Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Score:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ECE 2036 Test II**

**Open book and notes, PCs and tablets allowed, but no Internet Access and C++ code cannot be run on a PC**

1. (*10%*) Assuming the short C/C++ code segment below compiles and runs, what does it output?

**string str(" ECE 2036 ");**

**string mystr;**

**str = str + "Test II Fall 2014";**

**mystr = str.substr(7,8);**

**cout <<mystr <<dec<<str[5]<<mystr.length()<<hex<< str.find("1");**

**cout.put(0x42);**

**cout << endl;**

Output: \_\_\_\_\_\_*36 Test 2819B*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(**Note:** make sure any spaces are apparent in the answer )

1. (*6%*) What does the following code print out?

**#include <iostream>**

**using namespace std;**

**int main() {**

**try {**

**try {**

**cout << "Start try block" << endl;**

**throw 2031;**

**cout << "End try block" << endl;**

**throw 2035;**

**}**

**catch (int a) { cout << "Error number:" << a << endl;** t**hrow a;}**

**throw 13;**

**}**

**catch (int a) { cout << "Fatal Error: " << a; }**

**cout << endl;**

**}**

|  |
| --- |
| *Start try block* |
| *Error number: 2031* |
| *Fatal Error: 2031* |
|  |

1. (*4%*) The C++ ver 11 \_\_\_\_\_*array*\_\_\_\_\_\_\_\_\_\_\_ STL container is the most efficient choice when a random access array is needed, size can only be determined at runtime, and size does not change after the initial allocation call.
2. (*6%)* Write a C/C++ template definition that automatically generates a function called “Biggest” that returns the larger of two values of the same type (i.e., for integers *Biggest(5, 3)* returns 5, or for doubles *Biggest(3.0, 5.0)* returns 5.0). You can assume that the “>” operator is defined (i.e, overloaded, if needed) for any type that is used with the template. Use pass by value for arguments. Include a sample function call for two ints *x* and *y*.

***template <class T>***

***T Biggest( T a, T b)***

***{***

***if (a>b) return a;***

***else return b;***

***}***

***//sample call: int w = Biggest(x,y);***

1. (*10%*) On mbed, what two major C++ Object Oriented features made it possible for the *uLCD.printf()* function to work for the mbed color uLCD even though *printf()* is not found anywhere in the uLCD class code? (Include the two common terms used for these C++ features in your answer)

***Inheritance: printf() is defined in the stream base class used by the uLCDclass***

***Polymorphism: The virtual function \_putc is used by printf() to send a character to the I/O device (LCD). The uLCD class supplies this virtual function for sending the character to the uLCD***

1. (*10%*) Write a new “*polar*” class for a complex number that is stored in polar coordinates (i.e., a magnitude and an angle in degrees). Both internal values in the class should be type double using private (and no dynamic allocation). Include the extra code needed so that “*cout<<mypolar*;” where *mypolar* is a *polar* object would print the value in the format “*magnitude*@*angle*”. No member functions are needed other than a constructor that takes two type double arguments (i.e., magnitude and angle) to initialize data in a *polar* object.

***class polar {***

***friend std::ostream &operator<<(std::ostream &, const polar &);***

***public:***

***polar(double m, double a): mag(m), ang(a){}***

***//for problem 7: polar operator\*(const polar &rhs);***

***private:***

***double mag;***

***double ang;***

***};***

***ostream &operator<<(ostream &output, const polar &rhs){***

***output<<rhs.mag<<"@"<<rhs.ang;***

***return output;***

***}***

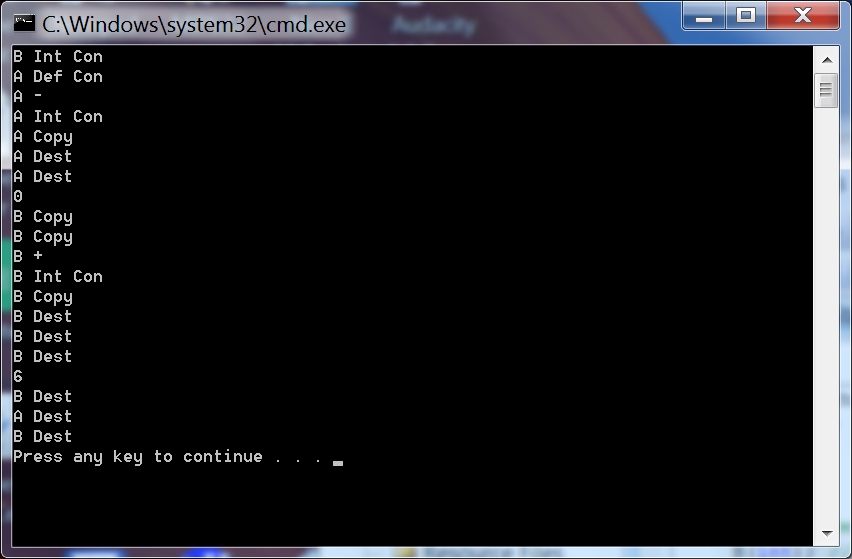
1. (*10%*) Write the additional code segment needed to be added so that a C++ statement like “*mypolar1* = *mypolar2* \* *mypolar3*;” would work in the new *polar* class as expected (i.e., magnitudes are multiplied and angles are added). Assume the new function prototype needed for this code is already added to the *polar* class above and put the remaining code needed outside of the class definition in the space below.

***polar polar::operator\* (const polar &rhs){***

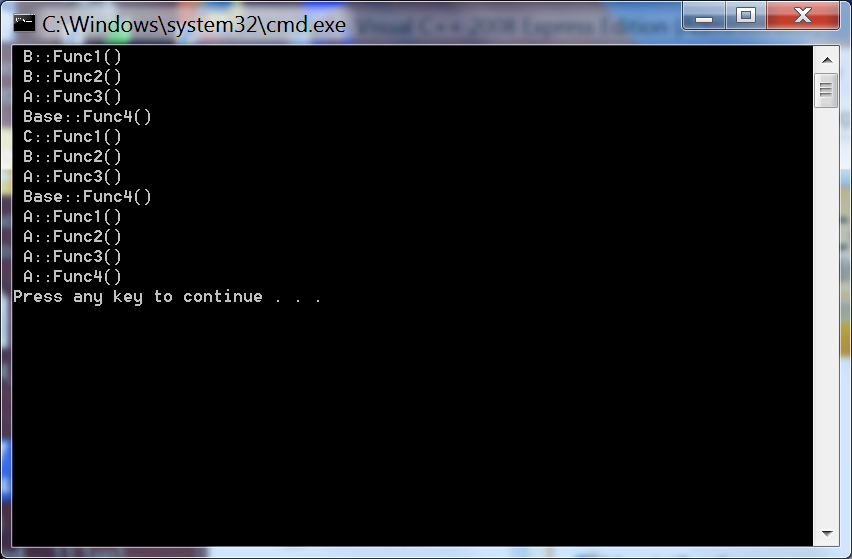
***return polar(mag\*rhs.mag, ang+rhs.ang);***

***}***

1. (***2****0%*) Write the output in the space below that is produced by the Class constructor, destructor and operator overloading example C/C++ code provided with the test. You may have extra spaces in the table below. Recall that most compilers also use the copy constructor to make a new copy of the object whenever pass by value is used (instead of a pass by reference). Assume this also happens whenever a function returns a value that was previously setup as a local variable. Note: There may be extra lines in the table provided below.



1. (*24%*) Write the output in the space below that is produced by the Inheritance and Polymorphism example C/C++ code provided with the test. You may have extra spaces in the table below.



**// Problem 8: C++ Code - Class Constructors, Destructors and Operator Overloading**

**#include <iostream>**

**using namespace std;**

**class A{**

**public:**

**A(); // Default constructor**

**A(int); // int Constructor**

**A(const A&); // Copy constructor**

**~A(); // Destructor**

**A operator-(const A& rhs) const;**

**public:**

**int x; // Single data member**

**};**

**A::A(): x(1){ cout<<"A Def Con"<<endl; };**

**A::A(int x): x(x){ cout<<"A Int Con"<<endl; };**

**A::~A(){ cout<<"A Dest"<<endl; };**

**A::A(const A &a){**

**x=a.x;**

**cout<<"A Copy"<<endl;**

**};**

**A A::operator-(const A& rhs) const**

**{**

**cout<<"A -"<<endl;**

**A r(x - rhs.x);**

**return r;**

**}**

**class B{**

**public:**

**B(); // Default Constructor**

**B(int); // int Constructor**

**B(const B&); // Copy constructor**

**~B(); // Destructor**

**B operator+(B rhs) const;**

**void print() { cout << x << endl;}**

**private:**

**int x; // Single data member**

**};**

**B::B(): x(1){ cout<< "B Def Con"<<endl;};**

**B::B(int x): x(x){ cout<< "B Int Con"<<endl;};**

**B::~B(){ cout<< "B Dest"<<endl;};**

**B::B(const B &b){**

**x=b.x;**

**cout<<"B Copy"<<endl;**

**};**

**B B::operator+(B rhs) const {**

**cout<< "B +"<<endl;**

**B r(x + rhs.x);**

**return r;**

**}**

**int main()**

**{**

**B b(3);**

**A a;**

**a = a - a;**

**cout << a.x << endl;**

**B c(b);**

**b = b + c;**

**b.print();**

**}**

**// Problem 9 : Inheritance and Polymorphism Example C++ Code**

**//**

**#include "stdafx.h"**

**#include <iostream>**

**using namespace std;**

**class Base**

**{ // Define a base class**

**public:**

**virtual void Func1() = 0;**

**virtual void Func2();**

**virtual void Func3();**

**void Func4();**

**};**

**class A : public Base**

**{ // Class A derives from Base**

**public:**

**virtual void Func1();**

**virtual void Func2();**

**virtual void Func3();**

**virtual void Func4();**

**};**

**class B : public A**

**{// Class B derives from A**

**public:**

**virtual void Func1();**

**void Func2();**

**};**

**class C : public B**

**{ // Class C derives from B**

**public:**

**virtual void Func1();**

**virtual void Func4();**

**};**

**// Base Class Methods**

**void Base::Func2(){ cout << " Base::Func2()" << endl;}**

**void Base::Func3(){ cout << " Base::Func3()" << endl;}**

**void Base::Func4(){ cout << " Base::Func4()" << endl;}**

**// Class A Methods**

**void A::Func1() { cout << " A::Func1()" << endl; }**

**void A::Func2() { cout << " A::Func2()" << endl; }**

**void A::Func3() { cout << " A::Func3()" << endl; }**

**void A::Func4() { cout << " A::Func4()" << endl; }**

**// Class B Methods**

**void B::Func1() { cout << " B::Func1()" << endl; }**

**void B::Func2() { cout << " B::Func2()" << endl; }**

**// Class C Methods**

**void C::Func1() { cout << " C::Func1()" << endl; }**

**void C::Func4() { cout << " C::Func4()" << endl; }**

**void TestFuncVal(A x){**

**x.Func1();**

**x.Func2();**

**x.Func3();**

**x.Func4();**

**}**

**void TestFuncRef(Base& x){**

**x.Func1();**

**x.Func2();**

**x.Func3();**

**x.Func4();**

**}**

**int main(){**

**B b;**

**C c;**

**TestFuncRef(b);**

**TestFuncRef(c);**

**TestFuncVal(c);**

**}**