**Example 1: Program with a function template (one generic parameter)**

// This program uses a function template.

#include <iostream>

using namespace std;

// Template definition for square function.

template <class T>

T square(T number)

{

return number \* number;

}

int main()

{

int userInt; // To hold integer input

double userDouble; // To hold double input

auto var = 6; // Testing the template with an auto number

cout << "Enter an integer and a floating-point value: ";

cin >> userInt >> userDouble;

cout << "Here are their squares: ";

cout << square(userInt) << " and "

<< square(userDouble) << endl;

cout << "Here is the square of an auto variable: "

<< square(var) << endl; //... and it works!

}

**Example 2: Program with a function template (two generic parameters of the same “type”)**

// This program demonstrates the swapVars function template.

#include <iostream>

using namespace std;

template <class T>

void swapVars(T &var1, T &var2) // generic reference parameters

{

T temp;

temp = var1;

var1 = var2;

var2 = temp;

}

int main()

{

char firstChar, secondChar; // Two chars

int firstInt, secondInt; // Two ints

double firstDouble, secondDouble; // Two doubles

// Get and swapVars two chars

cout << "Enter two characters: ";

cin >> firstChar >> secondChar;

swapVars(firstChar, secondChar);

cout << firstChar << " " << secondChar << endl;

// Get and swapVars two ints

cout << "Enter two integers: ";

cin >> firstInt >> secondInt;

swapVars(firstInt, secondInt);

cout << firstInt << " " << secondInt << endl;

// Get and swapVars two doubles

cout << "Enter two floating-point numbers: ";

cin >> firstDouble >> secondDouble;

swapVars(firstDouble, secondDouble);

cout << firstDouble << " " << secondDouble << endl;

}

**Example 3: Program with a function template (two generic parameter of different “types”)**

// This program demonstrates a function template

// with two type parameters.

#include <iostream>

using namespace std;

template <class T1, class T2>

int largest(const T1 &var1, T2 &var2)

{

if (sizeof(var1) > sizeof(var2))

return sizeof(var1);

else

return sizeof(var2);

}

int main()

{

int i = 0;

char c = ' ';

float f = 0.0;

double d = 0.0;

cout << "Comparing an int and a double, the largest\n"

<< "of the two is " << largest(i, d) << " bytes.\n";

cout << "Comparing an char and a float, the largest\n"

<< "of the two is " << largest(c, f) << " bytes.\n";

}

**Example 4: Program with overloaded function templates**

// This program demonstrates an overloaded function template.

#include <iostream>

using namespace std;

template <class T>

T sum(T val1, T val2)

{

return val1 + val2;

}

template <class T>

T sum(T val1, T val2, T val3)

{

return val1 + val2 + val3;

}

int main()

{

double num1, num2, num3;

// Get two values and display their sum.

cout << "Enter two values: ";

cin >> num1 >> num2;

cout << "Their sum is " << sum(num1, num2) << endl;

// Get three values and display their sum.

cout << "Enter three values: ";

cin >> num1 >> num2 >> num3;

cout << "Their sum is " << sum(num1, num2, num3) << endl;

return 0;

}

**Example 5a: Class Template header file**

#ifndef RECTTEMP\_H

#define RECTTEMP\_H

template<class Z>

class RectTemp

{

private:

Z width;

Z length;

public:

void setWidth(Z w)

{ width = w; }

void setLength(Z len)

{ length = len;}

Z getWidth() const

{ return width; }

Z getLength() const

{ return length; }

Z getArea() const

{ return width \* length; }

};

#endif

**Example 5b: Class Template Driver Program**

#include <iostream>

#include "RectTemp.h"

using namespace std;

int main()

{

RectTemp<double> board; // Define an instance of the Rectangle class

RectTemp<int> card; // Define another instance of the Rectangle class

double rectWidth; // Local variable for width

double rectLength; // Local variable for length

int rectW; // Local variable for width

int rectL; // Local variable for length

// Get the rectangle's width and length from the user.

cout << "This program will use floating point values to \n"

<< "calculate the area of a board. \n";

cout << "What is the width? ";

cin >> rectWidth;

cout << "What is the length? ";

cin >> rectLength;

// Store the width and length of the rectangle

// in the box object.

board.setWidth(rectWidth);

board.setLength(rectLength);

// Display the rectangle's data.

cout << "Here is the board's data:\n";

cout << "Width: " << board.getWidth() << endl;

cout << "Length: " << board.getLength() << endl;

cout << "Area: " << board.getArea() << endl;

// Get the rectangle's width and length from the user.

cout << "This program will now use integer values to "

<< "calculate the area of a card.\n";

cout << "What is the width? ";

cin >> rectW;

cout << "What is the length? ";

cin >> rectL;

// Store the width and length of the rectangle

// in the box object.

card.setWidth(rectW);

card.setLength(rectL);

// Display the rectangle's data.

cout << "Here is the card's data:\n";

cout << "Width: " << card.getWidth() << endl;

cout << "Length: " << card.getLength() << endl;

cout << "Area: " << card.getArea() << endl;

}

**Example 6: Program with an iterator**

// This program provides a simple iterator demonstration.

#include <iostream>

#include <vector> // Include the vector header

using namespace std;

int main()

{

int count; // Loop counter

// Define a vector object

vector<int> vect;

// Define an iterator object

vector<int>::iterator iter;

// Use push\_back to push values into the vector.

for (count = 0; count < 10; count++)

vect.push\_back(count);

// Step the iterator through the vector,

// and use it to display the vector's contents.

cout << "Here are the values in vect: ";

for (iter = vect.begin(); iter < vect.end(); iter++)

{

cout << \*iter << " ";

}

// Step the iterator through the vector backwards.

cout << "\nand here they are backwards: ";

for (iter = vect.end() - 1; iter >= vect.begin(); iter--)

{

cout << \*iter << " ";

}

return 0;

}

**Example 7: STL Algorithms**

// A simple demonstration of STL algorithms

#include <iostream>

#include <vector> // Required for the vector type

#include <algorithm> // Required for STL algorithms

using namespace std;

int main()

{

int count; // Loop counter

// Define a vector object.

vector<int> vect;

// Use push\_back to push values into the vector.

for (count = 0; count < 10; count++)

vect.push\_back(count);

// Display the vector's elements.

cout << "The vector has " << vect.size()

<< " elements. Here they are:\n";

for (count = 0; count < vect.size(); count++)

cout << vect[count] << " ";

cout << endl;

// Randomly shuffle the vector's contents.

random\_shuffle(vect.begin(), vect.end());

// Display the vector's elements.

cout << "The elements have been shuffled:\n";

for (count = 0; count < vect.size(); count++)

cout << vect[count] << " ";

cout << endl;

// Now sort the vector's elements.

sort(vect.begin(), vect.end());

// Display the vector's elements again.

cout << "The elements have been sorted:\n";

for (count = 0; count < vect.size(); count++)

cout << vect[count] << " ";

cout << endl;

// Now search for an element with the value 7.

if (binary\_search(vect.begin(), vect.end(), 7))

cout << "The value 7 was found in the vector.\n";

else

cout << "The value 7 was not found in the vector.\n";

return 0;

}