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

**ECE 2036 Test II**

**Open book and notes, No PCs, Tablets allowed, no Internet Access, and C++ code cannot be run on any device**

1. (*9%*) Assuming the short C/C++ code segment below compiles and runs, what does it output?

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

 **string mystr;**

 **str = str + " Spring 2014";**

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

 **cout << str[6]<<hex<<mystr<<mystr.length()<<dec<< str.find("1");**

 **cout.put(0x61);**

 **cout << endl;**

Output: \_\_\_\_***IECE 203 826a***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(**Note:** make sure any spaces are apparent in the answer above)

1. (*6%*) What does the following code print out? You may have extra spaces in the table below.

**#include <stdexcept>**

**#include <iostream>**

**using namespace std;**

**void funrun2(){**

 **cout << "funrun2" << endl;**

 **throw runtime\_error("bad mistake!");**

**}**

**void funrun(){**

 **cout << "funrun" << endl;**

 **funrun2();**

 **throw runtime\_error("fatal error");**

**}**

**int main(){**

 **try {**

 **funrun();**

 **throw 2036;**

 **}**

 **catch (runtime\_error &error){**

 **cout << "Error: " << error.what() << endl;**

 **}**

**}**

|  |
| --- |
| ***funrun*** |
| ***funrun2*** |
| ***Error: bad mistake!*** |
|  |

1. (*7%)* Write C/C++ code that automatically generates a function called “*AbsCubeIt*” that returns the absolute value of the cube of the value (i.e. abs(x3) ) for any type (i.e., for integers *AbsCubeIt(5)* returns 125, or for doubles *AbsCubeIt(-3.0)* returns 27.0). You can assume that the “\*”, ”<” and \*-\* operators are defined (i.e., overloaded, if needed) for any type that is used with *AbsCubeIt*, and that a conversion from int “0” to the type used is available. Use a const reference for the argument. Include sample function calls for an integer value of 2 and a double value of -8.0.

***template <class T> int x = AbsCubeIt(3);***

***T AbsCubeIt(const T& x){ double y = AbsCubeIt(-8.0);***

 ***T y = x \* x \* x;***

 ***if (y<0) y= -y;***

 ***return y;***

***}***

1. (*8%*) The C++ \_\_\_\_\_\_***forward\_list***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ STL container is the likely the most efficient choice when random access is not needed, size can only be determined at runtime, size changes after the initial allocation, only forward iterators are needed, and there are frequent insertions and deletions (not only at ends).
2. (*4%*) The mbed’s API classes (i.e., DigitalIn, DigitalOut…) control hardware pin I/O functions that need to be initialized to select the correct pin outputs by initializing several I/O registers prior to use. Where would this be easy to do in the class, and what forces every program to read in these mbed C++ class definitions?

*Initialize hardware in class* ***constructor***

 ***#include “mbed.h”*** *includes the new mbed class definition code*

1. (*10%*) Given the code below, fill in the missing C++ code so that the first *cout* in *main* will work and print out “1.0 i1.0” (just skip the + and – sign stuff before the “i” to keep it simple, after the “i” is OK)
2. (*12%*) Next, add the additional missing code needed below so that a line like “x = y;” below in *main* will work along with the entire program and be mathematically correct. **Hint:** C++ also has a *cos()* function and both *sin* and *cos* take radians for the angle argument. The Polar class already has the angle argument in radians. Don’t just overload the “=” operator, you have seen the idea of what is needed here before in the mbed’s TMP36 temperature sensor class code.

**#include "stdafx.h" //for Windows only**

***#include <iostream>***

***#include <cmath>***

***using namespace std;***

**class Complex {**

**public:**

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

 **Complex(double r, double i): real(r), imag(i){}**

**private:**

 **double real;**

 **double imag;**

**};**

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

 ***output << rhs.real << " i" << rhs.imag;***

 ***return output;***

**class Polar {**

**public:**

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

 **double mag;**

 **double ang;**

 ***operator Complex (){return Complex(mag\*cos(ang),mag\*sin(ang)); };***

**};**

**int main(){**

 **Complex x(1.0, 1.0);**

 **cout << x << endl;**

 **Polar y(1.414213, -3.14159/4.0);**

 **x = y;**

 **cout << x;**

 **return 0;**

**}**

***//Note: could also move Polar class first and add a new Complex //constructor with a Polar arg – compiler knows to use this to convert.***

***// i.e.Complex(Polar p): real(p.mag\*cos(p.ang),imag(p.mag\*sin(p.ang)){}***

1. (*20%*) 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 or pointer). Assume this also happens whenever a function returns a value that was previously setup as a local variable, but not when the return creates a new object. Note: There may be extra lines in the table provided below.



Note: Minor typo in A overload + operator code makes it subtract.

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(0){ 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;**

 **return B(x \* rhs.x);**

**}**

**int main()**

**{**

 **A a(3);**

 **B b(2);**

 **A c(a);**

 **B d;**

 **d = b \* b;**

 **a = a + c;**

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

 **d.print();**

**}**

**// Problem 9 C++ code – Virtual Functions, Inheritance, and Polymorphism**

**#include <iostream>**

**using namespace std;**

**class Base { // Define a base class**

**public:**

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

 **virtual void Func2();**

 **virtual void Func3();**

 **virtual void Func4() = 0;**

**};**

**class A : public Base { // Class A derives from Base**

**public:**

 **virtual void Func1();**

 **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;}**

**// Class A Methods**

**void A::Func1() { cout << " A::Func1()" << 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(){**

 **A a;**

 **B b;**

 **C c;**

 **TestFuncRef(a);**

 **TestFuncRef(b);**

 **TestFuncRef(c);**

 **TestFuncVal(c);**

**}**