## 面向对象程序设计模拟试卷

一. 单选题(15)。

1. 关于定义“struct A{int x; const int y=3;};volatile a={1};”，如下叙述那个\_\_D\_\_\_正确:

A. a.x是int、a.y是const int类型 B. a.x是volatile int、a.y是const int类型

C. a.x是int、a.y是const volatile int类型D. a.x是volatile int、a.y是const volatile int类型

2. 关于inline constexpr int &f(int &&a)的调用叙述\_\_B\_\_\_正确:

A.该用常量做实参调用，返回传统右值 B. 该用常量做实参调用，返回传统左值

C.可用变量做实参调用，返回传统右值 D. 可用变量做实参调用，返回传统左值

3. 若A是B的父类，B是C的父类，则catch应按如下哪种\_\_\_B\_\_放置:

A. catch(A x)在catch(B x)的前面， catch(B x)在catch(C x)的前面

B. catch(A x)在catch(B x)的后面， catch(B x)在catch(C x)的后面

C. catch(A x)在catch(B x)的前面， catch(C x)在catch(A x)的前面

D. catch(A x)在catch(B x)的后面， catch(C x)在catch(A x)的后面

4. 对于定义struct A{int x; int y:2; }m;下列叙述\_\_\_A\_\_正确:

A. m.x是有址左值, m.y是无址左值 B. m.x是无址左值, m.y是有址左值

C. m.x是有址右值, m.y是无址右值 B. m.x是无址右值, m.y是有址右值

5. 对于定义“char \* &&f( );”，如下哪个语句是错误的\_\_A\_\_\_:

A. f( )=(char\*) "abcd"; B. \*f( )= 'A';

C. char \*p=f( ); D. \*f( )= 1 ["ABC"];

二. 在最多使用一级作用域“：：”访问如A::c的情况下，指出各类可访问的成员及其访问权限(20) 。

 

类A的可访问成员：

private: int a(或A::a);

protected: int b(或A::b);

public: int c(或A::c);

类B的可访问成员：

private: int d(或B::d);

protected: int b(或A::b, B::b),e(或B::e);

public: int c(或A::c, B::c),f(或B::f);

类C的可访问成员：

private: int g(或C::g);

protected: int b(或A::b, C::b), h(或C::h);

public: int c(或A::c, C::c),i(或C::h);

类D的可访问成员：

private: int C::b,B::c,C::c,e(或B::e，D::e), f(或B::f,D::f),h(或C::h,D::h), i(或C::i,D::i),

j(或D::j),n(或D::n);

protected: int B::b,k(或D::k);

三. 指出main中每行的输出结果(20) 。

#include <iostream>

using namespace std;

struct A { A( ) { cout << 'A'; } };

struct B { B() { cout << 'B'; } };

struct C : A { C() { cout << 'C'; } };

struct D : virtual B, virtual C { D() { cout << 'D'; } };

struct E : A {

C c;

E() : c() { cout << 'E'; }

};

struct F : virtual B, C, virtual D, E {

F() { cout << 'F'; }

};

void main() {

A a; cout << '\n'; //输出=A

B b; cout << '\n'; //输出=B

C c; cout << '\n'; //输出=AC

D d; cout << '\n'; //输出=BACD

E e; cout << '\n'; //输出=AACE

F f; cout << '\n'; //输出=BACDACAACEF

}

四. 指出以下程序的语法错误及其原因(15) 。

class A {

static int a=0;//(1)前面有inline或const或constexpr才能初始化,否则在类外初始化

protected:

int b;

public:

int c;

A(int) {};

inline constexpr operator int() { return b; };

} a(1, 2); //(2)没有两个参数的构造函数

class B: A {

B(int m) { b = m; }; //(3)A不存在无参构造函数供“B(int m):”后调用

using A::b;

virtual int d; //(4)virtual不能用于说明数据成员

int e;

public:

int b; //(5)前有using A::b, 不得重复定义同名数据成员b

friend B& operator =(const B&b){return \*this;}//(6)等号仅单参，应为实例函数成员

virtual B(int, int); //(7)不能用virtual定义构造函数

} b = 5; //(8)单参构造函数是私有的，无法访问

class C: B {

public:

C operator++(double) { return \*this; };//（9）后置运算必须int类型定义显式参数

}c; //(10)类C无法生成无参构造函数初始化c

int main( ) {

int \* A::\* p, i;

i = a.a; //(11)私有成员，main函数无法访问

i = A(4);

i = b.c; //（12）A继承到B的成员B::c是私有的，main函数无法访问

p = &A::c;//(13)&A::c的类型为int A::\*，与p的类型int\* A::\*不同，不能赋值给p

i = b;//(14)B未定义operator int函数，继承自A的operator int私有化，main不能访问

return;//(15)main要求返回一个整型值

}

五. 指出main变量i在每条赋值语句执行后的值(15) 。

int x=\_\_\_\_\_\_\_(请填入本人学号最后一位数字), y=x+30;

struct A{

static int x;

int y;

public:

operator int( ){ return x-y; }

A operator ++(int){ return A(x++, y++); }

A(int x=::x+2, int y=::y+3){ A::x=x; A::y=y; }

int &h(int &x);

};

int &A::h(int &x)

{

for(int y=1; y!=1|| x<201; x+=11, y++) if(x>200) { x-=21; y-=2;}

return x-=10;

}

int A::x=23;

void main( ){

A a(54, 3), b(65), c;

int i, &z=i, A::\*p=&A::y;

z=b.x;

i=a.x;

i=c.\*p;

i=a++;

i=::x+c.y;

i=a+b;

b.h(i)=7;

}

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 学号尾数 | z = b.x | i = a.x | i=c.\*p | i= a++ | i=::x+c.y | i=a+b | b.h(i)=7 |
| 0 | 2 | 2 | 33 | -1 | 33 | -33 | 7 |
| 1 | 3 | 3 | 34 | 0 | 35 | -32 | 7 |
| 2 | 4 | 4 | 35 | 1 | 37 | -31 | 7 |
| 3 | 5 | 5 | 36 | 2 | 39 | -30 | 7 |
| 4 | 6 | 6 | 37 | 3 | 41 | -29 | 7 |
| 5 | 7 | 7 | 38 | 4 | 43 | -28 | 7 |
| 6 | 8 | 8 | 39 | 5 | 45 | -27 | 7 |
| 7 | 9 | 9 | 40 | 6 | 47 | -26 | 7 |
| 8 | 10 | 10 | 41 | 7 | 49 | -25 | 7 |
| 9 | 11 | 11 | 42 | 8 | 51 | -24 | 7 |

六. 英文字典的单词区分大小写，并且不能存放重复的单词，定义如下字典类DIC中的函数成员(15)。

#define \_CRT\_SECURE\_NO\_WARNINGS //防止strcpy等出现安全报警

#include <string.h>

class DIC {

    char\*\* const e; //用于存放单词，部分大小写

    const int m; //能够存放的最大单词个数

    int c; //已经存放的单词个数

public:

    DIC( )noexcept; //用于构造空字典

    DIC(int m); //用于构造单词个数最多为m的字典

    DIC(const DIC&); //字典深拷贝构造函数

    DIC(DIC&&)noexcept; //字典移动构造函数

    DIC& operator=(const DIC&); //字典深拷贝赋值运算符重载函数

    DIC& operator=(DIC&&)noexcept; //字典移动赋值运算符重载函数

    DIC& operator<<(const char\*); //将一个单词不重复放入字典

    int  where(const char\*); //查找单词在字典中的下标位置，没找到时返回-1

    char\* operator[](int x); //返回下标值为x的单词

    ~DIC()noexcept; //字典析构函数

};

解：

DIC::DIC()noexcept : e(nullptr), m(0), c(0) {}

DIC::DIC(int m) : e(new char\* [m]), m(e ? m : 0), c(0) {

    for (int i = 0; i < m; i++) e[i] = nullptr;

}

DIC::DIC(const DIC&d):e(new char\* [d.m]), m(e ? d.m : 0) {

    if(e==nullptr) throw "insufficient memory\n";

    for (c = 0; c < d.c; c++) {

        e[c] = new char[strlen(d.e[c])+1];

        if (e[c] == nullptr) throw "insufficient memory\n";

        strcpy(e[c], d.e[c]);

    }

}

DIC::DIC(DIC&& d)noexcept :e(d.e), m(d.m), c(d.c) {

    (char\*&)d.e = nullptr;

    (int&)d.m = d.c = 0;

}

DIC& DIC::operator=(const DIC&d) {

    if (this == &d) return \*this;

    if (e) {

        for (int i = 0; i < c; i++) delete e[i];

        delete e;

    }

    (char\*\*&)e = new char\* [d.m];

    if (e == nullptr) throw "insufficient memory\n";

    (int&)m = e ? d.m : 0;

    for (c = 0; c < d.c; c++) {

        e[c] = new char[strlen(d.e[c]) + 1];

        if (e[c] == nullptr) throw "insufficient memory\n";

        strcpy(e[c], d.e[c]);

    }

    return \*this;

}

DIC& DIC::operator=(DIC&& d)noexcept {

    if (this == &d) return \*this;

    if (e) {

        for (int i = 0; i < c; i++) delete e[i];

        delete e;

    }

    (char\*\*&)e = d.e;

    (int&)m = d.m;

    c = d.c;

    (char\*\*&)d.e = nullptr;

    (int&)d.m = d.c = 0;

}

DIC& DIC::operator<<(const char\*s){

    if (where(s)!=-1) return \*this;

    if (c == m) throw "dictionary is full\n";

    e[c] = new char[strlen(s) + 1];

    if (e[c] == nullptr) throw "insufficient memory\n";

    strcpy(e[c++], s);

    return \*this;

}

int  DIC::where(const char\*s) {

    for (int i = 0; i < c; i++)

        if (strcmp(e[i], s) == 0) return i;

    return -1;

}

char\* DIC::operator[](int x) {

    if (x < 0 || x >= c) throw "index is illegal\n";

    return e[x];

}

DIC::~DIC()noexcept {

    if (e) {

        for (int i = 0; i < c; i++) delete e[i];

        delete e;

        (char\*\*&)e = nullptr;

        (int&)m = c = 0;

    }

}