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

**ECE 2036 Test 1**

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

1. (*5%)* A \_\_\_\_*stack*\_\_\_\_\_\_\_\_\_\_ is used automatically by compilers to store subroutine or function return addresses along with any local variables.
2. (*5%)* What new C++ language feature (not available in C and not directly string related, but related to the use of >> and <<) is used to implement a simple output statement such as “*cout << i;* “ or input statement such as “*cin >> i;*”.

*operator overloading of “<<” and “>>”*

1. (*5%)* Fill in the typical size (per textbook and slides) in bytes of the following C/C++ variable types:

char \_\_*\_1*\_\_ bytes int \_\_\_*4*\_\_\_bytes float\_\_\_*4\_\_* bytes

double array[1000] \_\_*8000*\_\_ bytes C style string with 80 characters \_\_*81*\_\_\_ bytes

1. (*5%)* As a good programming practice, what should always be the last case be in a very complex switch statement?

*The default case*

1. (*10%)* Write a C/C++ *for* statement using an integer loop control variable “*j*” that would add the first ten elements of an existing 1D integer array called “*Array”* using an existinginteger variable *“ total”* that is already initialized to zero*. ”j”* must go out of scope when the for loop exits.

*for( int j=0; j<10; j++) {*

*total = total + Array[j];*

*}*

1. (*10%)* Write a C/C++ *void* function called *absolute* with one argument that changes the argument to its absolute value in the calling program. The value of the argument is an integer. Show an example call with the integer variable *j*.

*Need to use a pointer or reference to change the value in the calling program*

*void absolute(int \*arg) { OR void absolute(int &arg){*

 *if (\*arg<0) \*arg = -\*arg; if (\*arg<0) \*arg = -\*arg;*

*} }*

*//example call*

*absolute(&j); absolute(j);*

1. (*20%)* Write the output produced by this program exactly as it will appear on the output device.

*-3 -6 -9*

*-12 -15 -18*

*-21 -24 -27*

*-30*

#include <iostream>

using namespace std;

class test

{

public:

 test();

 void x(test y);

public:

 int w;

};

test::test(){

 w = 0;

}

void test::x(test y)

{

 w = y.w - 3 ;

}

int main(int argc, \_TCHAR\* argv[])

{

 test A;

 for (int i=1; i<=10; i++){

 A.x(A);

 cout << A.w << " ";

 if (i%3==0) cout<< endl;

 }

}

1. (*25%)* Write the output that is produced by this C/C++ program.

*34854*

#include <iostream>

using namespace std;

int main(int argc, \_TCHAR\* argv[])

{

 int a[8]={1,2,3,4,5,6,7,8};

 int \*aptr;

 aptr = &a[0];

 a[1] = a[2] + 1;

 a[3] = a[2] + a[4];

 aptr++;

 (\*aptr)--;

 (\*(++aptr))++;

 cout << a[1] << a[2] << a[3] << a[4] << \*aptr << endl;

}

1. (*15%)* Write the output in the space below that is produced by the Constructor Destructor example C/C++ code provided with the test on the last page. Recall that most compilers 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.

|  |
| --- |
| *Y Int Con* |
| *Y Copy Con* |
| *X Def Con* |
| *Y Copy Con* |
| *Y +* |
| *Y Int Con* |
| *-Y Destructor* |
| *-Y Destructor* |
| *X +* |
| *X Int Con* |
| *X Copy Con* |
| *-X Destructor* |
| *-X Destructor* |
|  *6 4* |
| *-X Destructor* |
| *-Y Destructor* |
| *-Y Destructor* |
|  |
|  |
|  |
|  |
|  |

// Constructors and Destructors Problem Code

#include <iostream>

using namespace std;

class X {

public:

 X(); // Default constructor

 X(int); // int Constructor

 X(const X&); // Copy constructor

 ~X(); // Destructor

 X operator+(const X& rhs) const; // Addition operator

public:

 int x; // Single data member

};

X::X(): x(2){ cout<<"X Def Con "<<endl; };

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

X::~X(){ cout<<"-X Destructor "<<endl; };

X::X(const X &a){

 x=a.x;

 cout<<"X Copy Con "<<endl;

};

X X::operator+(const X& rhs) const

{

 cout<<"X + "<<endl;

 X r(x + rhs.x);

 return r;

}

class Y {

public:

 Y(); // Default Constructor

 Y(int); // int Constructor

 Y(const Y&); // Copy constructor

 ~Y(); // Destructor

 Y operator+(Y rhs) const; // Addition operator

public:

 int x; // Single data member

};

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

Y::Y(int x): x(x){ x++; cout<< "Y Int Con "<<endl;};

Y::~Y(){ cout<< "-Y Destructor "<<endl;};

Y::Y(const Y &b){

 x=b.x;

 cout<<"Y Copy Con "<<endl;

};

Y Y::operator+(Y rhs) const

{

 cout<< "Y + "<<endl;

 return Y(x + rhs.x);

}

int main(){

 Y b(3);

 Y c(b);

 X a;

 b = b + c;

 a = a + a;

 cout << b.x << " " << a.x <<endl;

}