ECE 7650 ASSIGNMENT REPORT

(C++ and Object Oriented Computing for Engineers)

Author:	Jamiu Babatunde Mojolagbe
Department:	Electrical and Computer Engineering
Student ID:	#7804719
Email:	mojolagm@myumanitoba.ca
Course:	ECE 7650
Sub. Date:	February 12, 2018

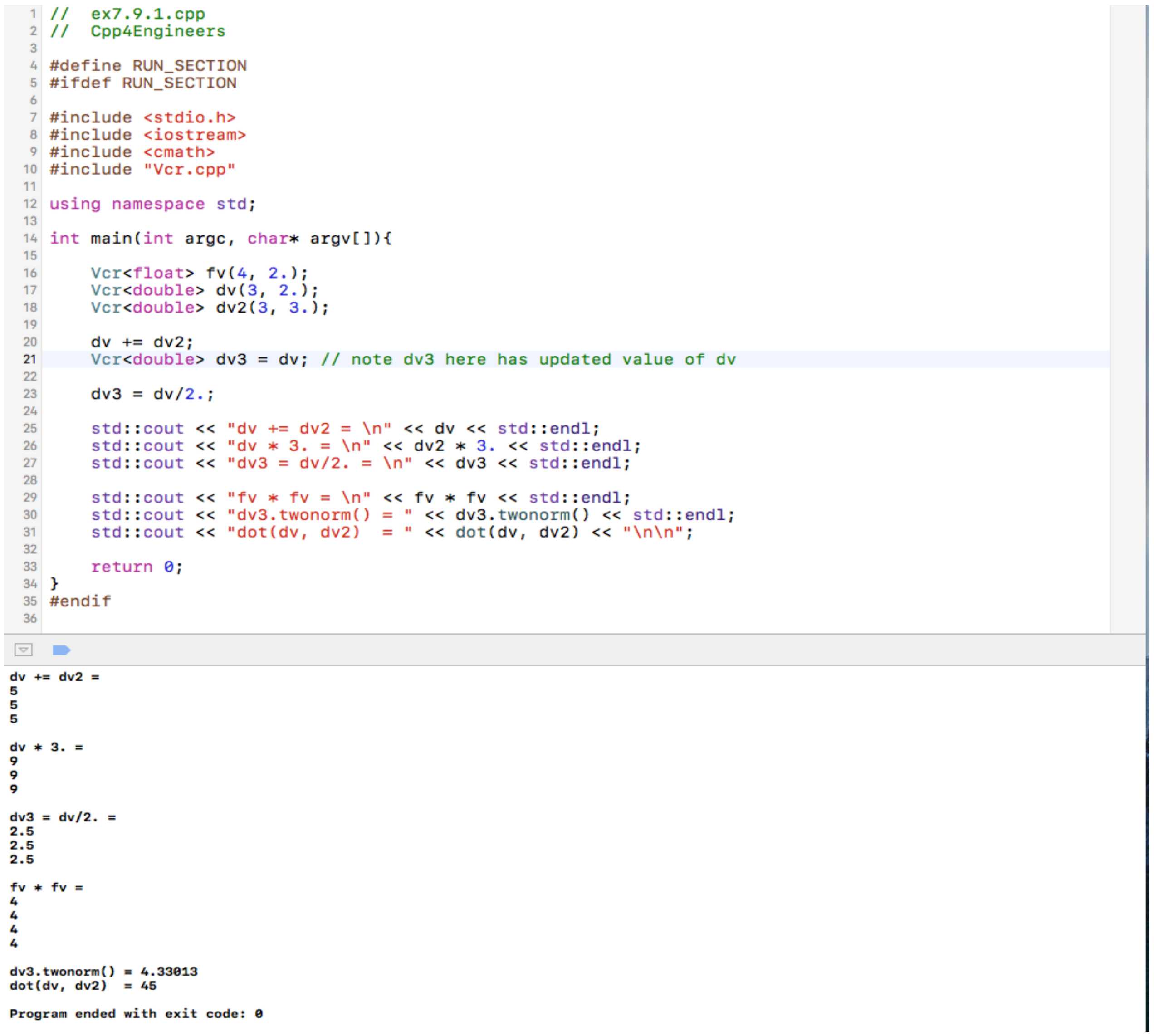
QUESTION (7)

All codes as regards to this question 7 are contained in folder named “**Chapter7**”.

Question 7.9.1

The class template for vectors as illustrated in §7.1 with additional operations such as vector addition and vector multiplication are contained in the files named “**Vcr.h**” and “**Vcr.cpp**”. While the “Vcr.h” is the header file containing the class definition with the class data and function signatures; “Vcr.cpp” contains the implementation of the class template itself.

The driver program for this vector class implementation is contained in the file named “**ex7.9.1.cpp**”. The content of the driver program and the results obtained are shown below:



```cpp
//   ex7.9.1.cpp
//   Cpp4Engineers

#define RUN_SECTION
#ifdef RUN_SECTION

#include <stdio.h>
#include <iostream>
#include <cmath>
#include "Vcr.cpp"

using namespace std;

int main(int argc, char* argv[]){

    Vcr<float> fv(4, 2.);
    Vcr<double> dv(3, 2.);
    Vcr<double> dv2(3, 3.);

    dv += dv2;
    Vcr<double> dv3 = dv; // note dv3 here has updated value of dv

    dv3 = dv/2.;

    std::cout << "dv += dv2 = \n" << dv << std::endl;
    std::cout << "dv * 3. = \n" << dv2 * 3. << std::endl;
    std::cout << "dv3 = dv/2. = \n" << dv3 << std::endl;

    std::cout << "fv * fv = \n" << fv * fv << std::endl;
    std::cout << "dv3.twonorm() = " << dv3.twonorm() << std::endl;
    std::cout << "dot(dv, dv2)  = " << dot(dv, dv2) << "\n\n";

    return 0;
}
#endif
```

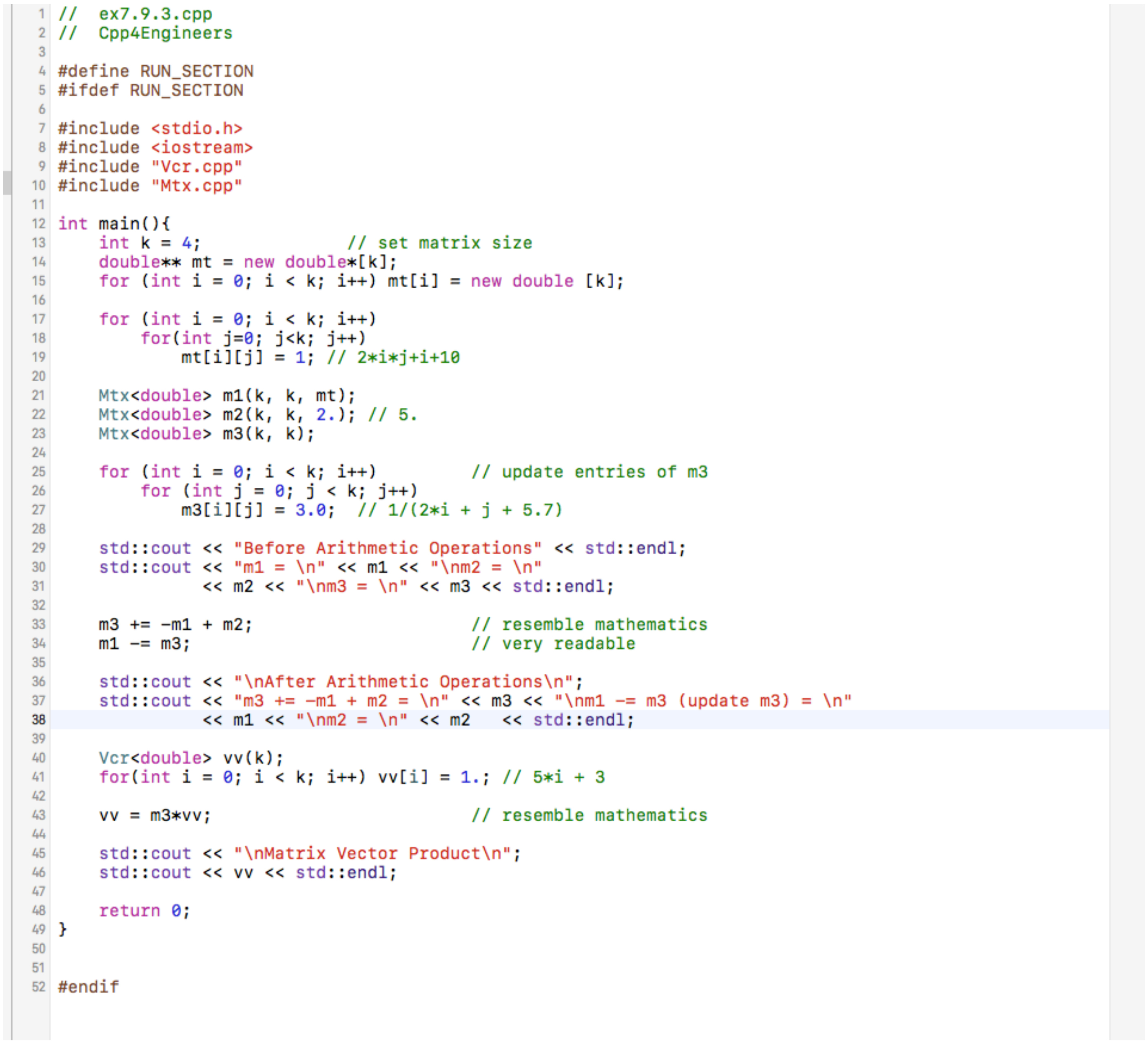
```
dv += dv2 =
5
5
5

dv * 3. =
9
9
9

dv3 = dv/2. =
2.5
2.5
2.5

fv * fv =
4
4
4
4

dv3.twonorm() = 4.33013
dot(dv, dv2)  = 45

Program ended with exit code: 0
```

Question 7.9.3

The matrix class defined in §6.3 was implemented as a template class. The **Mtx** class template header file and definition file are named “**Mtx.h**” and “**Mtx.cpp**” respectively. However, the template implementation in these files also contain template specialization for complex case involving **Question 7.9.15**.

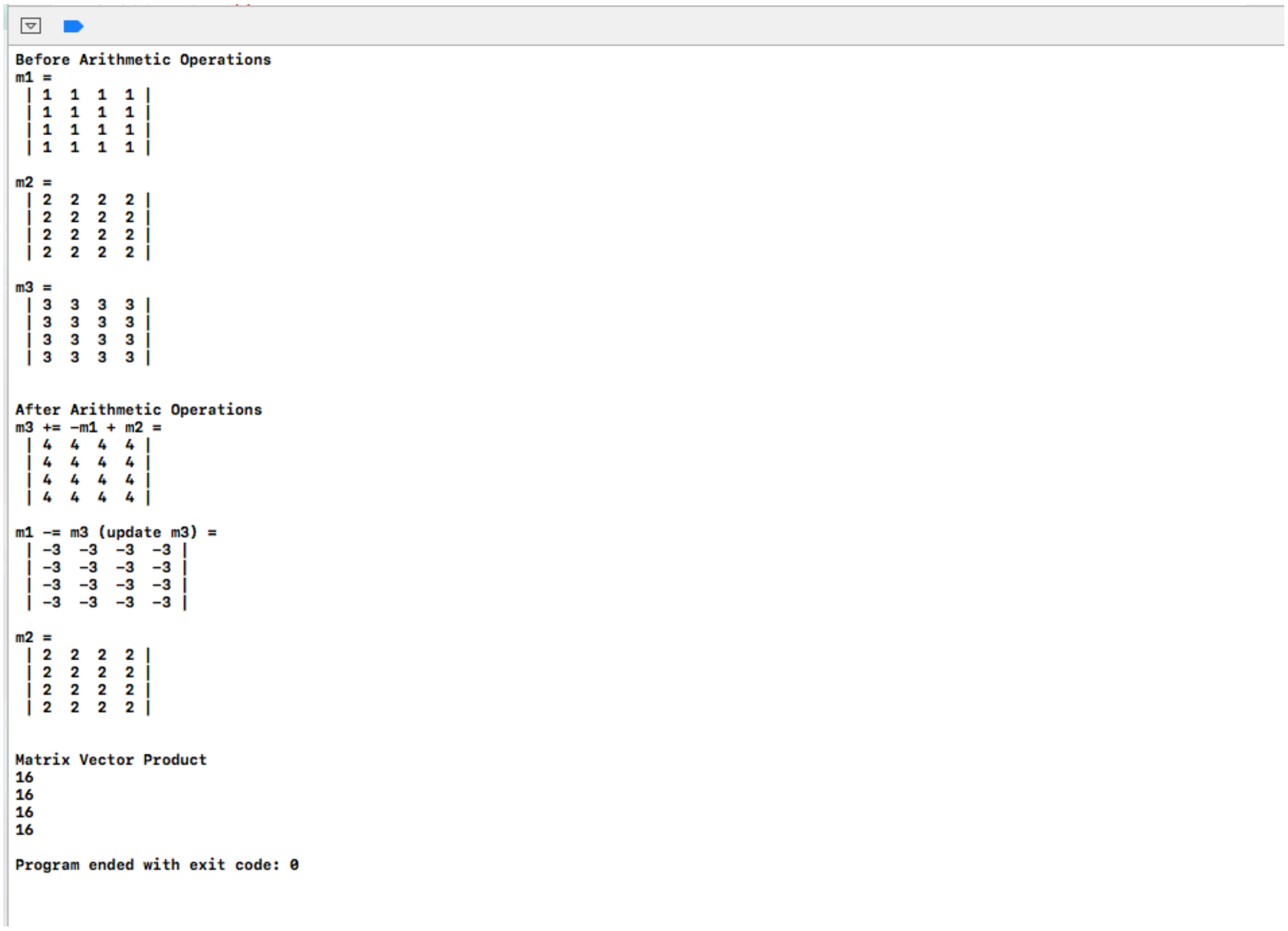
The driver code – named “**ex7.9.3.cpp**” - and the output window are shown in the figures below:

Driver Program:



```cpp
//  ex7.9.3.cpp
//  Cpp4Engineers

#define RUN_SECTION
#ifdef RUN_SECTION

#include <stdio.h>
#include <iostream>
#include "Vcr.cpp"
#include "Mtx.cpp"

int main(){
    int k = 4;              // set matrix size
    double** mt = new double*[k];
    for (int i = 0; i < k; i++) mt[i] = new double [k];

    for (int i = 0; i < k; i++)
        for(int j=0; j<k; j++)
            mt[i][j] = 1; // 2*i*j+i+10

    Mtx<double> m1(k, k, mt);
    Mtx<double> m2(k, k, 2.); // 5.
    Mtx<double> m3(k, k);

    for (int i = 0; i < k; i++)         // update entries of m3
        for (int j = 0; j < k; j++)
            m3[i][j] = 3.0;  // 1/(2*i + j + 5.7)

    std::cout << "Before Arithmetic Operations" << std::endl;
    std::cout << "m1 = \n" << m1 << "\nm2 = \n"
              << m2 << "\nm3 = \n" << m3 << std::endl;

    m3 += -m1 + m2;                     // resemble mathematics
    m1 -= m3;                           // very readable

    std::cout << "\nAfter Arithmetic Operations\n";
    std::cout << "m3 += -m1 + m2 = \n" << m3 << "\nm1 -= m3 (update m3) = \n"
              << m1 << "\nm2 = \n" << m2   << std::endl;

    Vcr<double> vv(k);
    for(int i = 0; i < k; i++) vv[i] = 1.; // 5*i + 3

    vv = m3*vv;                         // resemble mathematics

    std::cout << "\nMatrix Vector Product\n";
    std::cout << vv << std::endl;

    return 0;
}


#endif
```

Output Window:

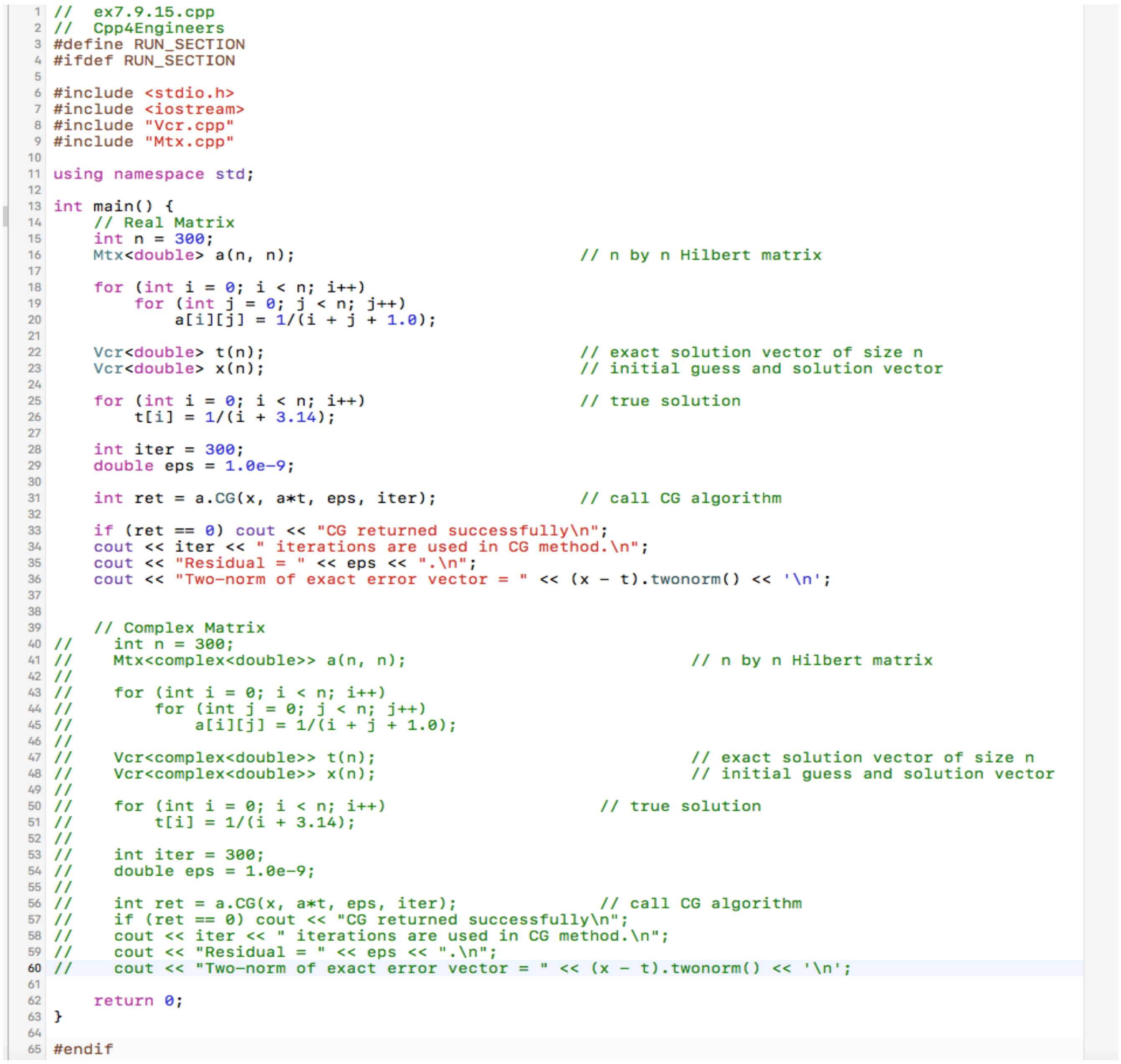


```
Before Arithmetic Operations
m1 =
 | 1  1  1  1 |
 | 1  1  1  1 |
 | 1  1  1  1 |
 | 1  1  1  1 |

m2 =
 | 2  2  2  2 |
 | 2  2  2  2 |
 | 2  2  2  2 |
 | 2  2  2  2 |

m3 =
 | 3  3  3  3 |
 | 3  3  3  3 |
 | 3  3  3  3 |
 | 3  3  3  3 |


After Arithmetic Operations
m3 += -m1 + m2 =
 | 4  4  4  4 |
 | 4  4  4  4 |
 | 4  4  4  4 |
 | 4  4  4  4 |

m1 -= m3 (update m3) =
 | -3  -3  -3  -3 |
 | -3  -3  -3  -3 |
 | -3  -3  -3  -3 |
 | -3  -3  -3  -3 |

m2 =
 | 2  2  2  2 |
 | 2  2  2  2 |
 | 2  2  2  2 |
 | 2  2  2  2 |


Matrix Vector Product
16
16
16
16

Program ended with exit code: 0
```

Question 7.9.15

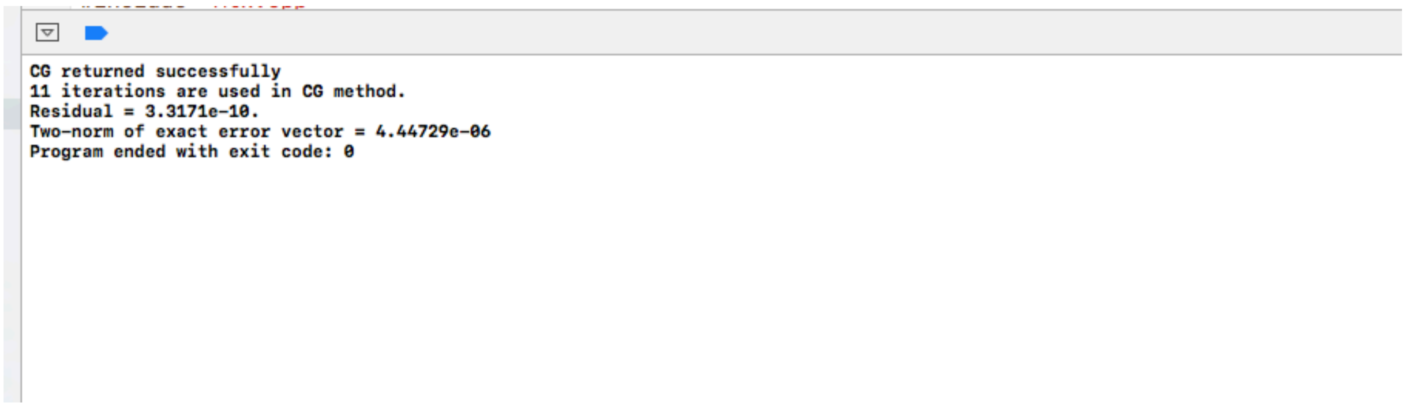
The implementation of this question is partially contained in the "**Vcr.cpp**" and "**Mtx.cpp**". In both of these files the complex specialization of vector operations and complex specialization of matrix operations are implemented for simplicity – so as to avoid duplicate codes. Again, Conjugate Gradient, CG, is implemented in the "**Mtx**" template class for both the real and complex precisions.

Therefore, the file named "**ex7.9.15.cpp**" is the driver program for this question. The driver program is coded to test for both real and complex cases with known solution as depicted in the C++ Textbook on page 228 (Section 6.6). Below is the driver's screenshot:



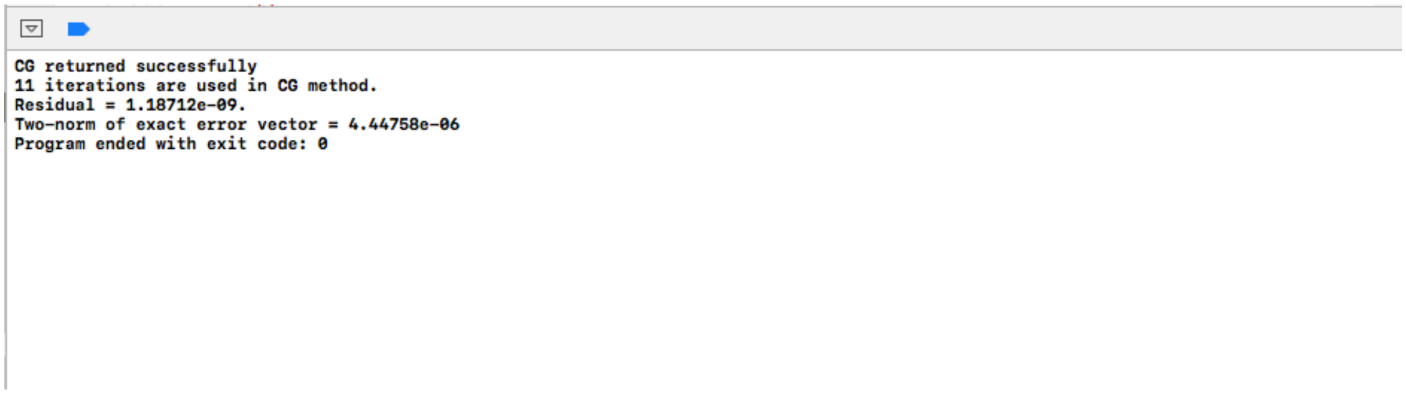
```cpp
//  ex7.9.15.cpp
//  Cpp4Engineers
#define RUN_SECTION
#ifdef RUN_SECTION

#include <stdio.h>
#include <iostream>
#include "Vcr.cpp"
#include "Mtx.cpp"

using namespace std;

int main() {
    // Real Matrix
    int n = 300;
    Mtx<double> a(n, n);                                // n by n Hilbert matrix

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            a[i][j] = 1/(i + j + 1.0);

    Vcr<double> t(n);                                   // exact solution vector of size n
    Vcr<double> x(n);                                   // initial guess and solution vector

    for (int i = 0; i < n; i++)                         // true solution
        t[i] = 1/(i + 3.14);

    int iter = 300;
    double eps = 1.0e-9;

    int ret = a.CG(x, a*t, eps, iter);                  // call CG algorithm

    if (ret == 0) cout << "CG returned successfully\n";
    cout << iter << " iterations are used in CG method.\n";
    cout << "Residual = " << eps << ".\n";
    cout << "Two-norm of exact error vector = " << (x - t).twonorm() << '\n';


    // Complex Matrix
//    int n = 300;
//    Mtx<complex<double>> a(n, n);                               // n by n Hilbert matrix
//
//    for (int i = 0; i < n; i++)
//        for (int j = 0; j < n; j++)
//            a[i][j] = 1/(i + j + 1.0);
//
//    Vcr<complex<double>> t(n);                                  // exact solution vector of size n
//    Vcr<complex<double>> x(n);                                  // initial guess and solution vector
//
//    for (int i = 0; i < n; i++)                        // true solution
//        t[i] = 1/(i + 3.14);
//
//    int iter = 300;
//    double eps = 1.0e-9;
//
//    int ret = a.CG(x, a*t, eps, iter);                 // call CG algorithm
//    if (ret == 0) cout << "CG returned successfully\n";
//    cout << iter << " iterations are used in CG method.\n";
//    cout << "Residual = " << eps << ".\n";
//    cout << "Two-norm of exact error vector = " << (x - t).twonorm() << '\n';

    return 0;
}

#endif
```

Output Screen for Real Matrix:



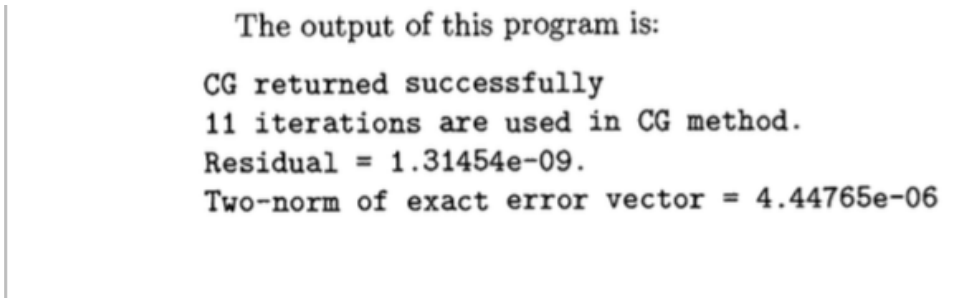
```
CG returned successfully
11 iterations are used in CG method.
Residual = 3.3171e-10.
Two-norm of exact error vector = 4.44729e-06
Program ended with exit code: 0
```

Output Screen for Complex Matrix with zero (0) imaginary part for the same problem (as the above):



```
CG returned successfully
11 iterations are used in CG method.
Residual = 1.18712e-09.
Two-norm of exact error vector = 4.44758e-06
Program ended with exit code: 0
```

The result obtained by the author of the textbook as contained in page 228 is:



The output of this program is:

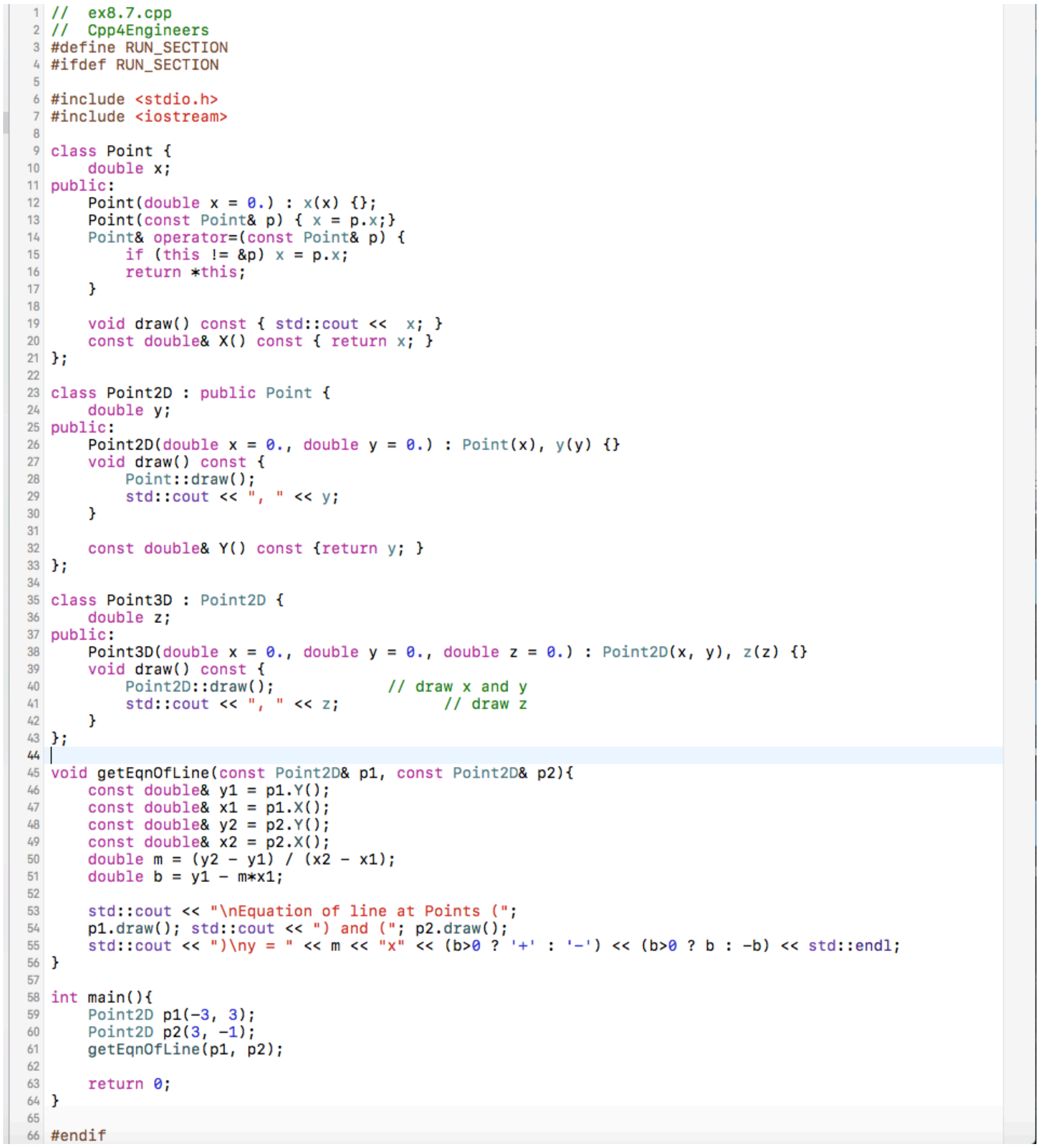
```
CG returned successfully
11 iterations are used in CG method.
Residual = 1.31454e-09.
Two-norm of exact error vector = 4.44765e-06
```

QUESTION (8)

All codes as regards to this question 8 are contained in folder named “**Chapter8**”.

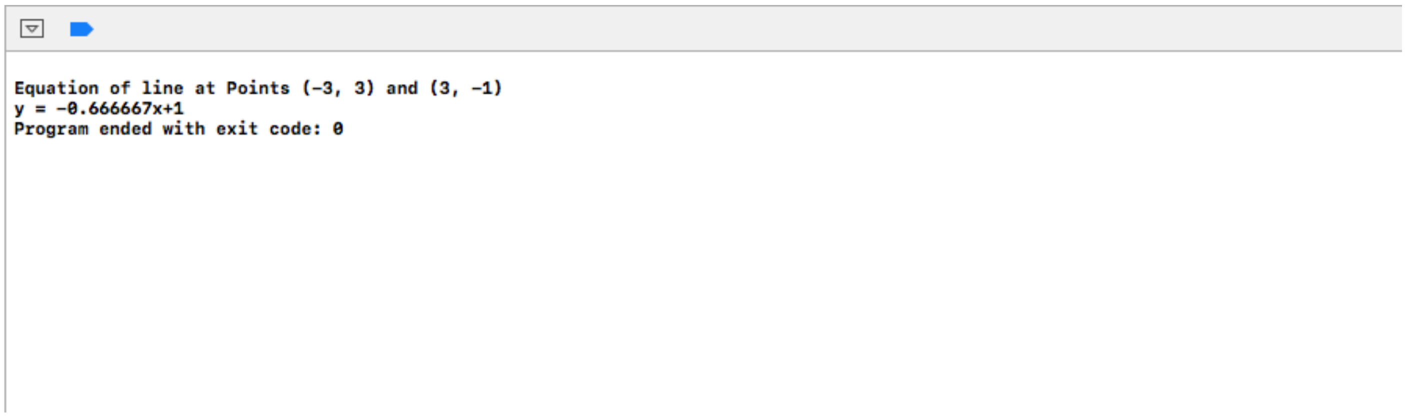
Question 8.7.1

The class hierarchy for points 1D, 2D and 3D are implemented in the file named “**ex8.7.1.cpp**”. As stated in the question, the function for finding equation of line passing through two (2) points is also implemented in the same file. This additional function solves the equation of line and prints out the resulting equation on the screen. The implementations and driver codes are shown below:



```cpp
//  ex8.7.cpp
//  Cpp4Engineers
#define RUN_SECTION
#ifdef RUN_SECTION

#include <stdio.h>
#include <iostream>

class Point {
    double x;
public:
    Point(double x = 0.) : x(x) {};
    Point(const Point& p) { x = p.x;}
    Point& operator=(const Point& p) {
        if (this != &p) x = p.x;
        return *this;
    }

    void draw() const { std::cout <<  x; }
    const double& X() const { return x; }
};

class Point2D : public Point {
    double y;
public:
    Point2D(double x = 0., double y = 0.) : Point(x), y(y) {}
    void draw() const {
        Point::draw();
        std::cout << ", " << y;
    }

    const double& Y() const {return y; }
};

class Point3D : Point2D {
    double z;
public:
    Point3D(double x = 0., double y = 0., double z = 0.) : Point2D(x, y), z(z) {}
    void draw() const {
        Point2D::draw();            // draw x and y
        std::cout << ", " << z;           // draw z
    }
};

void getEqnOfLine(const Point2D& p1, const Point2D& p2){
    const double& y1 = p1.Y();
    const double& x1 = p1.X();
    const double& y2 = p2.Y();
    const double& x2 = p2.X();
    double m = (y2 - y1) / (x2 - x1);
    double b = y1 - m*x1;

    std::cout << "\nEquation of line at Points (";
    p1.draw(); std::cout << ") and ("; p2.draw();
    std::cout << ")\ny = " << m << "x" << (b>0 ? '+' : '-') << (b>0 ? b : -b) << std::endl;
}

int main(){
    Point2D p1(-3, 3);
    Point2D p2(3, -1);
    getEqnOfLine(p1, p2);

    return 0;
}

#endif
```

Output Window:



```
Equation of line at Points (-3, 3) and (3, -1)
y = -0.666667x+1
Program ended with exit code: 0
```

Question 8.7.3

A virtual destructor is a virtual function defined in the base class of a derived class. A virtual destructor is used such that proper cleanup in the allocated memory is achieved. Such use is required when an object or a class of the derived class is manipulated through the base class's pointer and both the superclass and sub-class have dynamic memory allocations; This implies that there is run-time polymorphism. However, there is no need of virtual destructor if and only if the derived class is not manipulated through the base class's pointer; In this case memory allocation will be properly deallocated.

Concisely, a class with run-time polymorphism (virtual functions) needs to have virtual destructor.

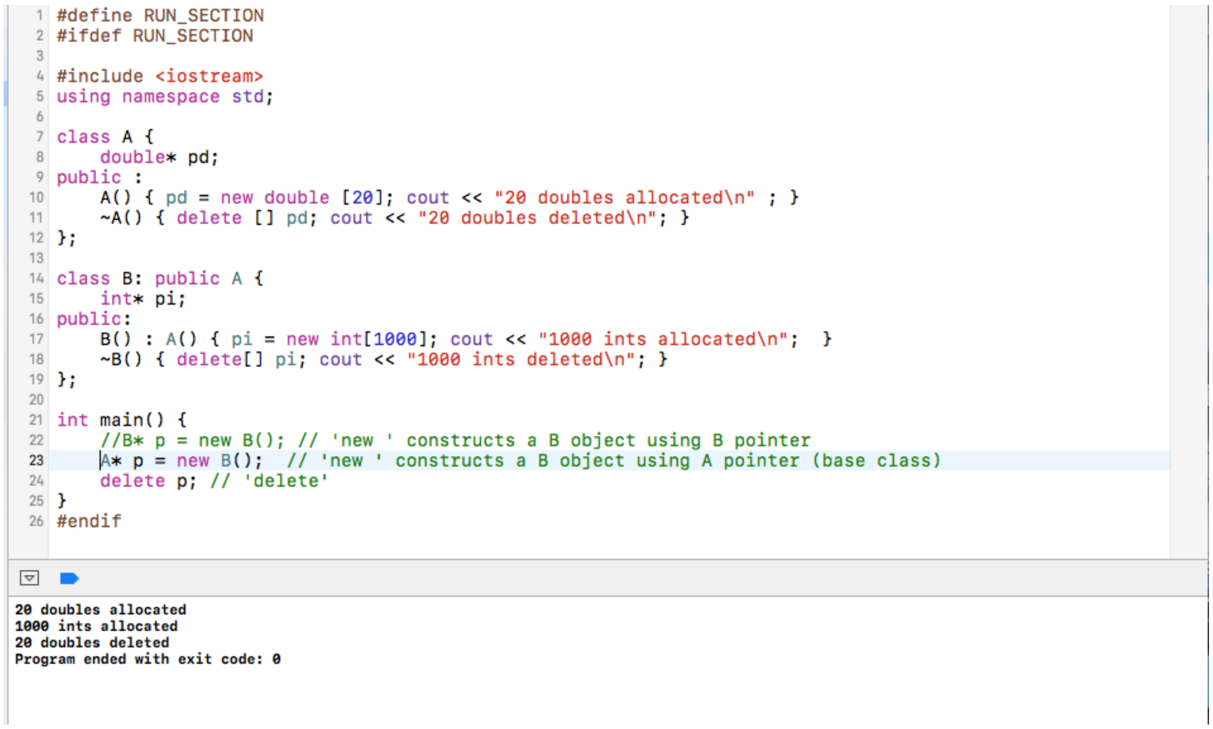
Example: Consider a derived class B having a base class A such that memory is allocated in both A and B. Expression like:

```
A* obj = new B( );  // manipulation of derived class through the base class
delete obj;         // proper cleanups are not achieved here if the base destructor is not virtual
```

To achieve the desired memory deallocation, the destructor in base class A should be made virtual and as such proper cleanups can be achieved in B.

To demonstrate the above assertions, a short driver as well as classes programs was written as contained in the file named "**ex8.7.3.cpp**".

CASE 1: When an object of the derived class is manipulated by the base class's pointer but the destructor of the base class is not a virtual function.

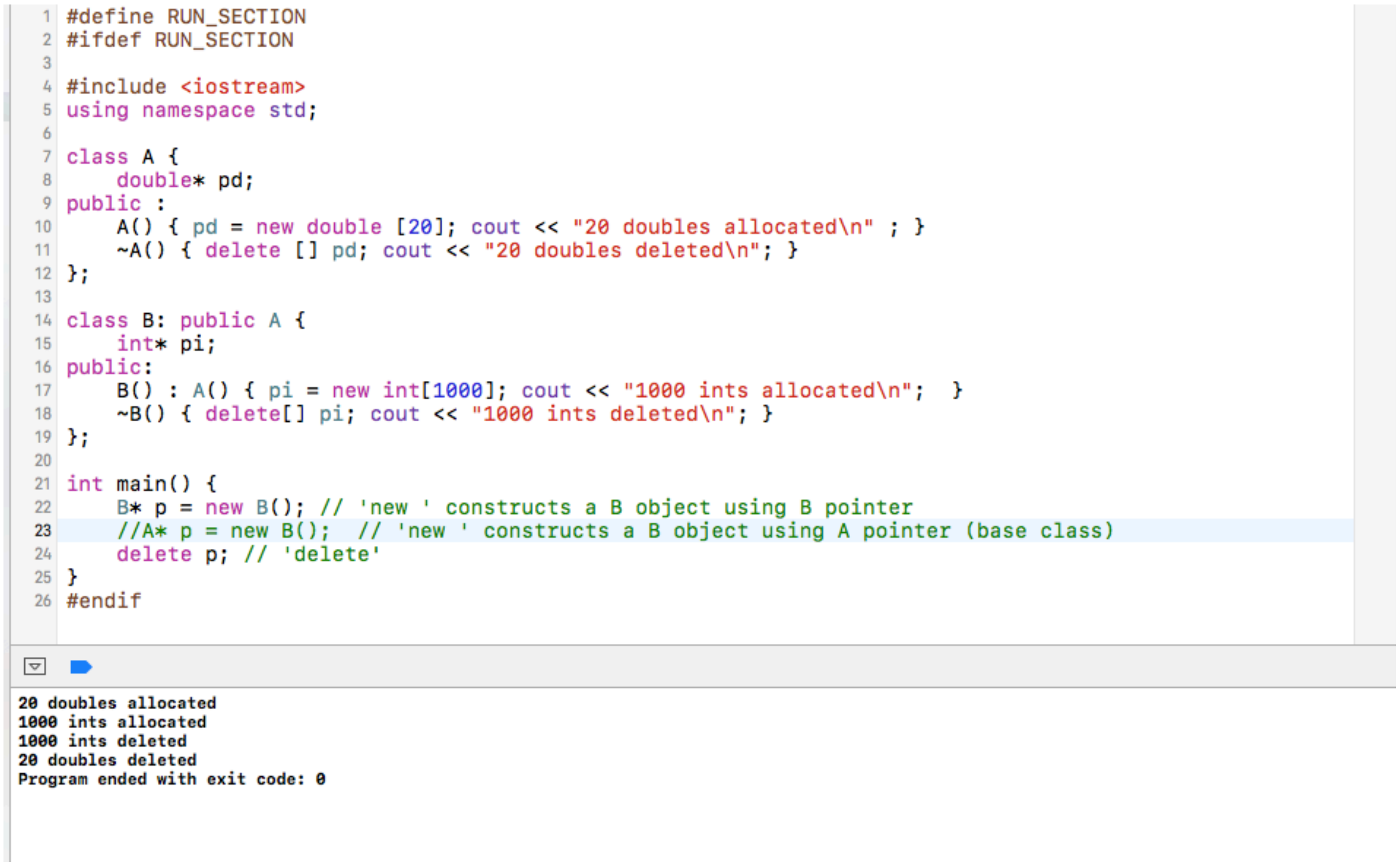


```cpp
#define RUN_SECTION
#ifdef RUN_SECTION

#include <iostream>
using namespace std;

class A {
    double* pd;
public :
    A() { pd = new double [20]; cout << "20 doubles allocated\n" ; }
    ~A() { delete [] pd; cout << "20 doubles deleted\n"; }
};

class B: public A {
    int* pi;
public:
    B() : A() { pi = new int[1000]; cout << "1000 ints allocated\n";  }
    ~B() { delete[] pi; cout << "1000 ints deleted\n"; }
};

int main() {
    //B* p = new B(); // 'new ' constructs a B object using B pointer
    A* p = new B();  // 'new ' constructs a B object using A pointer (base class)
    delete p; // 'delete'
}
#endif
```

```
20 doubles allocated
1000 ints allocated
20 doubles deleted
Program ended with exit code: 0
```

Note: Not all allocated memory spaces are freed - Improper cleanup.

CASE 2: When an object of the derived class is manipulated by the derived class's pointer but the destructor of the base class is not a virtual function.

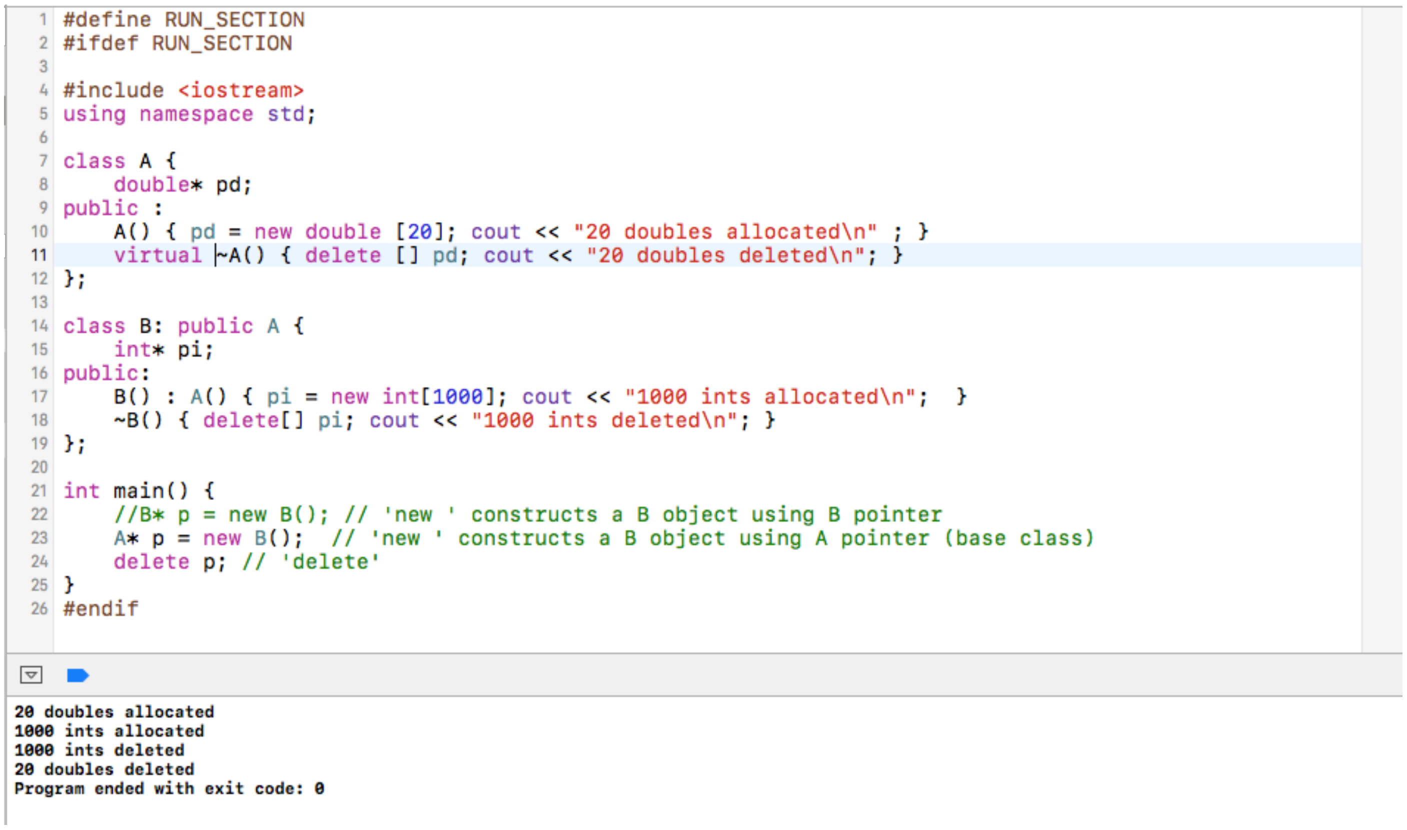


```cpp
#define RUN_SECTION
#ifdef RUN_SECTION

#include <iostream>
using namespace std;

class A {
    double* pd;
public :
    A() { pd = new double [20]; cout << "20 doubles allocated\n" ; }
    ~A() { delete [] pd; cout << "20 doubles deleted\n"; }
};

class B: public A {
    int* pi;
public:
    B() : A() { pi = new int[1000]; cout << "1000 ints allocated\n";  }
    ~B() { delete[] pi; cout << "1000 ints deleted\n"; }
};

int main() {
    B* p = new B(); // 'new ' constructs a B object using B pointer
    //A* p = new B();  // 'new ' constructs a B object using A pointer (base class)
    delete p; // 'delete'
}
#endif
```

```
20 doubles allocated
1000 ints allocated
1000 ints deleted
20 doubles deleted
Program ended with exit code: 0
```

Note: Here proper cleanups was achieved as all allocated memory spaces got freed.

CASE 3: When an object of the derived class is manipulated by the base class's pointer and the destructor of the base class is a virtual function.



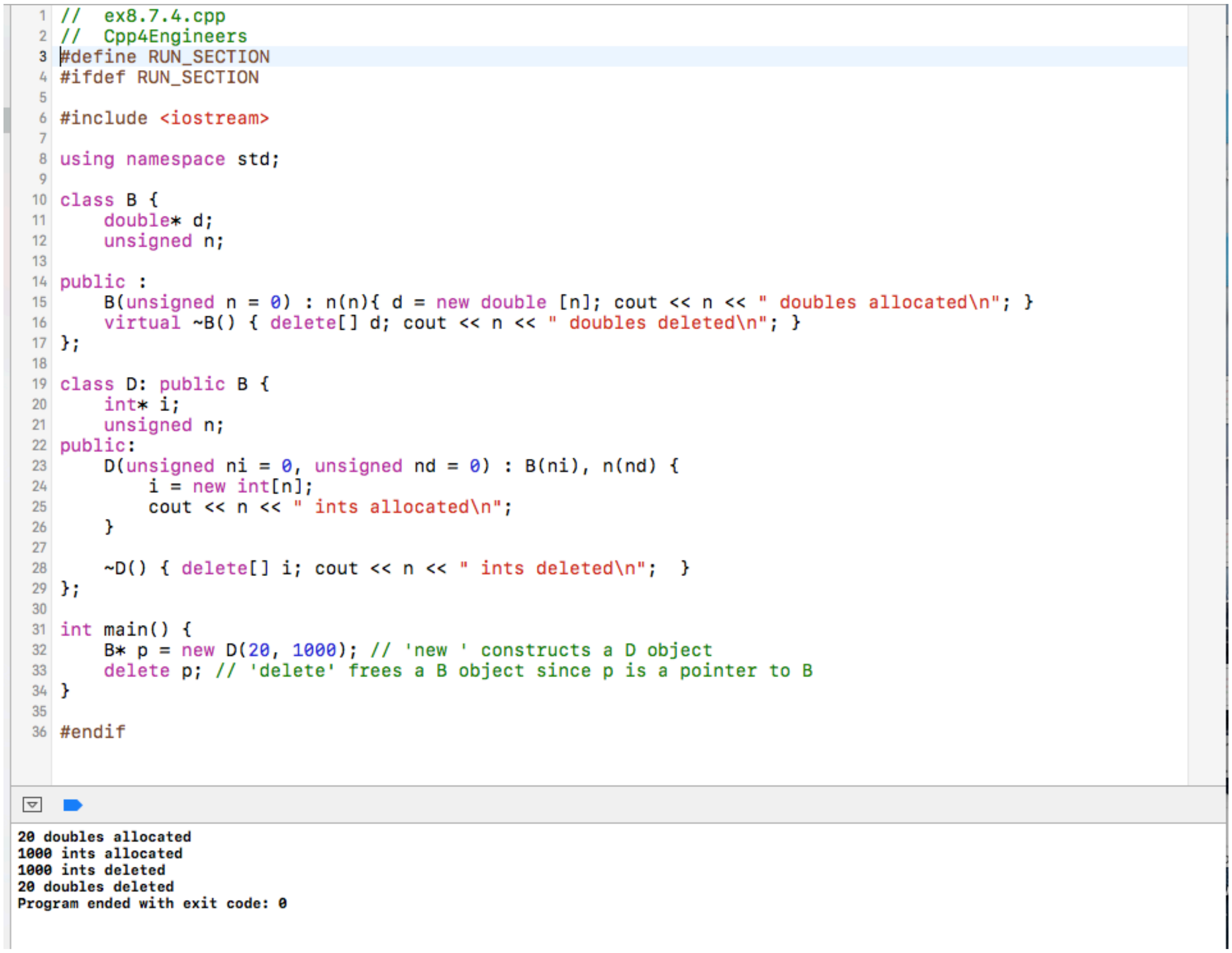
```cpp
#define RUN_SECTION
#ifdef RUN_SECTION

#include <iostream>
using namespace std;

class A {
    double* pd;
public :
    A() { pd = new double [20]; cout << "20 doubles allocated\n" ; }
    virtual ~A() { delete [] pd; cout << "20 doubles deleted\n"; }
};

class B: public A {
    int* pi;
public:
    B() : A() { pi = new int[1000]; cout << "1000 ints allocated\n";  }
    ~B() { delete[] pi; cout << "1000 ints deleted\n"; }
};

int main() {
    //B* p = new B(); // 'new ' constructs a B object using B pointer
    A* p = new B();  // 'new ' constructs a B object using A pointer (base class)
    delete p; // 'delete'
}
#endif
```

```
20 doubles allocated
1000 ints allocated
1000 ints deleted
20 doubles deleted
Program ended with exit code: 0
```

Note: Here proper cleanups was achieved as all allocated memory spaces got freed.

Question 8.7.4

“**ex8.7.4.cpp**” is the implementation file of this question. Below is the code and the output screen.



```
//  ex8.7.4.cpp
//  Cpp4Engineers
#define RUN_SECTION
#ifdef RUN_SECTION

#include <iostream>

using namespace std;

class B {
    double* d;
    unsigned n;

public :
    B(unsigned n = 0) : n(n){ d = new double [n]; cout << n << " doubles allocated\n"; }
    virtual ~B() { delete[] d; cout << n << " doubles deleted\n"; }
};

class D: public B {
    int* i;
    unsigned n;
public:
    D(unsigned ni = 0, unsigned nd = 0) : B(ni), n(nd) {
        i = new int[n];
        cout << n << " ints allocated\n";
    }

    ~D() { delete[] i; cout << n << " ints deleted\n";  }
};

int main() {
    B* p = new D(20, 1000); // 'new ' constructs a D object
    delete p; // 'delete' frees a B object since p is a pointer to B
}

#endif
```

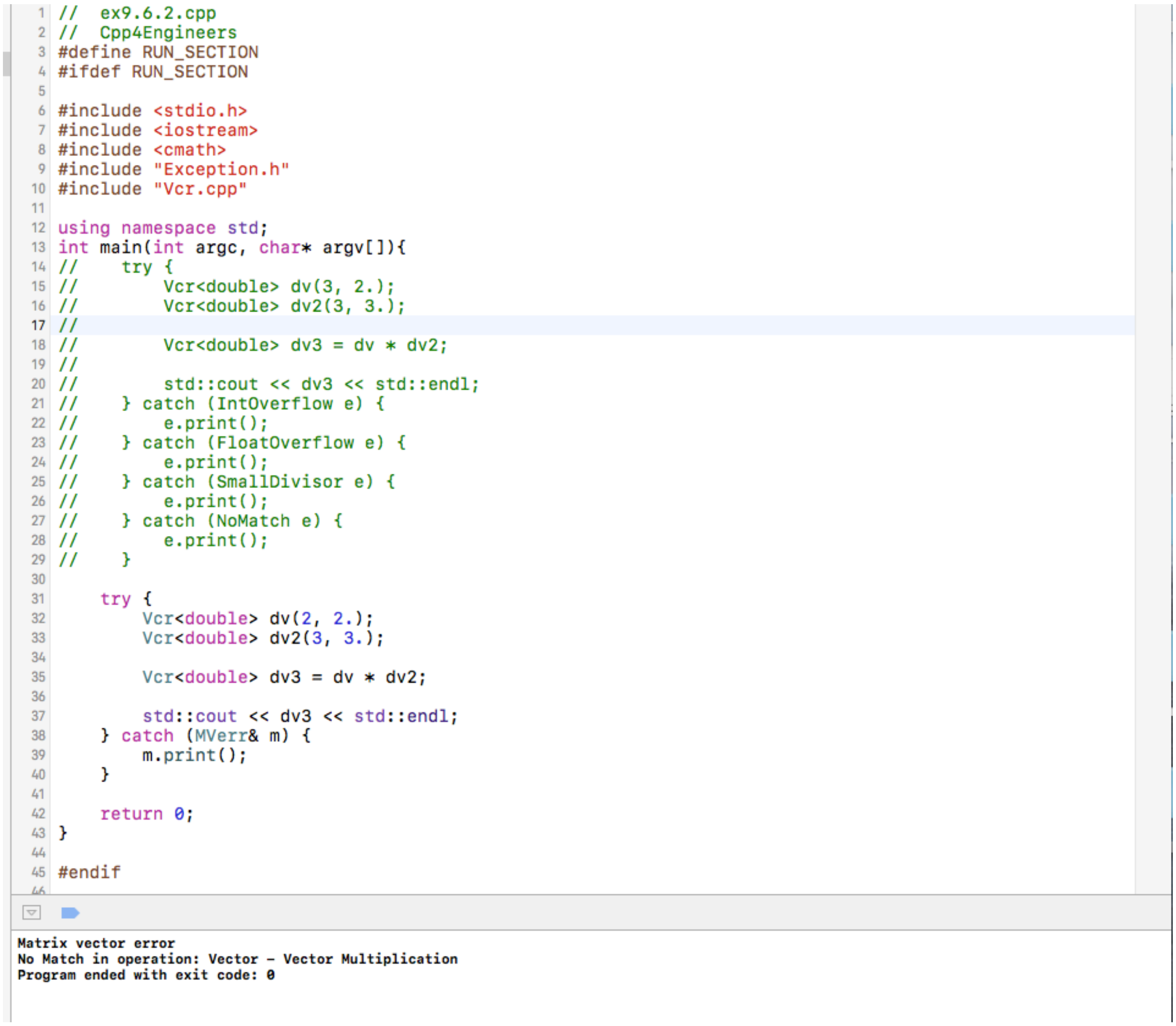
```
20 doubles allocated
1000 ints allocated
1000 ints deleted
20 doubles deleted
Program ended with exit code: 0
```

QUESTIONS (9 & 10)

All codes as regards to these questions are contained in folder named “**Chapter9_10**”.

Question 9.6.2

Vector template for throwing an exception was implemented in “**Vcr.h**” and “**Vcr.cpp**”. While the class hierarchical exception handler is implemented in the file named “**Exception.h**”. The driver program for this question is named “**ex9.6.2.cpp**”. Here, there is a naïve implementation (commented part) and efficient implementation using catch with base class’s object reference.

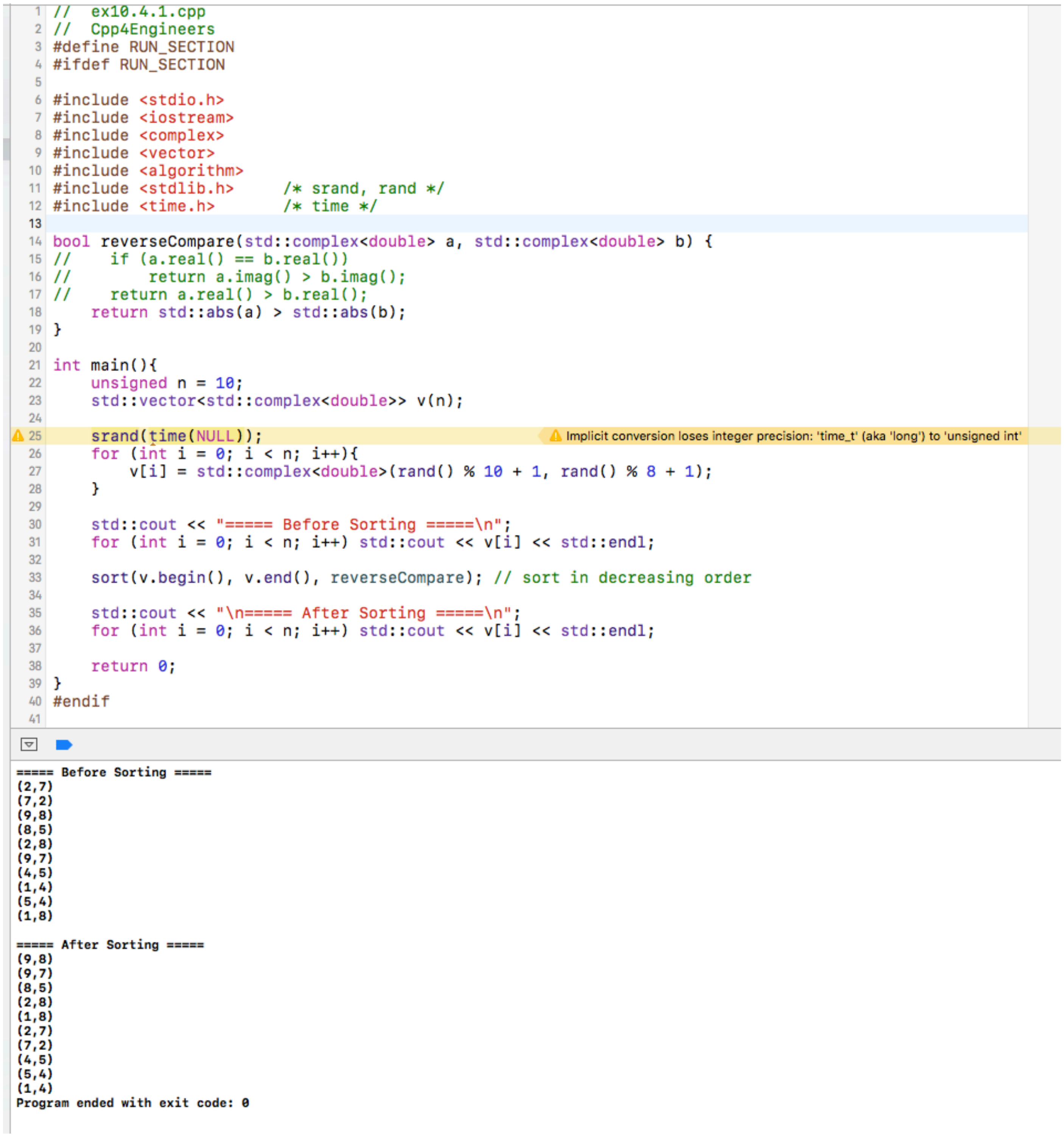


```cpp
//  ex9.6.2.cpp
//  Cpp4Engineers
#define RUN_SECTION
#ifdef RUN_SECTION

#include <stdio.h>
#include <iostream>
#include <cmath>
#include "Exception.h"
#include "Vcr.cpp"

using namespace std;
int main(int argc, char* argv[]){
//    try {
//        Vcr<double> dv(3, 2.);
//        Vcr<double> dv2(3, 3.);
//
//        Vcr<double> dv3 = dv * dv2;
//
//        std::cout << dv3 << std::endl;
//    } catch (IntOverflow e) {
//        e.print();
//    } catch (FloatOverflow e) {
//        e.print();
//    } catch (SmallDivisor e) {
//        e.print();
//    } catch (NoMatch e) {
//        e.print();
//    }

    try {
        Vcr<double> dv(2, 2.);
        Vcr<double> dv2(3, 3.);

        Vcr<double> dv3 = dv * dv2;

        std::cout << dv3 << std::endl;
    } catch (MVerr& m) {
        m.print();
    }

    return 0;
}

#endif
```

```
Matrix vector error
No Match in operation: Vector - Vector Multiplication
Program ended with exit code: 0
```

Question 10.4.1

The implementation and driver program for this question is in the file named "**ex10.4.1.cpp**". Below is the output screen and the driver program.

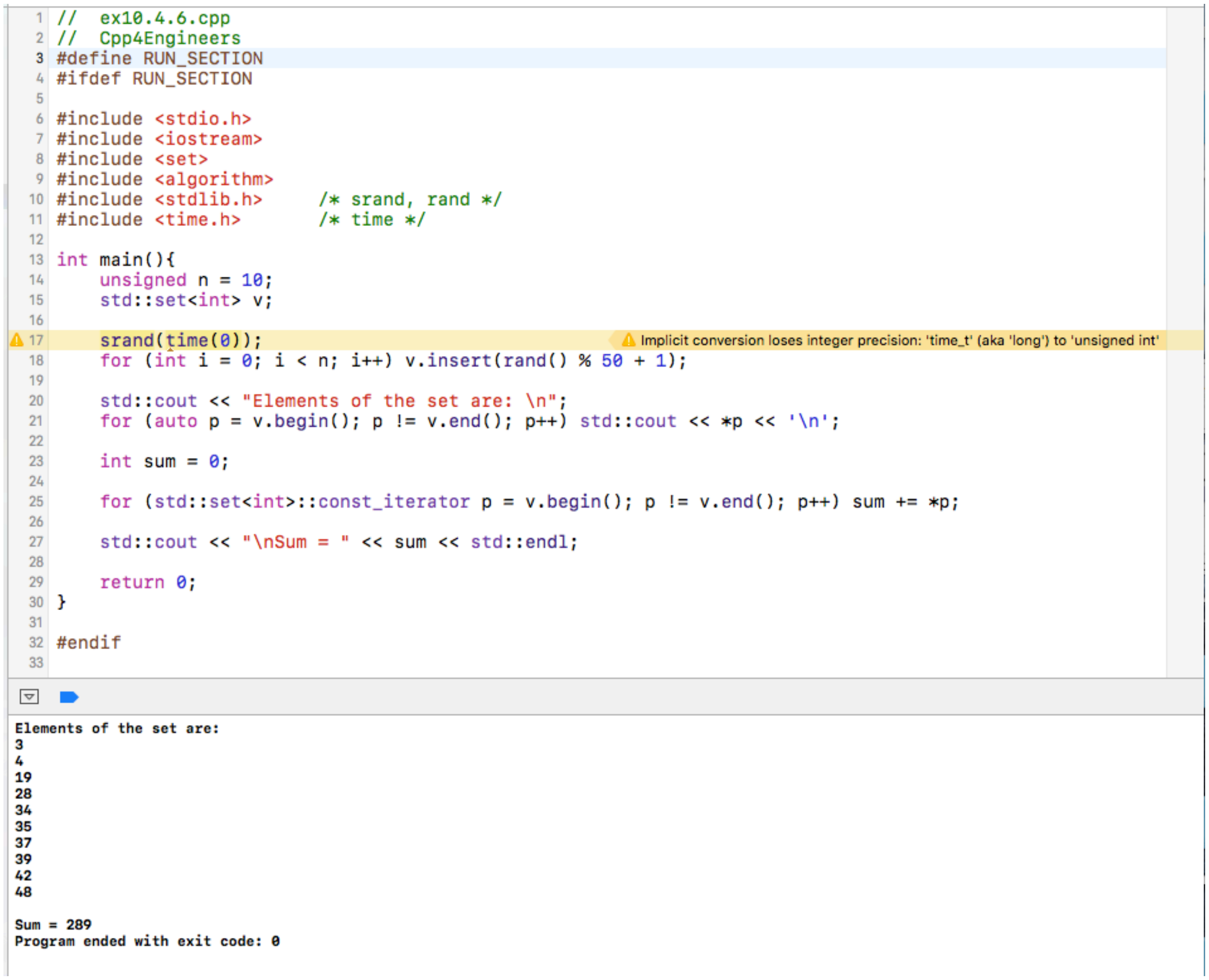


```
//   ex10.4.1.cpp
//   Cpp4Engineers
#define RUN_SECTION
#ifdef RUN_SECTION

#include <stdio.h>
#include <iostream>
#include <complex>
#include <vector>
#include <algorithm>
#include <stdlib.h>       /* srand, rand */
#include <time.h>         /* time */

bool reverseCompare(std::complex<double> a, std::complex<double> b) {
//    if (a.real() == b.real())
//        return a.imag() > b.imag();
//    return a.real() > b.real();
    return std::abs(a) > std::abs(b);
}

int main(){
    unsigned n = 10;
    std::vector<std::complex<double>> v(n);

    srand(time(NULL));                                    Implicit conversion loses integer precision: 'time_t' (aka 'long') to 'unsigned int'
    for (int i = 0; i < n; i++){
        v[i] = std::complex<double>(rand() % 10 + 1, rand() % 8 + 1);
    }

    std::cout << "===== Before Sorting =====\n";
    for (int i = 0; i < n; i++) std::cout << v[i] << std::endl;

    sort(v.begin(), v.end(), reverseCompare); // sort in decreasing order

    std::cout << "\n===== After Sorting =====\n";
    for (int i = 0; i < n; i++) std::cout << v[i] << std::endl;

    return 0;
}
#endif
```

```
===== Before Sorting =====
(2,7)
(7,2)
(9,8)
(8,5)
(2,8)
(9,7)
(4,5)
(1,4)
(5,4)
(1,8)

===== After Sorting =====
(9,8)
(9,7)
(8,5)
(2,8)
(1,8)
(2,7)
(7,2)
(4,5)
(5,4)
(1,4)
Program ended with exit code: 0
```

Question 10.4.6

The implementation and driver program for this question is in the file named “**ex10.4.6.cpp**”. Below is the driver program as well as the output screen.



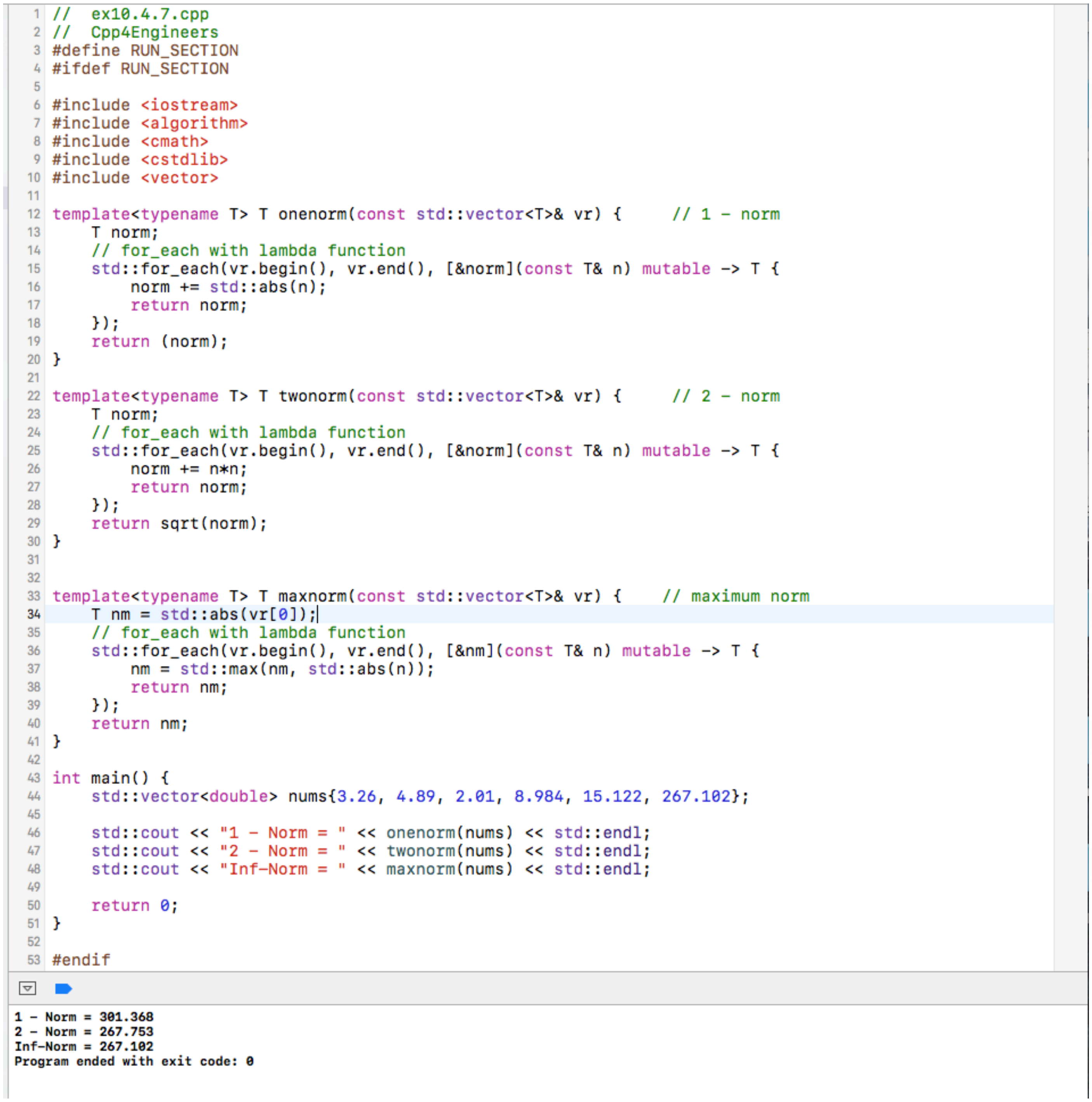
```cpp
//  ex10.4.6.cpp
//  Cpp4Engineers
#define RUN_SECTION
#ifdef RUN_SECTION

#include <stdio.h>
#include <iostream>
#include <set>
#include <algorithm>
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */

int main(){
    unsigned n = 10;
    std::set<int> v;

    srand(time(0));
    for (int i = 0; i < n; i++) v.insert(rand() % 50 + 1);

    std::cout << "Elements of the set are: \n";
    for (auto p = v.begin(); p != v.end(); p++) std::cout << *p << '\n';

    int sum = 0;

    for (std::set<int>::const_iterator p = v.begin(); p != v.end(); p++) sum += *p;

    std::cout << "\nSum = " << sum << std::endl;

    return 0;
}

#endif
```

⚠ Implicit conversion loses integer precision: 'time_t' (aka 'long') to 'unsigned int'

```
Elements of the set are:
3
4
19
28
34
35
37
39
42
48

Sum = 289
Program ended with exit code: 0
```

Question 10.4.7

The implementation and driver program for this question is in the file named “**ex10.4.7.cpp**”. Below is the driver program as well as the output screen.



```cpp
//  ex10.4.7.cpp
//  Cpp4Engineers
#define RUN_SECTION
#ifdef RUN_SECTION

#include <iostream>
#include <algorithm>
#include <cmath>
#include <cstdlib>
#include <vector>

template<typename T> T onenorm(const std::vector<T>& vr) {     // 1 - norm
    T norm;
    // for_each with lambda function
    std::for_each(vr.begin(), vr.end(), [&norm](const T& n) mutable -> T {
        norm += std::abs(n);
        return norm;
    });
    return (norm);
}

template<typename T> T twonorm(const std::vector<T>& vr) {     // 2 - norm
    T norm;
    // for_each with lambda function
    std::for_each(vr.begin(), vr.end(), [&norm](const T& n) mutable -> T {
        norm += n*n;
        return norm;
    });
    return sqrt(norm);
}


template<typename T> T maxnorm(const std::vector<T>& vr) {    // maximum norm
    T nm = std::abs(vr[0]);
    // for_each with lambda function
    std::for_each(vr.begin(), vr.end(), [&nm](const T& n) mutable -> T {
        nm = std::max(nm, std::abs(n));
        return nm;
    });
    return nm;
}

int main() {
    std::vector<double> nums{3.26, 4.89, 2.01, 8.984, 15.122, 267.102};

    std::cout << "1 - Norm = " << onenorm(nums) << std::endl;
    std::cout << "2 - Norm = " << twonorm(nums) << std::endl;
    std::cout << "Inf-Norm = " << maxnorm(nums) << std::endl;

    return 0;
}

#endif
```

```
1 - Norm = 301.368
2 - Norm = 267.753
Inf-Norm = 267.102
Program ended with exit code: 0
```

References

Yang, D. (2012). C and object-oriented numeric computing for scientists and engineers. New York: Springer.