

C++ Programming Workshop

Updated on 30 March 2016

By

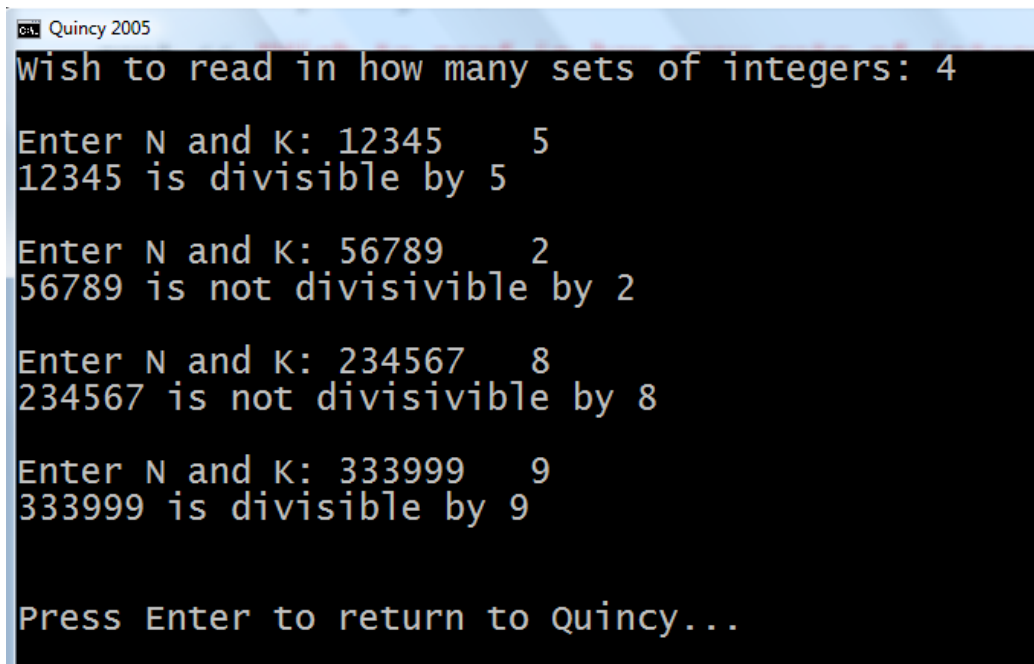
Heng Aik Koan

Workshop 1 – Basic C++ features

A positive integer N is divisible by K if and only if the sum of digits is divisible by K .

For example, if $N = 12345$ and $K = 5$, the sum of digits of N is $1 + 2 + 3 + 4 + 5 = 15$, and $15 \% 5 = 0$; therefore 12345 is divisible by 5.

Design a C++ program, read in a few positive integers N and a few K , and check if each integer read is divisible by the K read. The following shows one of the possible interactions and displays:



```
Quincy 2005
Wish to read in how many sets of integers: 4
Enter N and K: 12345    5
12345 is divisible by 5
Enter N and K: 56789    2
56789 is not divisivible by 2
Enter N and K: 234567    8
234567 is not divisivible by 8
Enter N and K: 333999    9
333999 is divisible by 9
Press Enter to return to Quincy...
```

// Work shop 1

```
#include <iostream>
using namespace std;

int main ()
{
    int N, K;
    int howManyIntegers;

    cout << "Wish to read in how many sets of integers: ";
    cin >> howManyIntegers;

    cout << endl;
    for (int i = 1; i <= howManyIntegers; i++)
    {
        cout << "Enter N and K: ";
        cin >> N >> K;

        cin.clear ();
        cin.ignore (100, '\n');

        int sumOfDigits = 0;
        int temp = N;

        while (temp > 0)
        {
            sumOfDigits += temp % 10;
            temp /= 10;
        }

        if (sumOfDigits % K)
            cout << N << " is not divisibile by " << K << endl;
        else
            cout << N << " is divisible by " << K << endl;

        cout << endl;
    }
}
```

Workshop 2 – User defined functions

Convert program written in Workshop 1 using user defined functions, and explore the concepts of functions passing by value and by reference.

// Work shop 2

```
#include <iostream>
```

```
using namespace std;
```

```
void getTwoIntegers (int&, int&);
```

```
int getSumOfDigits (int);
```

```
bool divisibleK (int, int);
```

```
int main ()
```

```
{
```

```
    int N, K;
```

```
    int howManyIntegers;
```

```
    cout << "Wish to read in how many sets of integers: ";
```

```
    cin >> howManyIntegers;
```

```
    cout << endl;
```

```
    for (int i = 1; i <= howManyIntegers; i++)
```

```
    {
```

```
        getTwoIntegers (N, K);
```

```
        int sumOfDigits = getSumOfDigits (N);
```

```
        bool ok = divisibleK (sumOfDigits, K);
```

```
        if (!ok)
```

```
            cout << N << " is not divisivable by " << K << endl;
```

```
        else
```

```
            cout << N << " is divisible by " << K << endl;
```

```
        cout << endl;
```

```
    }
```

```
}
```

```

void getTwoIntegers (int& N, int& K)
{
    cout << "Enter N and K: ";
    cin >> N >> K;
}

int getSumOfDigits (int N)
{
    int sumOfDigits = 0;

    while (N > 0)
    {
        sumOfDigits += N % 10;
        N /= 10;
    }

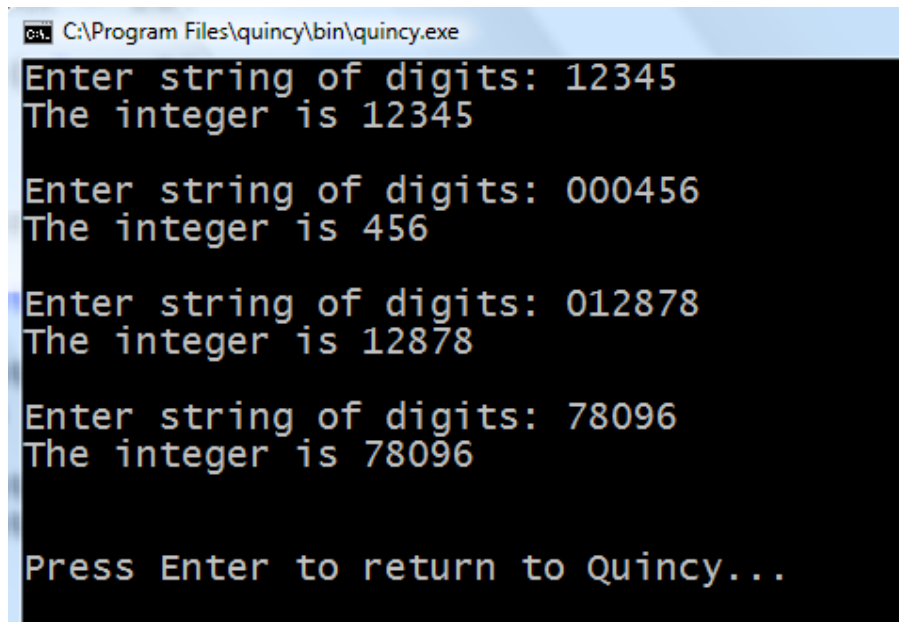
    return sumOfDigits;
}

bool divisibleK (int sum, int K)
{
    if (sum % K)
        return false;
    else
        return true;
}

```

Workshop 3 – Array and C-string

In Java, there is function to convert a string of digits to its integer equivalent. Write a C++ program to read a few strings of digits and to display their corresponding integers' equivalent. The following shows one of the possible interactions:



```
C:\Program Files\quincy\bin\quincy.exe
Enter string of digits: 12345
The integer is 12345

Enter string of digits: 000456
The integer is 456

Enter string of digits: 012878
The integer is 12878

Enter string of digits: 78096
The integer is 78096

Press Enter to return to Quincy...
```

Think of doing the string manipulation rather than looking for some available functions in C language.

// Workshop 3

```
#include <iostream>
#include <cctype>
using namespace std;

const int MAX = 20;

int getInt (char *);

int main ()
{
    char str [MAX];

    for (int i = 1; i <= 4; i++)
    {
        cout << "Enter string of digits: ";
        cin >> str;

        cout << "The integer is " << getInt (str) << endl;
        cout << endl;
    }
}

int getInt (char *str)
{
    int i = 0;
    int digit = 0;

    while (str [i] != '\0')
    {
        digit = digit * 10 + str [i] - '0';
        ++i;
    }

    return digit;
}
```

Workshop 4 – User defined data types

Generate some sets of three integers, we wish to see the status of the third integer and see if this third integer is the biggest, smallest or in between the first two integers.

Explore the use of three enumeration constants, for example, **Biggest**, **Smallest** and **Between** in your design and display them.

The following shows some of the generations and analysis:

```
31945    8695    4924
==> Smallest
15090    14196   29807
==> Biggest
15784    23151   28656
==> Biggest
11358    32501   13182
==> Between
1316     9692    6249
==> Between
16792    1718    27596
==> Biggest
31722    14171    984
==> Smallest
7010     12657    7727
==> Between
15461    8419     12475
==> Between
6441     26668    12716
==> Between
```

// Work shop 4

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

enum Status {Biggest, Smallest, Between};

const int MAX = 80;

void get3Integers (int&, int&, int&);
Status getStatus (int, int, int);
void getStatusLabel (Status, char []);

int main ()
{
    int a, b, c;
    char label [MAX];

    srand (time (NULL));

    for (int i = 1; i <= 10; i++)
    {
        get3Integers (a, b, c);
        Status s = getStatus (a, b, c);
        getStatusLabel (s, label);

        cout << a << "\t" << b << "\t" << c << endl;
        cout << "==> " << label << endl;
    }
}

void get3Integers (int& a, int& b, int& c)
{
    a = rand ();
    b = rand ();
    c = rand ();
}

Status getStatus (int a, int b, int c)
{
    if (c >= a && c >= b)
        return Biggest;
```

```

        else if ((c >= a && c <= b) || (c >= b && c <= a))
            return Between;
        else
            return Smallest;
    }

void getStatusLabel (Status s, char str [ ])
{
    switch (s)
    {
        case Biggest: strcpy (str, "Biggest");
                        break;
        case Smallest: strcpy (str, "Smallest");
                        break;
        default       : strcpy (str, "Between");
    }
}

```

Workshop 5 – File processing

A text file consists of an English essay. Design a C++ program to convert the text file to a binary file; and from the binary file to a text file again. Make sure that the final text file and the original text file is the same.

// workshop 5

```
#include <fstream>
#include <iostream>
using namespace std;

const int MAX = 80;

int main ()
{
    fstream afile;
    ifstream infile;

    char line [80];

    // From text file to binary file
    infile.open ("infile.txt");
    afile.open ("infile.dat", ios::out | ios::binary);

    while (infile.getline (line, MAX))
        afile.write (reinterpret_cast <const char *>(line),
                    sizeof (line));

    infile.close ();
    afile.close ();

    // From binary file back to text file
    ofstream outfile;
    outfile.open ("outfile.txt");

    afile.open ("infile.dat", ios::in | ios::binary);

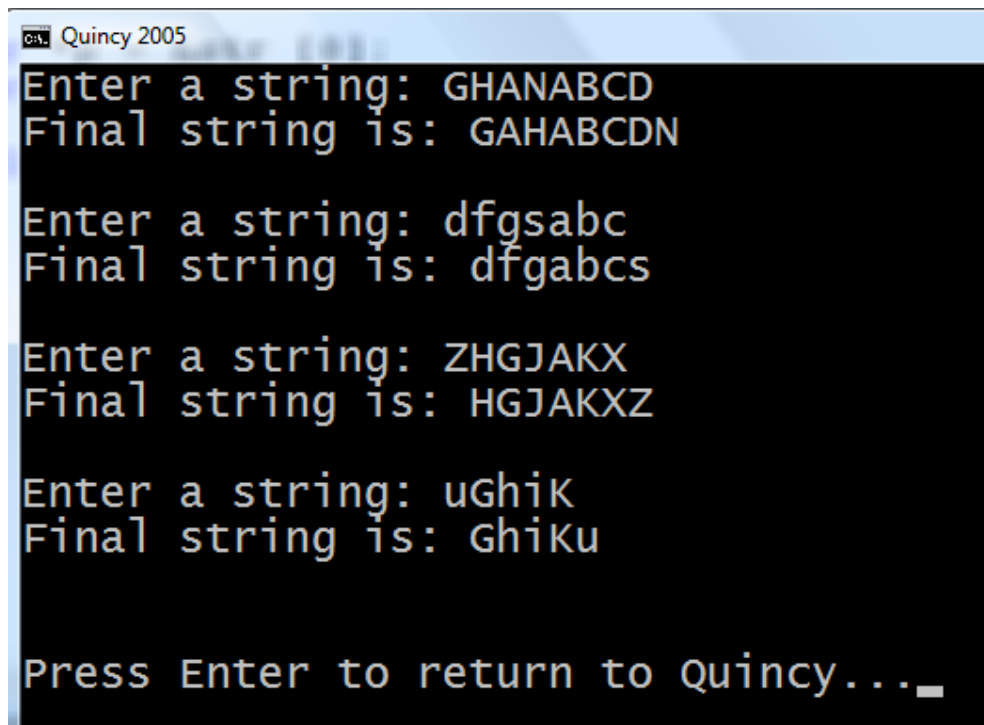
    while (afile.read (reinterpret_cast <char *>(line),
                    sizeof (line)))
        outfile << line << endl;

    afile.close ();
    outfile.close ();
}
```

Workshop 6 – Pointers

Let `str` be a C-string. Write a C++ program to move the largest character towards the end, i.e. the last character is the largest character. Explore the use of pointer arithmetic for the move.

Some possible interactions and displays:



```
Quincy 2005
Enter a string: GHANABCD
Final string is: GAHABCDN

Enter a string: dfgsabc
Final string is: dfgabcs

Enter a string: ZHGJAKX
Final string is: HGJAKXZ

Enter a string: uGhiK
Final string is: GhiKu

Press Enter to return to Quincy...
```

// workshop 6

```
#include <iostream>
using namespace std;
```

```
const int MAX = 20;
```

```
void moveChar (char *);
```

```
int main ()
```

```
{
    char str [MAX];

    for (int i = 1; i <= 4; i++)
    {
        cout << "Enter a string: ";
        cin >> str;

        moveChar (str);
        cout << "Final string is: " << str << endl;

        cout << endl;
    }
}
```

```
void moveChar (char *str)
```

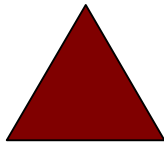
```
{
    char *p = &str [0];
    char *q = &str [1];

    while (*q != '\0')
    {
        if (*p > *q)
        {
            char temp = *p;
            *p = *q;
            *q = temp;
        }

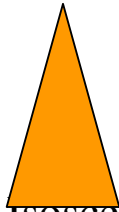
        ++p;
        ++q;
    }
}
```

Workshop 7 – Classes and Objects

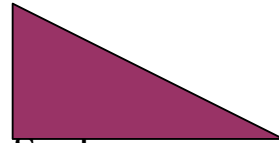
Three types of triangles



Equilateral



Isosceles



Scalene

Suggest a class to describe these three triangles. In your suggestions, try to put in as much as possible everything you know in classes and objects.

// Workshop 7

// File name: Triangle.h

#include <iostream>

#include <cmath>

using namespace std;

class Triangle

{

friend istream& operator>> (istream&, Triangle&);

friend ostream& operator<< (ostream&, const Triangle&);

public:

Triangle ();

Triangle (int);

Triangle (int, int);

Triangle (int, int, int);

Triangle (const Triangle&);

double area () const;

bool operator>= (const Triangle&) const;

Triangle& operator> (Triangle&);

static int getNo ();

private:

int a, b, c;

static int no;

};

```

// File name: Triangle.cpp
#include "Triangle.h"

ostream& operator<< (ostream& os, const Triangle& t)
{
    if (t.a * t.b * t.c != 0)
        os << "Scalene (" << t.a << ", " << t.b << ", "
            << t.c << ")";

    else if (t.b == 0)
        os << "Equilateral (" << t.a << ")";

    else
        os << "Isocetes (" << t.a << ", " << t.b << ")";

    return os;
}

istream& operator>> (istream& is, Triangle& t)
{
    char type;

    t.a = 0;
    t.b = 0;
    t.c = 0;

    is >> type;

    switch (type)
    {
        case 's':
        case 'S': is >> t.a >> t.b >> t.c;
                break;

        case 'e':
        case 'E': is >> t.a;
                break;

        case 'i':
        case 'I': is >> t.a >> t.b;
                break;

    }

    return is;
}

int Triangle::no = 0;

```

```
Triangle::Triangle ()
{
    a = 1;
    b = 0;
    c = 0;
    no++;
}
```

```
Triangle::Triangle (int a)
{
    this -> a = a;
    b = 0;
    c = 0;
}
```

```
Triangle::Triangle (int a, int b)
{
    this -> a = a;
    this -> b = b;
    c = 0;
}
```

```
Triangle::Triangle (int a, int b, int c)
{
    (*this).a = a;
    (*this).b = b;
    (*this).c = c;
}
```

```
Triangle::Triangle (const Triangle& other)
{
    a = other.a;
    b = other.b;
    c = other.c;
}
```

```
double Triangle::area () const
{
    double s;

    if (a * b * c == 0)
```

```

    {
        s = (a + b + c) / 2.0;
        return sqrt (s * (s - a) * (s - b) * (s - c));
    }
    else if (b == 0)
    {
        s = (a + a + a) / 2.0;
        return sqrt (s * (s - a) * (s - a) * (s - a));
    }
    else
    {
        s = (a + b + b) / 2.0;
        return sqrt (s * (s - a) * (s - b) * (s - b));
    }
}

bool Triangle::operator>= (const Triangle& t) const
{
    if (this -> area () > t.area ())
        return true;
    else
        return false;
}

Triangle& Triangle::operator> (Triangle& t)
{
    if ((*this).area () > t.area ())
        return *this;
    else
        return t;
}

int Triangle::getNo ()
{
    return no;
}

```

```

// File name: MainTriangle.cpp
#include "Triangle.h"

int main ()
{
    Triangle *t = new Triangle (3, 4, 5);
    cout << *t << endl;

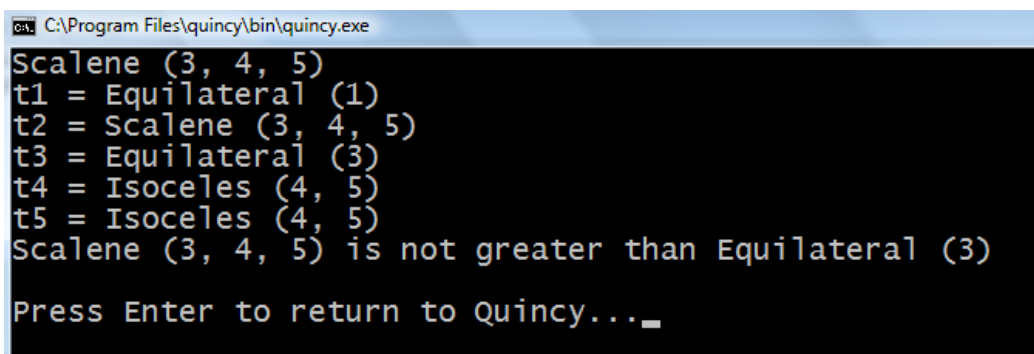
    Triangle t1;
    Triangle t2 (3, 4, 5);
    Triangle t3 (3);
    Triangle t4 (4, 5);

    cout << "t1 = " << t1 << endl;
    cout << "t2 = " << t2 << endl;
    cout << "t3 = " << t3 << endl;
    cout << "t4 = " << t4 << endl;

    Triangle t5 = (t3 > t4);
    cout << "t5 = " << t5 << endl;

    if (t2 >= t3)
        cout << t2 << " is greater than " << t3 << endl;
    else
        cout << t2 << " is not greater than " << t3 << endl;
}

```



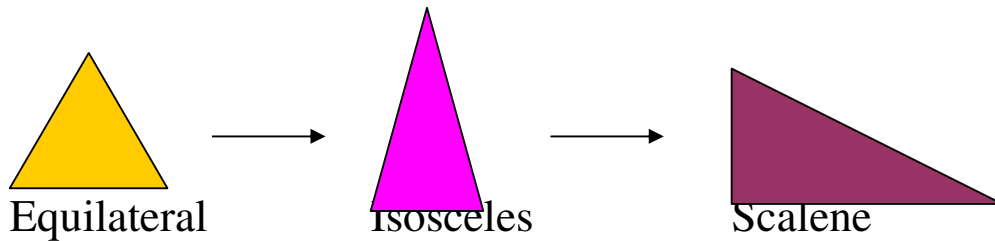
```

C:\Program Files\quincy\bin\quincy.exe
Scalene (3, 4, 5)
t1 = Equilateral (1)
t2 = Scalene (3, 4, 5)
t3 = Equilateral (3)
t4 = Isoceles (4, 5)
t5 = Isoceles (4, 5)
Scalene (3, 4, 5) is not greater than Equilateral (3)
Press Enter to return to Quincy..._

```

Workshop 7a – Inheritance

Three types of triangles



// Workshop 7a (inheritance)

```
#include <iostream>
using namespace std;
```

```
class Equilateral
```

```
{
    friend ostream& operator<< (ostream&, const Equilateral&);

    public:
        Equilateral ();
        Equilateral (int);

    protected:
        int a;
};
```

```
ostream& operator<< (ostream& os, const Equilateral& e)
{
    os << "Equilateral (" << e.a << ")";
    return os;
}
```

```
Equilateral::Equilateral ()
{
    this -> a = 1;
}
```

```
Equilateral::Equilateral (int a)
{
```

```

        this -> a = a;
    }

class Isosceles : public Equilateral
{
    friend ostream& operator<< (ostream&, const Isosceles&);

    public:
        Isosceles ();
        Isosceles (int);
        Isosceles (int, int);

    protected:
        int b;
};

```

```

ostream& operator<< (ostream& os, const Isosceles& i)
{
    os << "Isosceles (";

    if (i.b == 0)
    {
        Equilateral e (i.a);
        os << e;
    }
    else
        os << i.a << ", " << i.b;

    os << ")";

    return os;
}

```

```

Isosceles::Isosceles ()
{
    b = 0;
}

```

```

Isosceles::Isosceles (int a) : Equilateral (a)
{
    this -> b = 0;
}

```

```

Isosceles::Isosceles (int a, int b) : Equilateral (a)
{
    this -> b = b;
}

class Scalene : public Isosceles
{
    friend ostream& operator<< (ostream&, const Scalene&);

    public:
        Scalene ();
        Scalene (int);
        Scalene (int, int);
        Scalene (int, int, int);

    private:
        int c;
};

ostream& operator<< (ostream& os, const Scalene& s)
{
    os << "Scalene (";

    if (s.b == 0 && s.c == 0)
    {
        Equilateral e (s.a);
        os << e;
    }
    else if (s.c == 0)
    {
        Isosceles i (s.a, s.b);
        os << i;
    }
    else
        os << s.a << ", " << s.b << ", " << s.c;

    os << ")";

    return os;
}

Scalene::Scalene ()

```

```

{
    c = 0;
}

Scalene::Scalene (int a) : Isosceles (a)
{
    b = 0;
    c = 0;
}

Scalene::Scalene (int a, int b) : Isosceles (a, b)
{
    c = 0;
}

Scalene::Scalene (int a, int b, int c) : Isosceles (a, b)
{
    this -> c = c;
}

int main ()
{
    Equilateral e1;
    Equilateral e2 (5);

    cout << "e1 = " << e1 << endl;
    cout << "e2 = " << e2 << endl;

    Isosceles i1 (6);
    Isosceles i2 (6, 7);

    cout << "i1 = " << i1 << endl;
    cout << "i2 = " << i2 << endl;

    Scalene s1 (3);
    Scalene s2 (3, 4);
    Scalene s3 (3, 4, 5);

    cout << "s1 = " << s1 << endl;
    cout << "s2 = " << s2 << endl;
    cout << "s3 = " << s3 << endl;
}

```

```
C:\Program Files\quincy\bin\quincy.exe
e1 = Equilateral (1)
e2 = Equilateral (5)
i1 = Isosceles (Equilateral (6))
i2 = Isosceles (6, 7)
s1 = Scalene (Equilateral (3))
s2 = Scalene (Isosceles (3, 4))
s3 = Scalene (3, 4, 5)

Press Enter to return to Quincy...
```

// Workshop 7b – template version

```
#include <iostream>
using namespace std;

template <class T>
class Equilateral
{
    friend ostream& operator<< (ostream&, const Equilateral <int>&);
    friend ostream& operator<< (ostream&, const Equilateral <double>&);

    public:
        Equilateral ();
        Equilateral (T);
        ~Equilateral ();

    protected:
        T a;
};

ostream& operator<< (ostream& os, const Equilateral <int>& e)
{
    os << "Equilateral (" << e.a << ")";
    return os;
}

ostream& operator<< (ostream& os, const Equilateral <double>& e)
{
    os << "Equilateral (" << e.a << ")";
    return os;
}

template <class T>
Equilateral <T>::Equilateral ()
{
    this -> a = 1;
}

template <class T>
Equilateral <T>::Equilateral (T a)
{
    this -> a = a;
}

template <class T>
Equilateral <T>::~~Equilateral ()
{
    cout << " - Garbage collection in class Equilateral" << endl;
}
```

```

template <class T>
class Isosceles : public Equilateral <T>
{
    friend ostream& operator<< (ostream&, const Isosceles <int> &);
    friend ostream& operator<< (ostream&, const Isosceles <double> &);

    public:
        Isosceles ();
        Isosceles (T);
        Isosceles (T, T);
        ~Isosceles ();

    public:
        T b;
};

```

```

ostream& operator<< (ostream& os, const Isosceles <int> & i)
{
    os << "Isosceles (";

    if (i.b == 0)
    {
        Equilateral <int> e (i.a);
        os << e;
    }
    else
        os << i.a << ", " << i.b;

    os << ")";

    return os;
}

```

```

ostream& operator<< (ostream& os, const Isosceles <double> & i)
{
    os << "Isosceles (";

    if (i.b == 0)
    {
        Equilateral <double> e (i.a);
        os << e;
    }
    else
        os << i.a << ", " << i.b;

    os << ")";

    return os;
}

```

```

}

template <class T>
Isosceles <T>::Isosceles ()
{
    b = 0;
}

template <class T>
Isosceles <T>::Isosceles (T a) : Equilateral <T> (a)
{
    this -> b = 0;
}

template <class T>
Isosceles <T>::Isosceles (T a, T b) : Equilateral <T>(a)
{
    this -> b = b;
}

template <class T>
Isosceles <T>::~Isosceles ()
{
    cout << " - Garbage collection in class Isosceles" << endl;
}

template <class U>
class Scalene : public Isosceles <U>
{
    friend ostream& operator<< (ostream&, const Scalene <int> &);
    friend ostream& operator<< (ostream&, const Scalene <double> &);

    public:
        Scalene ();
        Scalene (U);
        Scalene (U, U);
        Scalene (U, U, U);
        ~Scalene ();

    private:
        U c;
};

ostream& operator<< (ostream& os, const Scalene <int> & s)
{
    os << "Scalene (";

    if (s.b == 0 && s.c == 0)
    {
        Equilateral <int> e (s.a);
    }
}

```

```

        os << e;
    }
    else if (s.c == 0)
    {
        Isosceles <int> i (s.a, s.b);
        os << i;
    }
    else
        os << s.a << ", " << s.b << ", " << s.c;

    os << ")";

    return os;
}

ostream& operator<< (ostream& os, const Scalene <double> & s)
{
    os << "Scalene (";

    if (s.b == 0 && s.c == 0)
    {
        Equilateral <double> e (s.a);
        os << e;
    }
    else if (s.c == 0)
    {
        Isosceles <double> i (s.a, s.b);
        os << i;
    }
    else
        os << s.a << ", " << s.b << ", " << s.c;

    os << ")";

    return os;
}

template <class U>
Scalene <U>::Scalene ()
{
    c = 0;
}

template <class U>
Scalene <U>::Scalene (U a) : Isosceles <U> (a)
{
    c = 0;
}

template <class U>

```

```

Scalene <U>::Scalene (U a, U b) : Isosceles <U> (a, b)
{
    c = 0;
}

template <class U>
Scalene <U>::Scalene (U a, U b, U c) : Isosceles <U> (a, b)
{
    this -> c = c;
}

template <class U>
Scalene <U>::~~Scalene ()
{
    cout << " - Garbage collection in class Scalene" << endl;
}

int main ()
{

    Scalene <int> s3 (3, 4, 5);
    Scalene <double> s4 (3.02, 4.09, 5.2);

    cout << "s3 = " << s3 << endl;
    cout << "s4 = " << s4 << endl;
}

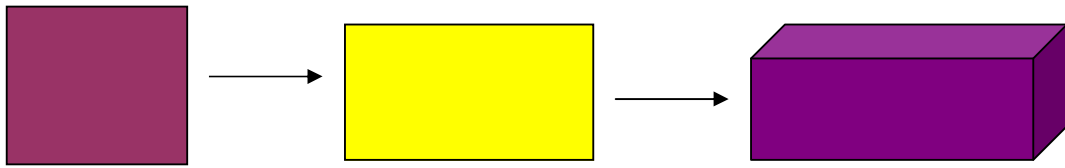
```

```

s3 = Scalene (3, 4, 5)
s4 = Scalene (3.02, 4.09, 5.2)
- Garbage collection in class Scalene
- Garbage collection in class Isosceles
- Garbage collection in class Equilateral
- Garbage collection in class Scalene
- Garbage collection in class Isosceles
- Garbage collection in class Equilateral

```

Workshop 8 - Inheritance



An extension of a square is a rectangle; an extension of a rectangle is a rectangular box.

Suggest a few classes to link them together.

// Workshop 8

```
#include <iostream>
using namespace std;

class Square
{
    public:
        Square ();
        Square (int);

        int area () const;
        void print () const;

    protected:
        int length;
};

Square::Square ()
{
    length = 1;
}

Square::Square (int length)
{
    this -> length = length;
}

int Square::area () const
{
    return length * length;
}

void Square::print () const
{
    cout << "Square (" << length << ")";
}

class Rectangle : public Square
{
    public:
        Rectangle ();
        Rectangle (int, int);
};
```

```

        int area () const;
        void print () const;

    protected:
        int width;
};

Rectangle::Rectangle ()
{
    // By default it will call the
    // default constructor of
    // Square class to initialize
    // the length = 1

    this -> width = 1;
}

Rectangle::Rectangle (int length, int width) : Square (length)
{
    this -> width = width;
}

int Rectangle::area () const
{
    if (length == width)
        return Square::area ();
    else
        return length * width;
}

void Rectangle::print () const
{
    if (length == width)
        Square::print ();
    else
        cout << "Rectangle (" << length << ", " << width << ")";
}

class Box : public Rectangle
{
    public:
        Box ();

```

```

        Box (int, int, int);

        int volume () const;
        void print () const;

    private:
        int height;
};

Box::Box ()
{
    height = 1;
}

Box::Box (int length, int width, int height) : Rectangle (length, width)
{
    this -> height = height;
}

int Box::volume () const
{
    return area () * height;
}

void Box::print () const
{
    cout << "Box (";
    Rectangle::print ();
    cout << ", " << height << ")";
}

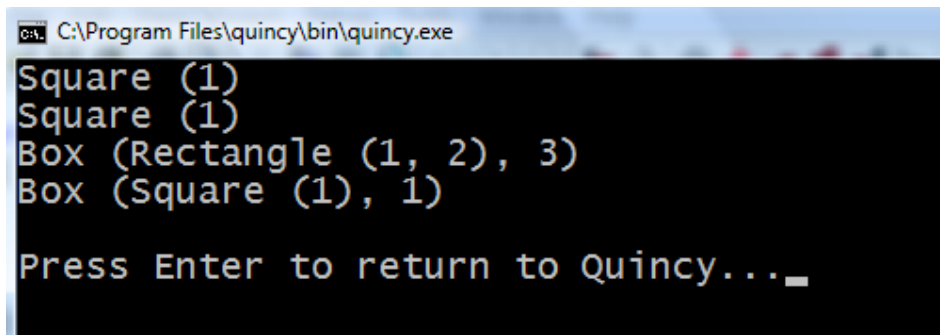
int main ()
{
    Square s;
    s.print ();
    cout << endl;

    Rectangle r;
    r.print ();
    cout << endl;

    Box b1 (1, 2, 3);
    Box b2;
}

```

```
b1.print ();  
cout << endl;  
  
b2.print ();  
cout << endl;  
  
}
```



```
C:\Program Files\quincy\bin\quincy.exe  
Square (1)  
Square (1)  
Box (Rectangle (1, 2), 3)  
Box (Square (1), 1)  
  
Press Enter to return to Quincy..._
```

Workshop 9 – template classes + STL

Queue has two operations:

Enqueue - Queue up from the back

Dequeue – Remove from the front

A queue can be implemented using vector. Suggest a template class to represent a queue and implement it.

// Workshop 9

// A queue is implemented using a vector

```
#include <iostream>
```

```
#include <vector>
```

```
using namespace std;
```

```
template <class T>
```

```
class Queue {
```

```
    public:
```

```
        Queue();
```

```
        void enqueue(T);
```

```
        T dequeue ();
```

```
        bool isEmpty () const;
```

```
    private:
```

```
        vector <T> v;
```

```
};
```

```
template <class T>
```

```
Queue<T>::Queue()
```

```
{
```

```
}
```

```
template <class T>
```

```
void Queue<T>::enqueue(T item)
```

```
{
```

```
    v.insert (v.end (), item);
```

```
}
```

```

template <class T>
T Queue<T>::dequeue()
{
    T item;

    if (!isEmpty ())
    {
        item = v [0];
        v.erase (v.begin ());
    }

    return item;
}

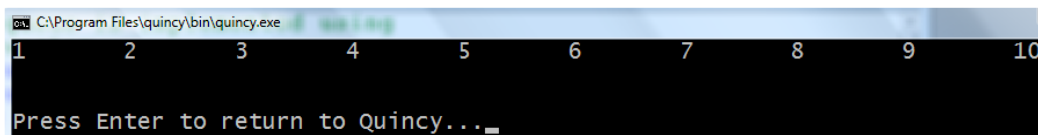
template <class T>
bool Queue <T>::isEmpty () const
{
    return v.size () == 0;
}

int main ()
{
    Queue <int> s;

    for (int i = 1; i <= 10; i++)
    {
        int element = i;
        s.enqueue (element);
    }

    while (!s.isEmpty ())
        cout << s.dequeue () << "\t";
    cout << endl;
}

```



Workshop 10 – All features

An extension of a real number is a complex number that both real and imaginary parts are real numbers. In computing, we have explore the assignment operators, for example, +=, -=, *= or /=.

For this workshop, design two classes, Number class and the Complex Number class which is an extension of Number class. In each of the classes, you should some assignment operators; for example, only += and -=.

// Workshop 10

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

class Number
{
    public:
        Number ();
        Number (double);

        void add (const Number&);
        void minus (const Number&);

        void operator+= (const Number&);
        void operator-= (const Number&);

        void print () const;

    protected:
        double x;
};

#include "Number.h"

Number::Number ()
```

```

{
    x = 0.0;
}

Number::Number (double x)
{
    this -> x = x;
}

void Number::add (const Number& otherNo)
{
    this -> x += otherNo.x;
}

void Number::operator+= (const Number& otherNo)
{
    this -> x += otherNo.x;
}

void Number::minus (const Number& otherNo)
{
    this -> x -= otherNo.x;
}

void Number::operator-= (const Number& otherNo)
{
    this -> x -= otherNo.x;
}

void Number::print () const
{
    cout << x;
}

#include "Number.h"

class ComplexNumber : public Number
{
public:
    ComplexNumber ();
    ComplexNumber (double);           // 0 + yi

```

```

        ComplexNumber (double, double);

        void add (const ComplexNumber&);
        void minus (const ComplexNumber&);

        void operator+= (const ComplexNumber&);
        void operator-= (const ComplexNumber&);

        void print () const;

    private:
        double y;
};

#include "ComplexNumber.h"

ComplexNumber::ComplexNumber ()
{
    // The constructor invokes
    // the superclass constructor
    // to initialize x to 0.0

    y = 0.0;
}

ComplexNumber::ComplexNumber (double y) : Number (0.0)
{
    this -> y = y;
}

ComplexNumber::ComplexNumber (double x, double y) : Number (x)
{
    this -> y = y;
}

void ComplexNumber::add (const ComplexNumber& otherComplex)
{
    x += otherComplex.x;
    y += otherComplex.y;
}

```

```

void ComplexNumber::operator+= (const ComplexNumber&
otherComplex)
{
    x += otherComplex.x;
    y += otherComplex.y;
}

void ComplexNumber::minus (const ComplexNumber&
otherComplex)
{
    x -= otherComplex.x;
    y -= otherComplex.y;
}

void ComplexNumber::operator-= (const ComplexNumber&
otherComplex)
{
    x -= otherComplex.x;
    y -= otherComplex.y;
}

void ComplexNumber::print () const
{
    cout << "(";
    Number::print ();
    cout << ", ";
    cout << y << ")";
}

#include "ComplexNumber.h"
#include "vector"
#include "list"

double getNo ();

int main ()
{
    vector <ComplexNumber> v;
    list <ComplexNumber> alist;
    list <ComplexNumber>::iterator itL = alist.begin ();

    srand (time (NULL));

```

```

cout << "During construction" << endl;

cout << "A vector\t\tA list" << endl;
for (int i = 1; i <= 5; i++)
{
    ComplexNumber c1 (getNo (), getNo ());
    v.push_back (c1);
    c1.print ();
    cout << "\t";

    ComplexNumber c2 (getNo (), getNo ());
    itL = alist.insert (alist.end (), c2); // add from end
    c2.print ();
    cout << endl;
}

cout << endl;

//    vector <ComplexNumber>::iterator itV = v.end ();

cout << "After construction" << endl;
cout << "A vector\t\tA list" << endl;

int index = 0;
itL = alist.begin ();

while (itL != alist.end () && !v.empty ())
{
    v.at (index).print ();
    cout << "\t";

    (*itL).print ();
    cout << endl;

    ++itL;
    ++index;
}

```

```

cout << endl;

itL = alist.begin ();
index = 0;

cout << "Test on Complex 1 += Complex 2 " << endl;
cout << "Complex 1\t\tComplex2\t\tComplex 1 updated" <<
endl;

while (itL != alist.end () && !v.empty ())
{

    v.at (index).print ();
    cout << "\t";

    (*itL).print ();
    cout << "\t";
    v.at (index) += *itL;

    v.at (index).print ();
    cout << endl;

    ++itL;
    ++index;
}

cout << endl;
cout << "Test on Complex 2 -= Complex 1 " << endl;
cout << "Complex 1\t\tComplex2\t\tComplex 2 updated"
<< endl;

itL = alist.begin ();
index = 0;

while (itL != alist.end () && !v.empty ())
{

    v.at (index).print ();
    cout << "\t";

    (*itL).print ();
    cout << "\t";

    *itL -= v.at (index);

```

```

        (*itL).print ();
        cout << endl;

        ++itL;
        ++index;
    }
}

double getNo ()
{
    return 1.0 * rand () / (rand () + 1);
}

```

```

During construction
A vector      A list
(2.00706, 2.72207) (0.663153, 0.398348)
(0.293799, 0.851027) (0.695528, 0.537935)
(0.841937, 0.0764351) (0.355456, 0.145937)
(0.37274, 1.19985) (0.517387, 1.84007)
(0.510529, 1.41511) (0.78728, 1.19648)

After construction
A vector      A list
(2.00706, 2.72207) (0.663153, 0.398348)
(0.293799, 0.851027) (0.695528, 0.537935)
(0.841937, 0.0764351) (0.355456, 0.145937)
(0.37274, 1.19985) (0.517387, 1.84007)
(0.510529, 1.41511) (0.78728, 1.19648)

Test on Complex 1 += Complex 2
Complex 1      Complex2      Complex 1 updated
(2.00706, 2.72207) (0.663153, 0.398348) (2.67021, 3.12042)
(0.293799, 0.851027) (0.695528, 0.537935) (0.989327, 1.38896)
(0.841937, 0.0764351) (0.355456, 0.145937) (1.19739, 0.222372)
(0.37274, 1.19985) (0.517387, 1.84007) (0.890127, 3.03992)
(0.510529, 1.41511) (0.78728, 1.19648) (1.29781, 2.61159)

Test on Complex 2 -= Complex 1
Complex 1      Complex2      Complex 2 updated
(2.67021, 3.12042) (0.663153, 0.398348) (-2.00706, -2.72207)
(0.989327, 1.38896) (0.695528, 0.537935) (-0.293799, -0.851027)
(1.19739, 0.222372) (0.355456, 0.145937) (-0.841937, -0.0764351)
(0.890127, 3.03992) (0.517387, 1.84007) (-0.37274, -1.19985)
(1.29781, 2.61159) (0.78728, 1.19648) (-0.510529, -1.41511)

```

Case Study – See lecture notes Module_10

```
// File name: Person.h
#include <string>
#include <iostream>
#include <vector>
#include <list>
using namespace std;

class Person
{
    friend ostream& operator<< (ostream&, const Person&);
public:
    Person ();
    Person (string, char, int);
    string getName () const;
    char getSex () const;
    int getAge () const;

protected:
    string name;
    char sex;
    int age;
};
```

```

// File name: Person.cpp
#include "Person.h"

ostream& operator<< (ostream& os, const Person& p)
{
    os << "Person ("
        << p.name << ", "
        << p.sex << ", "
        << p.age << ")";

    return os;
}

Person::Person ()
{
}

Person::Person (string name, char sex, int age)
{
    this -> name = name;
    this -> sex = sex;
    this -> age = age;
}

string Person::getName () const
{
    return name;
}

char Person::getSex () const
{
    return sex;
}

int Person::getAge () const
{
    return age;
}

```

```
// File name: Student.h
#include "Person.h"

class Student : public Person
{
    friend ostream& operator<< (ostream&, const Student&);

    public:
        Student ();
        Student (string, char, int, string); // Singaporean
        Student (string, char, int, string, string); // foreigner

    private:
        string matri;
        string country;
};
```

```

// File name: Student.cpp
#include "Student.h"

ostream& operator<< (ostream& os, const Student& s)
{
    Person p (s.name, s.sex, s.age);

    os << "Student (" << p << ", "
        << s.matri << ", ";

    if (s.country == "")
        os << "Singapore";
    else
        os << s.country;

    os << ")";

    return os;
}

Student::Student ()
{
}

Student::Student (string name, char sex, int age, string matri) :
    Person (name, sex, age)
{
    this -> matri = matri;
    this -> country = "";
}

Student::Student (string name, char sex, int age,
    string matri, string country) : Person (name, sex, age)
{
    this -> matri = matri;
    this -> country = country;
}

```

```

// File name: TestStudent.cpp
#include "Student.h"

void printVector (vector <Student>);
void printList (list <Student>);

int main ()
{
    vector <Student> s;

    Student s1 ("Heng Aik Koan", 'M', 28, "1234567");
    Student s2 ("Chen Chen", 'F', 18, "4567898", "China");
    Student s3 ("Heng James", 'M', 28, "1234587");
    Student s4 ("Chen Chen Chen", 'F', 18, "8888888", "Hong Kong");

    s.push_back (s1);
    s.push_back (s2);
    s.push_back (s3);
    s.push_back (s4);

    printVector (s);

    cout << "-----" << endl;

    list <Student> alist;
    list <Student>::iterator i = alist.begin ();

    i = alist.insert (i, s1);
    i = alist.insert (i, s2);
    i = alist.insert (i, s3);
    i = alist.insert (i, s4);

    printList (alist);
}

void printVector (vector <Student> s)
{
    vector <Student>::iterator it;

    for (it = s.begin (); it < s.end (); it++)
        cout << *it << endl;
}

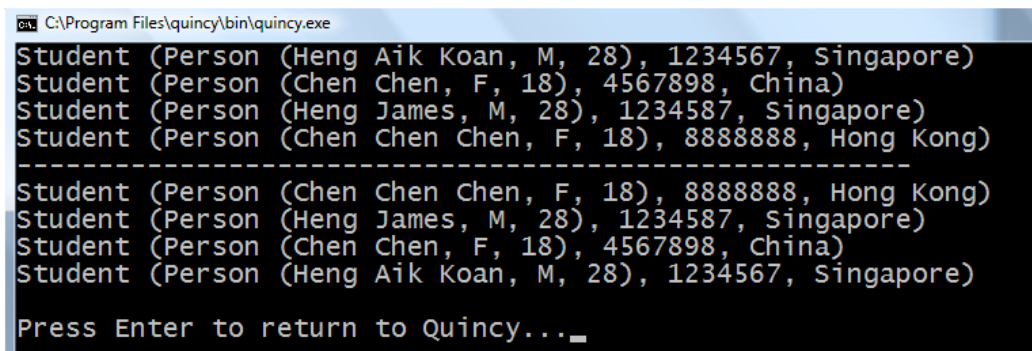
```

```

void printList (list <Student> alist)
{
    list <Student>::iterator it;

    it = alist.begin ();
    while (it != alist.end ())
    {
        cout << *it << endl;
        ++it;
    }
}

```



```

C:\Program Files\quincy\bin\quincy.exe
Student (Person (Heng Aik Koan, M, 28), 1234567, Singapore)
Student (Person (Chen Chen, F, 18), 4567898, China)
Student (Person (Heng James, M, 28), 1234587, Singapore)
Student (Person (Chen Chen Chen, F, 18), 8888888, Hong Kong)
-----
Student (Person (Chen Chen Chen, F, 18), 8888888, Hong Kong)
Student (Person (Heng James, M, 28), 1234587, Singapore)
Student (Person (Chen Chen, F, 18), 4567898, China)
Student (Person (Heng Aik Koan, M, 28), 1234567, Singapore)
Press Enter to return to Quincy..._

```