

= 存储1000个16 \*16点阵汉字所需要的存储空间是多少 =

- A 256KB

- B 32KB

- C 16KB

- D 31.25KB

【正确答案】 D

【解释】 需要的存储空间=1000 \* 16 \*16/8/1024=31.25KB，答案是D

【题目编号】 CSD-12-1201-4048

【知 识 点】 ARM 临界区

【难易程度】 65

【题目描述】

= 临界区是什么 =

- A 一个缓冲区

- B 一段程序

- C 一段共享数据区

- D 一个互斥资源

【正确答案】 B

【解释】

在多进程系统中，有些资源一次只能让一个进程使用，如打印机，串口等，这样的资源被称为临界资源，在进程中对临界资源实施操作的那段程序就被称为临界区，答案选B

【题目编号】 CSD-12-1201-4049

【知 识 点】 ARM CPSR的NZCV位

【难易程度】 65

【题目描述】

= 下列CPSR寄存器标志位的作用说法错误的是 =

- A N：负数

- B Z: 零

- C C: 进位

- D V: 借位

【正确答案】 D

【解释】

V:是溢出标志位，所谓借位就是C位，答案是D

【题目编号】 CSD-12-1201-4050

【知 识 点】 ARM RISC计算机

【难易程度】 65

【题目描述】

= 下列描述不属于RISC计算机的特点的是 =

- A 流水线每周期前进一步

- B 更多通用寄存器

- C 指令长度不固定，执行需要多个周期

- D 独立的Load和Store指令完成数据在寄存器和外部存储器之间的传输

【正确答案】 C

【解释】

RISC微处理器不仅精简了指令系统，采用超标量和超流水线结构；它们的指令数目只有几十条，却大大增强了并行处理能力。

- 性能特点一：由于指令集简化后，流水线以及常用指令均可用硬件执行，其指令通常是固定长度；

-　　 性能特点二：采用大量的寄存器，使大部分指令操作都在寄存器之间进行，提高了处理速度；

- 　　 性能特点三：采用缓存—主存—外存三级存储结构，使取数与存数指令分开执行，使处理器可以完成尽可能多的工作，且不因从存储器存取信息而放慢处理速度

【题目编号】 CSD-12-1201-4051

【知 识 点】 ARM 寄存器

【难易程度】 65

【题目描述】

= 寄存器R13除了可以做通用寄存器外，还可以做 =

- A 程序计数器

- B 链接寄存器

- C 栈指针寄存器

- D 基址寄存器

【正确答案】 C

【解释】

ARM处理器中通常将寄存器R13作为堆栈指针(SP)，寄存器R15用作程序计数器(PC), 14称为子程序链接寄存器LR(Link Register),答案是C

【题目编号】 CSD-12-1201-4052

【知 识 点】 ARM 中断模式

【难易程度】 65

【题目描述】

= 在嵌入式ARM处理器中，下面哪种中断方式优先级最高 =

- A 复位

- B 数据中止

- C FIQ

- D IRQ

【正确答案】 A

【解释】

复位的优先级是最高的，答案是A

【题目编号】 CSD-12-1201-4053

【知 识 点】 ARM公司的职责

【难易程度】 65

【题目描述】

属于ARM公司的产品或设计

- A 嵌入式微处理器

- B 嵌入式微控制器

- C 嵌入式DSP

- D SoC

【正确答案】 A

【解释】

ARM公司只设计处理器，答案是A

【题目编号】 CSD-12-1201-4054

【知 识 点】 ARM

【难易程度】 65

【题目描述】

= ARM9采用的ARM架构版本是 =

- A V2

- B V3

- C V4

- D V5

【正确答案】 C

【解ARM9使用的是ARMv4的架构版本 答案是C

【题目编号】 CSD-12-1201-4055

【知 识 点】 ARM

【难易程度】 65

【题目描述】

= ARM处理器上电复位后处于的模式是 =

- A 用户

- B 系统

- C 管理

- D 中断

【正确答案】 C

【解释】

RM处理器上电复位后处于管理模式，答案是C

【题目编号】 CSD-12-1201-4056

【知 识 点】 ARM load/store

【难易程度】 65

【题目描述】

= Load/Store的单字操作指令其地址应是（ ）的倍数 =

- A 1

- B 2

- C 4

- D 8

【正确答案】 C

【解释】

load用于从内存中取数据，而STORE用于存数据到内存，地址必须是字对齐的,否则从内存中读出的数值需进行循环，所以都是4的倍数，答案C

# 第13篇：Linux移植和驱动

【题目编号】 CSD-13-1301-4057

【知 识 点】 移植内核配置命令

【难易程度】 65

【题目描述】

= 内核的配置命令. =

【正确答案】

- 内核配置的方法：make config、make xconfig、make menuconfig、make oldconfig等等

【解释】 无

【题目编号】 CSD-12-1201-4058

【知 识 点】 移植U-boot的启动过程

【难易程度】 65

【题目描述】 U-boot的启动过程 =

【正确答案】

1》 硬件初始化：前4K是在cpu内部内存中运行的，即SRAM

- 1）主要作用设置异常向量表，进入管理模式，屏蔽IRQ,FIQ中断，关闭看门狗，关闭中断，关MMU，cache，时钟，内存，

- Start.S

- 初始化cpu

- 初始化时钟

- 关闭中断

- 关闭看门狗

- Lowlevel\_init.o:初始化各个bank

- 初始化内存

- 初始化串口

- 初始化flash

- 初始化网卡

- 2）加载U-Boot第二阶段代码到RAM空间 //选择从哪里启动，是nor flash还是nand flash

- 3）软件初始化：

- 设置堆栈

- 清空bss

- 4）跳转到第二阶段代码入口

-（搬运的第二段代码的作用）

- 初始化本阶段使用的硬件设备

-检测系统内存映射

-将内核从Flash读取到RAM中

-为内核设置启动参数

-调用内核

- 2扩展内容：

- LOGO显示

- 组合按键：实现一些特定的功能，如刷机

- 3引导系统

- 加载操作系统内核，并且给内核传递参数，并且正确挂接完根文件系统以后，uboot才算结束。

【解释】 无

【题目编号】 CSD-12-1201-4059

【知 识 点】 移植linux启动过程

【难易程度】 65

【题目描述】 简述ARM-linux启动过程 =

【正确答案】

- 对于 ARM 处理器来说上电或复位后执行的第一条指令，该地址为 0x00000000。对于一般的嵌入式系统，通常把 Flash 等非易失性存储器映射到这个地址处，而 bootloader就位于该存储器的最前端，所以系统上电或复位后执行的第一段程序便是 bootloader，在bootloader中主要完成：

- 1)设置和初始化 RAM，为调用 Linux内核做好准备。初始化 RAM 的任务包括设置 CPU 的控制寄存器参数，以便能正常使用 RAM 以及检测RAM 大小等。

- 2) 初始化串口 ， 串口在 Linux 的启动过程中有着非常重要的作用，它是 Linux内核和用户交互的方式之 一。Linux 在启动过程中可以将信息通过串口输出，这样便可清楚的了解 Linux 的启动过程。

- 3) 检测处理器类型

- 4) 设置 Linux启动参数 Bootloader在执行过程中必须设置和初始化 Linux 的内核启动参数

- 5) 调用 Linux内核映像

- Linux 内核有两种映像：一种是非压缩内核，叫 Image，另一种是它的压缩版本，叫 - zImage。根据内核映像的不同，Linux 内核的启动在开始阶段也有所不同。

- 但为了能使用 zImage，必须在它的开头加上 解压缩的代码，将 zImage 解压缩之后才能执行，zImage 的入口程序即为 arch/arm/boot/compressed/head.S。

- 它依次完成以下工作：开启 MMU 和 Cache，调用 decompress\_kernel()解压内核，最后通过 调用 call\_kernel()进入非压缩内核 Image 的启动。

- Linux 非压缩内核的入口位于文件/arch/arm/kernel/head-armv.S 中的 stext 段。该段的基地址就是压缩内核解压后的跳转地址。该程序通过查找处理器内核类型和处理器类型调用相应的初始化函数，再建立页表，最后跳转到start\_kernel()函数开始内核的初始化工作。完成剩余的与 硬件平台相关的初始化工作，在进行一系列与内核相关的初始化后，调用第一个用户进程－ init 进程并等待用户进程的执行，这样整个 Linux 内核便启动完毕。该函数所做的具体工作

- 1) 调用 setup\_arch()函数进行与体系结构相关的第一个初始化工作； 对不同的体系结构来说该函数有不同的定义。对于 ARM 平台而言，该函数定义在 arch/arm/kernel/Setup.c。它首先通过检测出来的处理器类型进行处理器内核的初始化，然后 通过 bootmem\_init()函数根据系统定义的 meminfo 结构进行内存结构的初始化，最后调用 paging\_init()开启 MMU，创建内核页表，映射所有的物理内存和 IO空间。

- 2) 创建异常向量表和初始化中断处理函数；

- 3) 初始化系统核心进程调度器和时钟中断处理机制；

- 4) 初始化串口控制台（serial-console）； ARM-Linux 在初始化过程中一般都会初始化一个串口做为内核的控制台，这样内核在 启动过程中就可以通过串口输出信息以便开发者或用户了解系统的启动进程。

- 5) 创建和初始化系统 cache，为各种内存调用机制提供缓存，包括;动态内存分配，虚拟文件系统（VirtualFile System）及页缓存。

- 6) 初始化内存管理，检测内存大小及被内核占用的内存情况；

- 7) 初始化系统的进程间通信机制（IPC）；

【解释】无

【题目编号】 CSD-12-1201-4060

【知 识 点】 内核中断处理机制的方法

【难易程度】 65

【题目描述】 = 在linux内核中希望你的中断处理程序执行时间越短越好，但是中断处理程序要完成的功能不变，内核是通过什么机制解决矛盾？ =

【正确答案】

- 在Linux内核中，为了在中断执行时间尽可能短和中断处理需完成大量工作之间找到一个平衡点，Linux将中断处理程序分为两个部分：上半部（top half）和下半部（bottom half）。 中断处理程序的上半部在接收到一个中断时就立即执行，但只做比较紧急的工作，这些工作都是在所有中断被禁止的情况下完成的，所以要快，否则其它的中断就得 不到及时的处理。那些耗时又不紧急的工作被推迟到下半部去。中断处理程序的下半部分（如果有的话）几乎做了中断处理程序所有的事情。它们最大的不同是上半 部分不可中断，而下半部分可中断。在理想的情况下，最好是中断处理程序上半部分将所有工作都交给下半部分执行，这样的话在中断处理程序上半部分中完成的工 作就很少，也就能尽可能快地返回。但是，中断处理程序上半部分一定要完成一些工作，例如，通过操作硬件对中断的到达进行确认，还有一些从硬件拷贝数据等对 时间比较敏感的工作。剩下的其他工作都可由下半部分执行。

- 对于上半部分和下半部分之间的划分没有严格的规则，靠驱动程序开发人员自己的编程习惯来划分，不过还是有一些习惯供参考：

- 如果该任务对时间比较敏感，将其放在上半部中执行。

- 如果该任务和硬件相关，一般放在上半部中执行。

- 如果该任务要保证不被其他中断打断，放在上半部中执行（因为这是系统关中断）。

- 其他不太紧急的任务， 一般考虑在下半部执行。

- 下半部分并不需要指明一个确切时间，只要把这些任务推迟一点，让它们在系统不太忙并且中断恢复后执行就可以了。通常下半部分在中断处理程序一返回就会马上运行。内核中实现下半部的手段不断演化，目前已经从最原始的BH（bottom half）衍生出BH（在2.5中去除）、软中断（softirq在2.3引入）、tasklet（在2.3引入）、工作队列（work queue在2.5引入）。稍后笔者将介绍后两种方式。

- 尽管上半部和下半部的结合能够改善系统的响应能力，但是，Linux设备驱动中的中断处理并不一定要分成两个半部。如果中断要处理的工作本身就很少，则完全可以直接在上半部全部完成。

【解释】

【题目编号】 CSD-12-1201-4061

【知 识 点】 驱动 字符设备和块设备

【难易程度】 65

【题目描述】= Linux设备中字符设备与块设备有什么主要的区别？请分别列举一些实际的设备说出它们是属于哪一类设备。 =

【正确答案】

- 字符设备：字符设备是个能够像字节流（类似文件）一样被访问的设备，由字符设备驱动程序来实现这种特性。字符设备驱动程序通常至少实现open,close,read和write系统调用。字符终端、串口、鼠标、键盘、摄像头、声卡和显卡等就是典型的字符设备。

块设备：和字符设备类似，块设备也是通过/dev目录下的文件系统节点来访问。块设备上能够容纳文件系统，如：u盘，SD卡，磁盘等。

- 字符设备和块设备的区别仅仅在于内核内部管理数据的方式，也就是内核及驱动程序之间的软件接口，而这些不同对用户来讲是透明的。在内核中，和字符驱动程序相比，块驱动程序具有完全不同的接口

【解释】无

【题目编号】 CSD-12-1201-4062

【知 识 点】 驱动 查看命令

【难易程度】 65

【题目描述】 = 查看驱动模块中打印信息应该使用什么命令？如何查看内核中已有的字符设备的信息？如何查看正在使用的有哪些中断号？ =

【正确答案】

- 1) 查看驱动模块中打印信息的命令：dmesg

- 2) 查看字符设备信息可以用lsmod 和modprobe，lsmod可以查看模块的依赖关系，modprobe在加载模块时会加载其他依赖的模块。

- 3）显示当前使用的中断号cat /proc/interrupt

【解释】 无

【题目编号】 CSD-12-1201-4063

【知 识 点】 移植 用户空间和内核空间

【难易程度】 65

【题目描述】 = ARM-linux用户空间和内核空间有什么区别？ =

【正确答案】

- 用户空间：USR模式

- 内核空间：SVC模式

- 用户不能直接访问内核空间的地址信息包括代码和数据

- 对于4G的虚拟地址空间的划分：

- 用户空间地址：0x00000000~0xBFFFFFFF

- 内核空间地址：0xC0000000~0xFFFFFFFF

- 用户空间和内核空间数据的交互通过系统调用（read,write..）

- 用户空间：包含应用程序和一堆的库，例如：

- printf，malloc:C库函数

- read,write:系统调用函数，实现的过程在C库里

- 用户空间不能直接访问硬件资源：用户应用程序不能直接访问寄 存器对应的物理地址信息

- 划分的目的：安全的保护，一旦应用程序进行非法的地址访问，操作系统有权将其停止，但是在内核空间编程一定要注意没有人能够管理内核，如果地址非放，系统崩溃！

【解释】无

【题目编号】 CSD-12-1201-4064

【知 识 点】 驱动中内核函数

【难易程度】 65

【题目描述】 copy\_to\_user()和copy\_from\_user()主要用于实现什么功能？一般用于file\_operations结构的哪些函数里面？

【正确答案】

- 由于内核空间和用户空间是不能互相访问的，如果需要访问就必须借助内核函数进行数据读写。copy\_to\_user():完成内核空间到用户空间的复制，copy\_from\_user()：是完成用户空间到内核空间的复制。一般用于file\_operations结构里的read,write,ioctl等内存数据交换作用的函数。当然，如果ioctl没有用到内存数据复制，那么就不会用到这两个函数。

【解释】无

【题目编号】 CSD-12-1201-4065

【知 识 点】 驱动注册字符设备

【难易程度】 65

【题目描述】 设备驱动程序中如何注册一个字符设备？分别解释一下它的几个参数的含义。

【正确答案】

- 由于内核空间和用户空间是不能互相访问的，如果需要访问就必须借助内核函数进行数据读写。copy\_to\_user():完成内核空间到用户空间的复制，copy\_from\_user()：是完成用户空间到内核空间的复制。一般用于file\_operations结构里的read,write,ioctl等内存数据交换作用的函数。当然，如果ioctl没有用到内存数据复制，那么就不会用到这两个函数。

注册一个字符设备驱动有两种方法：

- 1） void cdev\_init(struct cdev \*cdev, struct file\_operations \*fops)

- 该注册函数可以将cdev结构嵌入到自己的设备特定的结构中。cdev是一个指向结构体cdev的指针，而fops是指向一个类似于file\_operations结构（可以是file\_operations结构，但不限于该结构）的指针.

- 2） int register\_chrdev(unsigned int major, const char \*namem , struct file)operations \*fopen);

- 该注册函数是早期的注册函数，major是设备的主设备号，name是驱动程序的名称，而fops是默认的file\_operations结构（这是只限于file\_operations结构）。对于register\_chrdev的调用将为给定的主设备号注册0－255作为次设备号，并为每个设备建立一个对应的默认cdev结构。

【解释】无

【题目编号】 CSD-12-1201-4066

【知 识 点】 驱动中断注册函数和注销函数

【难易程度】 65

【题目描述】 Linux设备驱动程序中，使用哪个函数注册和注销中断处理程序？

【正确答案】

- 中断注册函数和中断注销函数

- 注册中断：

- int request\_irq(unsigned int irq, irqreturn\_t (\*handler)(int, void \*, struct pt\_regs \*), unsigned long flags, const char \*dev\_name, void \*dev\_id);

- 参数意义依次是：中断号，中断处理函数，中断管理有关的掩码，中断请求设备名，中断信号线。

- 过程是：dev\_name设备请求中断->cpu分配中断号->设置中断管理的掩码->分配中断信号线->处理中断函数->完成之后再根据设置情况返回原处理程序处继续处理程序。

- 注销中断；

- Void free\_irq(unsigned int irq, void \*dev\_id);

- 释放中断和中断信号线

【解释】无

【题目编号】 CSD-12-1201-4067

【知 识 点】 驱动主设备号和次设备号

【难易程度】 65

【题目描述】 请简述主设备号和次设备号的用途

【正确答案】

主设备号：主设备号标识设备对应的驱动程序。虽然现代的linux内核允许多个驱动程序共享主设备号，但我们看待的大多数设备仍然按照“一个主设备对应一个驱动程序”的原则组织。

- 次设备号：次设备号由内核使用，用于正确确定设备文件所指的设备。依赖于驱动程序的编写方式，我们可以通过次设备号获得一个指向内核设备的直接指针，也可将此设备号当作设备本地数组的索引。

【解释】无

【题目编号】 CSD-12-1201-4068

【知 识 点】 内核内存申请函数

【难易程度】 65

【题目描述】 linux内核里面，内存申请有哪几个函数，各自的区别？

【正确答案】

- Kmalloc() \_\_get\_free\_page() mempool\_create()

- kmalloc函数申请的内存可以任意大小，但是kmalloc最大只能开辟128k-16，16个字节是被页描述符结构占用了。

- 而get\_free\_page或get\_free\_pages是申请的内存以页的大小为基准，大小为页的整数倍。 get\_dma\_pages,这个函数除了以上功能外还能支持DMA传输。

【解释】无

【题目编号】 CSD-12-1201-4069

【知 识 点】 mmap实现原理

【难易程度】 65

【题目描述】 内核函数mmap的实现原理，机制？

【正确答案】

mmap函数实现把一个文件映射到一个内存区域，从而我们可以像读写内存一样读写文件，他比单纯调用read/write也要快上许多。在某些时候我们可以把内存的内容拷贝到一个文件中实现内存备份，当然，也可以把文件的内容映射到内存来恢复某些服务。另外，mmap实现共享内存也是其主要应用之一，mmap系统调用使得进程之间通过映射同一个普通文件实现共享内存

【解释】无

【题目编号】 CSD-12-1201-4070

【知 识 点】 驱动 并发 互斥

【难易程度】 65

【题目描述】 驱动里面为什么要有并发、互斥的控制？如何实现？

【正确答案】

- mmap函数实现把一个文件映射到一个内存区域，从而我们可以像读写内存一样读写文件，他比单纯调用read/write也要快上许多。在某些时候我们可以把内存的内容拷贝到一个文件中实现内存备份，当然，也可以把文件的内容映射到内存来恢复某些服务。另外，mmap实现共享内存也是其主要应用之一，mmap系统调用使得进程之间通过映射同一个普通文件实现共享内存。

【解释】 无

【题目编号】 CSD-12-1201-4071

【知 识 点】 驱动内核模块命令

【难易程度】 65

【题目描述】 = 操作内核模块的命令 =

【正确答案】

- insmod:加载模块

- lsmod:查看当前内核有哪些模块

- rmmod:卸载模块

- modinfo：查看模块本身的信息

- modprobe:加载和卸载模块的命令，它能够检查模块的依赖关系，这种检查要通过modules.dep文件来实现。insmod不会检查模块的依赖关系。每当使用modprobe加载模块时，它会到默认的路径/lib/modules/内核版本信息/modules.dep找这个文件。

【解释】无

【题目编号】 CSD-12-1201-4072

【知 识 点】 进程同步 进程互斥

【难易程度】 65

【题目描述】

实时操作系统中，两个任务并发执行，一个任务要等待其合作伙伴发来信息，或建立某个条件后再向前执行，这种制约性合作关系被成为

- A 同步

- B 互斥

- C 调度

- D 执行

【正确答案】 A

【解释】

进程同步是一个操作系统级别的概念,是在多道程序的环境下，存在着不同的制约关系，为了协调这种互相制约的关系，实现资源共享和进程协作，从而避免进程之间的冲突，引入了进程同步。

- 进程互斥是进程之间的间接制约关系。当一个进程进入临界区使用临界资源时，另一个进程必须等待。只有当使用临界资源的进程退出临界区后，这个进程才会解除阻塞状态。

【题目编号】 CSD-12-1201-4073

【知 识 点】 linux

【难易程度】 65

【题目描述】

= 每种嵌入式操作系统都有自身的特点以吸引相关用户，下列说法错误的是 =

- A 嵌入式Linux提供了完善的网络技术支持

- B uCLinux是专门为没有MMU的ARM芯片开发的

- C uC/OS-Ⅱ操作系统是一种实时操作系统

- D WinCE提供完全开放的源代码并且免费

【正确答案】 D

【解释】

WinCE系统是全部开源了,不过不是所有人都能拿到,除了wince安装时public,private里面带的以外,其它的需要满足一定条件,并向微软提出申请的才可以看到，并且不免费，使用时需要购买授权，答案是D

【题目编号】 CSD-12-1201-4074

【知 识 点】 内核系统

【难易程度】 65

【题目描述】

= 内核不包括的子系统是 =

- A 进程管理系统

- B 内存管理系统

- C I/O管理系统

- D 硬件管理系统

【正确答案】 D

【解释】

Linux内核是一个庞大而复杂的操作系统的核心，不过尽管庞大，但是却采用子系统和分层的概念很好地进行了组织。五个子系统 进程调度,内存管理,虚拟文件系统,网络接口,进程间通信