

// Workshop 1

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

int main ()
{
    int n, k;
    int sumDigits;

    for (int i = 1; i <= 3; i++)
    {
        do
        {
            cout << "Enter a positive integer: ";
            cin >> n;

            cin.clear ();
            cin.ignore (100, '\n');
        } while (n <= 0);

        cout << "Enter a digit: ";
        cin >> k;

        sumDigits = 0;
        while (n > 0)
        {
            sumDigits += n % 10;
            n /= 10;
        }

        if (sumDigits % k == 0)
            cout << "==> divisible by " << k << endl;
        else
            cout << "==> not divisible by " << k << endl;

        cout << "-----" << endl;
    }
}
```

```
Enter a positive integer: -123          // a negative number
Enter a positive integer: -3333        // rejected
Enter a positive integer: 56789
Enter a digit: 9
==> not divisible by 9
-----
Enter a positive integer: 3458
Enter a digit: 2
==> divisible by 2
-----
Enter a positive integer: 333999
Enter a digit: 9
==> divisible by 9
-----
```

// Workshop 2

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

// Get an integer greater than x
int getAnInteger(int);

// Get sum of digits of n (the parameter)
int getSumOfDigits (int);

// Test if the sum of digits (1st parameter)
// is divisible by k (2nd parameter)
bool testDivisible (int, int);

int main ()
{
    int n, k;
    int sumDigits;

    for (int i = 1; i <= 3; i++)
    {

        n = getAnInteger (0);

        cout << "Enter a digit: ";
        cin >> k;

        sumDigits = getSumOfDigits (n);

        bool ok = testDivisible (sumDigits, k);

        if (ok)
            cout << n << " is divisible by " << k << endl;
        else
            cout << n << " is not divisible by " << k << endl;

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

    }
}

// Get an integer greater than x
int getAnInteger(int x)
{
    int n;

    do
    {
        cout << "Enter a positive integer: ";
        cin >> n;

        cin.clear ();
        cin.ignore (100, '\n');
    } while (n <= x);

    return n;
}
```

```

// Get sum of digits of n (the parameter)
int getSumOfDigits (int n)
{
    int sumDigits = 0;
    while (n > 0)
    {
        sumDigits += n % 10;
        n /= 10;
    }

    return sumDigits;
}

// Test if the sum of digits (1st parameter)
// is divisible by k (2nd parameter)
bool testDivisible (int sumDigits, int k)
{
    if (sumDigits % k == 0)
        return true;
    else
        return false;
}

```

// Workshop 2a - Using rand function

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

// Generate an integer greater than x
int getAnInteger(int);

// Generate and return a single digit
int getADigit ();

// Get sum of digits of n (the parameter)
// - recursive version
int getSumOfDigits (int);

// Test if the sum of digits (1st parameter)
// is divisible by k (2nd parameter)
bool testDivisible (int, int);

int main ()
{
    int n, k;
    int sumDigits;

    srand (time (NULL));

    for (int i = 1; i <= 100; i++)
    {

        n = getAnInteger (10000);
        k = getADigit ();

        sumDigits = getSumOfDigits (n);

        bool ok = testDivisible (sumDigits, k);

        if (ok)
            cout << n << " is divisible by " << k << endl;
        else
            cout << n << " is not divisible by " << k << endl;

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

    }
}

// Get an integer greater than x
int getAnInteger(int x)
{
    int n;

    do
    {
        n = rand ();
```

```

        } while (n <= x);

        return n;
    }

    // Generate and return a single digit
    int getADigit ()
    {
        int k = rand () % 8 + 2;
        return k;
    }

    // Get sum of digits of n (the parameter)
    // - Recursive version
    int getSumOfDigits (int n)
    {
        if (n < 10)
            return n;
        else
            return n % 10 + getSumOfDigits (n / 10);
    }

    // Test if the sum of digits (1st parameter)
    // is divisible by k (2nd parameter)
    bool testDivisible (int sumDigits, int k)
    {
        if (sumDigits % k == 0)
            return true;
        else
            return false;
    }

```

// Workshop 3

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

const int MAX = 10;

// Convert a string of digits
// to an integer and return its value
int getInteger (const char []);

int main ()
{
    char str [MAX];

    for (int i = 1; i <= 5; i++)
    {
        cout << "Enter a string of digits: ";
        cin >> str;

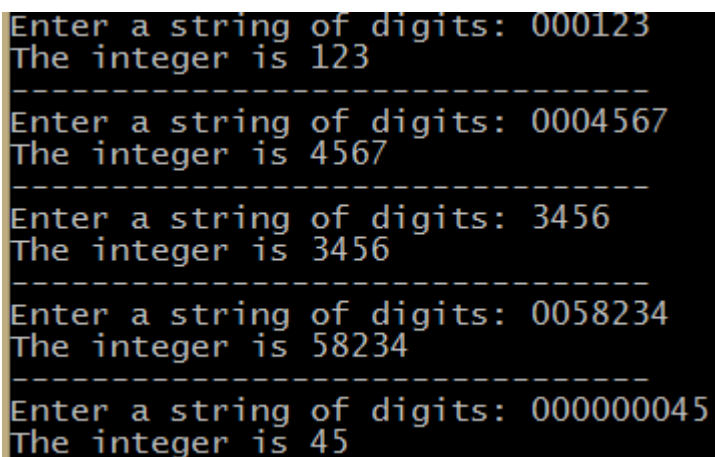
        cout << "The integer is "
             << getInteger (str)
             << endl;

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

// Convert a string of digits
// to an integer and return its value
int getInteger (const char str [])
{
    int n = 0;

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

    return n;
}
```



```
Enter a string of digits: 000123
The integer is 123
-----
Enter a string of digits: 0004567
The integer is 4567
-----
Enter a string of digits: 3456
The integer is 3456
-----
Enter a string of digits: 0058234
The integer is 58234
-----
Enter a string of digits: 000000045
The integer is 45
```

// Workshop 4

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

const int MAX = 20;

enum Status {Biggest, Between, Smallest};

// Generate and return three random integers
void get3Ints (int&, int&, int&);

// Return the status of the third integer
Status getStatus (int, int, int);

// Get a string to represent the status
void getStatusString (Status, char []);

int main ()
{
    int a, b, c;
    Status s;
    char str [MAX];

    srand (time (NULL));

    for (int i = 1; i <= 5; i++)
    {
        get3Ints (a, b, c);
        s = getStatus (a, b, c);
        getStatusString (s, str);

        cout << "Given three integers "
              << a << "\t" << b << "\t" << c
              << endl;

        cout << "==> " << str << endl;

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

// Generate and return three random integers
void get3Ints (int& a, int& b, int& c)
{
    a = rand ();
    b = rand ();
    c = rand ();
}

// Return the status of the third integer
Status getStatus (int a, int b, int c)
{
```

```

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

// Get a string to represent the status
void getStatusString (Status s, char str [])
{
    switch (s)
    {
        case Biggest: strcpy (str, "Biggest");
                      break;
        case Smallest: strcpy (str, "Smallest");
                      break;
        default:      strcpy (str, "Between");
    }
}

```

```

Given three integers 23064      8587      3263
==> Smallest
-----
Given three integers 10606      4012      25216
==> Biggest
-----
Given three integers 27149      24887      26746
==> Between
-----
Given three integers 2525       18394      4196
==> Between
-----
Given three integers 13432      32663      9943
==> Smallest
-----

```

// Workshop 5 - Text file and binary file processing

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

const int MAX = 80;

int main ()
{
    ifstream infile;
    fstream afile;

    infile.open ("Workshop_1.cpp");
    afile.open ("temp.dat", ios::out | ios::binary);

    char str [MAX];

    // Convert our program to a binary file
    while (infile.getline (str, MAX))
    {
        afile.write (reinterpret_cast <const char *>(str),
                     sizeof (str));
    }

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

    ofstream outfile;
    outfile.open ("outfile.txt");

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

    // Read the binary file records and store them in a text file
    while (afile.read (reinterpret_cast <char *>(str), sizeof (str)))
    {
        outfile << str << endl;
    }

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

// Workshop 6 - Pointers and arrays

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

const int MAX = 80;

// To move the largest characters toward the end
void move_1 (char []);

// As above, pointer version
void move_2 (char []);

int main ()
{
    char str1 [MAX];
    char str2 [MAX];

    for (int i = 1; i <= 3; i++)
    {
        cout << "Enter a string: ";
        cin.getline (str1, MAX);

        strcpy (str2, str1);

        move_1 (str1);
        move_2 (str2);

        cout << "==" << str1 << endl;
        cout << "==" << str2 << endl;

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

// To move the largest characters toward the end
void move_1 (char str [])
{
    int i = 0;
    int j = 1;

    while (str [j] != '\0')
    {
        if (str [i] > str [j])
        {
            char temp = str [i];
            str [i] = str [j];
            str [j] = temp;
        }

        ++i;
        ++j;
    }
}

// As above, pointer version
void move_2 (char str [])
{

```

```

char *i = &str [0];
char *j = &str [1];

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

    ++i;
    ++j;
}
}

```

```

Enter a string: ASDFGIUYTR
==> ADFGISUTRY
==> ADFGISUTRY
-----
Enter a string: sdfghiuyt
==> dfghisuty
==> dfghisuty
-----
Enter a string: SDFdfghASDF
==> DFSdfgASDFh
==> DFSdfgASDFh
-----

```

```

// Workshop 7 - Classes and Objects
//
// - A class to describe three types of triangles

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

class Triangle
{
public:
    // Default constructor
    Triangle ();

    // Equilateral
    Triangle (int);

    // Isosceles
    Triangle (int, int);

    // Scalene
    Triangle (int, int, int);

    // Copy constructor
    Triangle (const Triangle&);

    // destructor
    ~Triangle ();

    // What can we do?
    double area () const;
    int perimeter () const;
    void print () const;

    // Accessor methods
    int getA () const;
    int getB () const;
    int getC () const;

    // Mutator method
    void set (int, int, int);

private:
    int a, b, c;
};

int main ()
{
    Triangle t0;
    Triangle t1 (6);
    Triangle t2 (3, 4);
    Triangle t3 (5, 6, 7);
    Triangle t4 (t3);

    cout << "t0 = ";
    t0.print ();
}

```

```

    cout << endl;

    cout << "t1 = ";
    t1.print ();
    cout << endl;

    cout << "t2 = ";
    t2.print ();
    cout << endl;

    cout << "t3 = ";
    t3.print ();
    cout << endl;

    cout << "t4 = ";
    t4.print ();
    cout << endl;

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

    cout << "An array of triangles" << endl;
    Triangle *t = new Triangle [5];

    for (int i = 0; i < 5; i++)
    {
        cout << "t [" << i << "] = ";
        t [i].print ();
        cout << endl;
    }

    cout << "-----" << endl;
    cout << "Garbage collection" << endl;

    delete [] t;
}

// Default constructor
Triangle::Triangle ()
{
    a = 1;
    b = 0;
    c = 0;
}

// Equilateral
Triangle::Triangle (int a)
{
    if (a <= 0)
        throw new exception ();

    this -> a = a;
    b = 0;
    c = 0;
}

```

```

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

        // Scalene
Triangle::Triangle (int a, int b, int c)
{
    set (a, b, c);
}

        // Copy constructor
Triangle::Triangle (const Triangle& t)
{
    set (t.a, t.b, t.c);
}

        // destructor
Triangle::~Triangle ()
{
    static int i = 0;

    cout << ++i << " triangle(s) deleted" << endl;
}

        // What can we do?
double Triangle::area () const
{
    double s;

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

int Triangle::perimeter () const
{
    if (b == 0)
    {
        return 3 * a;
    }
    else if (c == 0)
    {

```

```

        return a + a + b;
    }
    else
    {
        return a + b + c;
    }
}

void Triangle::print () const
{
    if (b == 0)
    {
        cout << "Equilateral (" << a << ")";
    }
    else if (c == 0)
    {
        cout << "Isosceles (" << a << ", " << b << ")";
    }
    else
    {
        cout << "Scalene (" << a << ", " << b << ", " << c << ")";
    }
}

// Accessor methods
int Triangle::getA () const
{
    return a;
}

int Triangle::getB () const
{
    return b;
}

int Triangle::getC () const
{
    return c;
}

// Mutator method
void Triangle::set (int a, int b, int c)
{
    this -> a = a;
    this -> b = b;
    this -> c = c;
}

```

```
t0 = Equilateral (1)
t1 = Equilateral (6)
t2 = Isosceles (3, 4)
t3 = Scalene (5, 6, 7)
t4 = Scalene (5, 6, 7)
```

An array of triangles

```
t [0] = Equilateral (1)
t [1] = Equilateral (1)
t [2] = Equilateral (1)
t [3] = Equilateral (1)
t [4] = Equilateral (1)
```

Garbage collection

```
1 triangle(s) deleted
2 triangle(s) deleted
3 triangle(s) deleted
4 triangle(s) deleted
5 triangle(s) deleted
6 triangle(s) deleted
7 triangle(s) deleted
8 triangle(s) deleted
9 triangle(s) deleted
10 triangle(s) deleted
```

```

// Workshop 7a - Classes and Objects
//
// - A class to describe three types of triangles
// - friend functions
// - operators overloaded

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

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

public:
    // Default constructor
    Triangle ();

    // Equilateral
    Triangle (int);

    // Isosceles
    Triangle (int, int);

    // Scalene
    Triangle (int, int, int);

    // Copy constructor
    Triangle (const Triangle&);

    // destructor
    ~Triangle ();

    // What can we do?
    double area () const;
    int perimeter () const;

    // Accessor methods
    int getA () const;
    int getB () const;
    int getC () const;

    // Mutator method
    void set (int, int, int);

private:
    int a, b, c;
};

int main ()
{
    Triangle t0;
    Triangle t1 (6);
    Triangle t2 (3, 4);
    Triangle t3 (5, 6, 7);
    Triangle t4 (t3);
}

```

```

    cout << "t0 = " << t0 << endl;
    cout << "t1 = " << t1 << endl;
    cout << "t2 = " << t2 << endl;
    cout << "t3 = " << t3 << endl;
    cout << "t4 = " << t4 << endl;

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

    cout << "An array of triangles" << endl;
    Triangle *t = new Triangle [5];

    for (int i = 0; i < 5; i++)
    {
        cout << "t [" << i << "] = " << t [i] << endl;
    }

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

    cout << "Garbage collection" << endl;

    delete [] t;
}

ostream& operator<< (ostream& os, const Triangle& t)
{
    if (t.b == 0)
    {
        os << "Equilateral (" << t.a << ")";
    }
    else if (t.c == 0)
    {
        os << "Isosceles (" << t.a << ", " << t.b << ")";
    }
    else
    {
        os << "Scalene (" << t.a << ", " << t.b << ", " << t.c <<
    "
)";
    }

    return os;
}

// Default constructor
Triangle::Triangle ()
{
    a = 1;
    b = 0;
    c = 0;
}

// Equilateral
Triangle::Triangle (int a)
{
    if (a <= 0)
        throw new exception ();
}

```

```

        this -> a = a;
        b = 0;
        c = 0;
    }

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

        // Scalene
Triangle::Triangle (int a, int b, int c)
{
    set (a, b, c);
}

        // Copy constructor
Triangle::Triangle (const Triangle& t)
{
    set (t.a, t.b, t.c);
}

        // destructor
Triangle::~Triangle ()
{
    static int i = 0;

    cout << ++i << " triangle(s) deleted" << endl;
}

        // What can we do?
double Triangle::area () const
{
    double s;

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

int Triangle::perimeter () const
{

```

```

        if (b == 0)
        {
            return 3 * a;
        }
        else if (c == 0)
        {
            return a + a + b;
        }
        else
        {
            return a + b + c;
        }
    }

    // Accessor methods
    int Triangle::getA () const
    {
        return a;
    }

    int Triangle::getB () const
    {
        return b;
    }

    int Triangle::getC () const
    {
        return c;
    }

    // Mutator method
    void Triangle::set (int a, int b, int c)
    {
        this -> a = a;
        this -> b = b;
        this -> c = c;
    }

```

```

// Workshop 7b - Classes and Objects
//
// - A class to describe three types of triangles
// - Inheritance
// : Equilateral -> Isosceles -> Scalene

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

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

public:
    Equilateral ();

    Equilateral (int);

    int perimeter () const;

protected:
    int a;
};

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

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

    int perimeter () const;

protected:
    int b;
};

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

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

    int perimeter () const;

private:
    int c;
};

int main ()
{

```

```

Equilateral e0;
Equilateral e1 (3);

cout << "e0 = " << e0
      << " with perimeter " << e0.perimeter ()
      << endl;

cout << "e1 = " << e1
      << " with perimeter " << e1.perimeter ()
      << endl;

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

Isosceles i0;
Isosceles i1 (5);
Isosceles i2 (6, 7);

cout << "i0 = " << i0
      << " with perimeter " << i0.perimeter ()
      << endl;

cout << "i1 = " << i1
      << " with perimeter " << i1.perimeter ()
      << endl;

cout << "i2 = " << i2
      << " with perimeter " << i2.perimeter ()
      << endl;

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

Scalene s0;
Scalene s1 (5);
Scalene s2 (6, 7);
Scalene s3 (7, 8, 9);

cout << "s0 = " << s0
      << " with perimeter " << s0.perimeter ()
      << endl;

cout << "s1 = " << s1
      << " with perimeter " << s1.perimeter ()
      << endl;

cout << "s2 = " << s2
      << " with perimeter " << s2.perimeter ()
      << endl;

cout << "s3 = " << s3
      << " with perimeter " << s3.perimeter ()
      << endl;
}

ostream& operator<< (ostream& os, const Equilateral& e)
{

```

```

        os << "Equilateral (" << e.a << ")";

        return os;
}

Equilateral::Equilateral ()
{
    a = 1;
}

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

int Equilateral::perimeter () const
{
    return 3 * a;
}

// -----

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)
{
    b = 0;
}

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

int Isosceles::perimeter () const

```

```

{
    if (b == 0)
        return Equilateral::perimeter ();
    else
        return a + a + b;
}

// -----

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

    if (s.b == 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 << ")";
}

Scalene::Scalene ()
{
    c = 0;
}

Scalene::Scalene (int a) : Isosceles (a)
{
    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 Scalene::perimeter () const
{
    if (b == 0)
        return Equilateral::perimeter ();
    else if (c == 0)
        return Isosceles::perimeter ();
    else
        return a + b + c;
}

```

```
e0 = Equilateral (1) with perimeter 3
e1 = Equilateral (3) with perimeter 9
-----
i0 = Isosceles (Equilateral (1)) with perimeter 3
i1 = Isosceles (Equilateral (5)) with perimeter 15
i2 = Isosceles (6, 7) with perimeter 19
-----
s0 = Scalene (Equilateral (1)) with perimeter 3
s1 = Scalene (Equilateral (5)) with perimeter 15
s2 = Scalene (Isosceles (6, 7)) with perimeter 19
s3 = Scalene (7, 8, 9) with perimeter 24
```

```

// Workshop 7c - Classes and Objects
//
// - A class to describe three types of triangles
// - Inheritance
//   : Equilateral -> Isosceles -> Scalene
//
// - Virtual functions -> enjoy polymorphism

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

class Equilateral
{
    public:
        Equilateral ();

        Equilateral (int);

        virtual int perimeter () const;
        virtual void print () const;

    protected:
        int a;
};

class Isosceles : public Equilateral
{
    public:
        Isosceles ();
        Isosceles (int);
        Isosceles (int, int);

        int perimeter () const;
        void print () const;

    protected:
        int b;
};

class Scalene : public Isosceles
{
    public:
        Scalene ();
        Scalene (int);
        Scalene (int, int);
        Scalene (int, int, int);

        int perimeter () const;
        void print () const;

    private:
        int c;
};

int main ()

```

```

{
    Equilateral e (8);
    Isosceles i (6, 7);
    Scalene s (4, 5, 6);

    // an array of three Equilateral pointer
    Equilateral* array [3];

    // subclass objects are also objects theirs superclass
    array [0] = new Equilateral;
    array [0] = &e;

    array [1] = new Isosceles;
    array [1] = &i;

    array [2] = new Scalene;
    array [2] = &s;

    for (int i = 0; i < 3; i++)
    {
        array [i] -> print ();
        cout << " with perimeter "
             << array [i] -> perimeter ()
             << endl;
    }
}

Equilateral::Equilateral ()
{
    a = 1;
}

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

int Equilateral::perimeter () const
{
    return 3 * a;
}

void Equilateral::print () const
{
    cout << "Equilateral (" << a << ")";
}

// -----

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

Isosceles::Isosceles (int a) : Equilateral (a)

```

```

{
    b = 0;
}

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

int Isosceles::perimeter () const
{
    if (b == 0)
        return Equilateral::perimeter ();
    else
        return a + a + b;
}

void Isosceles::print () const
{
    cout << "Isosceles (" << a << ", " << b << ")";
}

// -----

Scalene::Scalene ()
{
    c = 0;
}

Scalene::Scalene (int a) : Isosceles (a)
{
    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 Scalene::perimeter () const
{
    if (b == 0)
        return Equilateral::perimeter ();
    else if (c == 0)
        return Isosceles::perimeter ();
    else
        return a + b + c;
}

void Scalene::print () const
{
    cout << "Scalene (" << a << ", " << b << ", " << c << ")";
}

```

Equilateral (8) with perimeter 24
Isosceles (6, 7) with perimeter 19
Scalene (4, 5, 6) with perimeter 15

```

// Workshop 7d - Classes and Objects
//
// - A class to describe three types of triangles
// - Inheritance
//   : Equilateral -> Isosceles -> Scalene
//
// - template class

#include <iostream>
#include <cmath>
#include <iomanip>
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);

        T perimeter () const;

    protected:
        T a;
};

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);

        T perimeter () const;

    protected:
        T b;
};

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

    public:

```

```

        Scalene ();
        Scalene (T);
        Scalene (T, T);
        Scalene (T, T, T);

        T perimeter () const;

    private:
        T c;
};

int main ()
{
    cout << fixed << showpoint << setprecision (3);

    Equilateral <int> e1 (3);
    Equilateral <double> e2 (8.8);
    Isosceles <int> i1 (4, 5);
    Isosceles <double> i2 (5.8, 9.89);
    Scalene <int> s1 (5, 6, 7);
    Scalene <double> s2 (4.5, 6.7 , 7.8);

    cout << e1 << endl;
    cout << e2 << endl;
    cout << i1 << endl;
    cout << i2 << endl;
    cout << s1 << endl;
    cout << s2 << endl;

}

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 ()
{
    a = 1;
}

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

```

```

template <class T>
T Equilateral <T>::perimeter () const
{
    return 3 * a;
}

// -----

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)
{
    b = 0;
}

template <class T>

```

```

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

template <class T>
T Isosceles <T>::perimeter () const
{
    T a = Equilateral <T>::a;

    if (b == 0)
        return Equilateral <T>::perimeter ();
    else
        return a + a + b;
}

// -----

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

    if (s.b == 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 << ")";
}

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

    if (s.b == 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 << ")";
}

```

```

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

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

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

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

template <class T>
T Scalene <T>::perimeter () const
{
    T a = Equilateral<T>::a;
    T b = Isosceles<T>::b;

    if (b == 0)
        return Equilateral<T>::perimeter ();
    else if (c == 0)
        return Isosceles<T>::perimeter ();
    else
        return a + b + c;
}

```

```

Equilateral (3)
Equilateral (8.800)
Isosceles (4, 5)
Isosceles (5.800, 9.890)
Scalene (5, 6, 7)
Scalene (4.500, 6.700, 7.800)

```