// template1.cpp

// Objective - Find the absolute value of an int

#include <iostream>

using namespace std;

int findAbs(int myInt);

int main()

{

cout << findAbs (3) << endl;

cout << findAbs (-3) << endl;

}

int findAbs(int myInt)

{

return (myInt < 0 ? -myInt: myInt);

}

output

3

3

// template2.cpp

// Objective - Find the absolute value of an float

#include <iostream>

using namespace std;

float findAbs(float myFloat);

int main()

{

cout << findAbs (3.3) << endl;

cout << findAbs (-3.3) << endl;

}

float findAbs(float myFloat)

{

return (myFloat < 0.0 ? -myFloat: myFloat);

}

output

3.3

3.3

// template3.cpp

// template used for absolute value function

#include <iostream>

using namespace std;

template <class flexibleData> // template prototype

flexibleData findAbs(flexibleData n);

int main()

{

int int1 = 5;

int int2 = -6;

long long1 = 70000L;

long long2 = -80000L;

double double1 = 9.95;

double double2 = -10.15;

// calls instantiate functions

cout << "\nAbs(" << int1 << ")=" << findAbs(int1); // Abs(int)

cout << "\nAbs(" << int2 << ")=" << findAbs(int2); // Abs(int)

cout << "\nAbs(" << long1 << ")=" << findAbs(long1); // Abs(long)

cout << "\nAbs(" << long2 << ")=" << findAbs(long2); // Abs(long)

cout << "\nAbs(" << double1 << ")=" << findAbs(double1); // Abs(double)

cout << "\nAbs(" << double2 << ")=" << findAbs(double2); // Abs(double)

}

template <class flexibleData> // template prefix

flexibleData findAbs(flexibleData n)

{

return (n < 0) ? -n : n;

}

output

abs(5)=5

abs(-6)=6

abs(70000)=70000

abs(-80000)=80000

abs(9.95)=9.95

abs(-10.15)=10.15

// template4.cpp

// template used for function that finds number in array

#include <iostream>

using namespace std;

// function returns index number of item, or -1 if not found

template <class atype>

int find(atype\* array, atype value, int size);

char charArray[] = {'A', 'B', 'C', 'D', 'E', 'F'}; // array

char myChar= 'E'; // value to find

int intArray[] = {1, 3, 5, 9, 11, 13};

int myInt = 6;

long longArray[] = {1L, 3L, 5L, 9L, 11L, 13L};

long myLong = 11L;

double doubleArray[] = {1.0, 3.0, 5.0, 9.0, 11.0, 13.0};

double myDouble = 4.0;

int main()

{

cout << "\n E in charArray: index=" << find(charArray, myChar, 6);

cout << "\n 6 in intArray: index=" << find(intArray, myInt, 6);

cout << "\n11 in longArray: index=" << find(longArray, myLong, 6);

cout << "\n 4 in doubleArray: index=" << find(doubleArray, myDouble, 6);

}

template <class atype>

int find(atype\* array, atype value, int size)

{

for(int j=0; j<size; j++)

if(array[j]==value)

return j;

return -1;

}

output

E in charArray: index=4

6 in intArray: index=-1

11 in longArray: index=4

4 in doubleArray: index=-1

// template5.cpp

// template used for function that finds number in array

// what happens if the arguments don't match

#include <iostream>

using namespace std;

// function returns index number of item, or -1 if not found

template <class atype>

int find(atype\* array, atype value, int size);

char charArray[] = {'A', 'B', 'C', 'D', 'E', 'F'}; // array

char myChar= 'E'; // value to find

int intArray[] = {1, 3, 5, 9, 11, 13};

int myInt = 6;

long longArray[] = {1L, 3L, 5L, 9L, 11L, 13L};

long myLong = 11L;

double doubleArray[] = {1.0, 3.0, 5.0, 9.0, 11.0, 13.0};

double myDouble = 4.0;

int main()

{

cout << "\n E in charArray: index=" << find(charArray, myChar, 6);

// compilation errors occur when parameters don't match

// cout << "\n 6 in intArray: index=" << find(intArray, myLong, 6);

cout << "\n11 in longArray: index=" << find(longArray, myLong, 6);

// cout << "\n 4 in doubleArray: index=" << find(doubleArray, myInt, 6);

}

template <class atype>

int find(atype\* array, atype value, int size)

{

for(int j=0; j<size; j++)

if(array[j]==value)

return j;

return -1;

}

output

E in charArray: index=4

11 in longArray: index=4

// template6.cpp

// template used for function that finds number in array

// more than one template argument

#include <iostream>

using namespace std;

// function returns index number of item, or -1 if not found

template <class atype, class btype>

btype find(atype\* array, atype value, btype size);

int intArray[] = {1, 3, 5, 9, 11, 13};

int myInt = 6;

long longArray[] = {1L, 3L, 5L, 9L, 11L, 13L};

long myLong = 11L;

int main()

{

int intArraySize=6;

long longArraySize=6L;

cout << "\n 6 in intArray: index=" << find(intArray, myInt, longArraySize);

cout << "\n11 in longArray: index=" << find(longArray, myLong, intArraySize);

}

template <class atype, class btype>

btype find(atype\* array, atype value, btype size)

{

for(btype j=0; j<size; j++)

if(array[j]==value)

return j;

return (btype)-1;

}

output

6 in intArray: index=-1

11 in longArray: index=4

// template7.cpp

**#ifndef** H\_StackType

**#define** H\_StackType

**#include** <iostream>

**using** **namespace** std;

**template** <**class** **Type**>

**class** stackType

{

**public**:

**bool** **isEmptyStack**();

//Function returns true if the stack is empty;

//otherwise, it returns false.

**bool** **isFullStack**();

//Function returns true if the stack is full;

//otherwise, it returns false.

**void** **destroyStack**();

//Remove all elements from the stack

//Post: top = 0

**void** **push**(**const** **Type**& newItem);

//Add the newItem to the stack

//Post: stack is changed and the newItem

// is added to the top of stack

**void** **pop**(**Type**& poppedItem);

//Remove the top element of the stack

//Post: Stack is changed and the top element

// is removed from the stack. The top element

// of the stack is saved in poppedItem.

**stackType**(**int** stackSize = 100);

//constructor

//Create an array of size stackSize to hold the

// stack elements. The default stack size is 100

//Post: The variable list contains the base

// address of the array, top = 0 and

// maxStackSize = stackSize

**stackType**(**const** stackType<**Type**>& otherStack);

//copy constructor

**~stackType**();

//destructor

//Remove all elements from the stack

//Post: The array (list) holding the stack

// elements is deleted

**private**:

**int** maxStackSize; //variable to store the maximum stack size

**int** top; //variable to point to the top of the stack

**Type** \*list; //pointer to the array that holds

//the stack elements

};

**template**<**class** **Type**>

**void** **stackType<Type>::destroyStack**()

{

top = 0;

}//end destroyStack

**template**<**class** **Type**>

**bool** **stackType<Type>::isEmptyStack**()

{

**return**(top == 0);

}//end isEmptyStack

**template**<**class** **Type**>

**bool** **stackType<Type>::isFullStack**()

{

**return**(top == maxStackSize);

} //end isFullStack

**template**<**class** **Type**>

**stackType<Type>::stackType**(**int** stackSize)

{

**if**(stackSize <= 0)

{

cout<<"Size of the array to hold the stack must "

<<"be positive."<<**endl**;

cout<<"Creating an array of size 100."<<**endl**;

maxStackSize = 100;

}

**else**

maxStackSize = stackSize; //set the stack size to

//the value specified by

//the parameter stackSize

top = 0; //set top to 0

list = **new** **Type**[maxStackSize]; //create the array to

//hold the stack elements

}//end constructor

**template**<**class** **Type**>

**stackType<Type>::stackType**(**const** stackType<**Type**>& otherStack)

{

maxStackSize = otherStack.maxStackSize;

top = otherStack.top;

list = **new** **Type**[maxStackSize];

**for**(**int** index = 0; index < maxStackSize; index++)

{

list[index] = otherStack.list[index];

}

}

**template**<**class** **Type**>

**stackType<Type>::~stackType**() //destructor

{

**delete** [] list; //deallocate memory occupied by the array

}//end destructor

**template**<**class** **Type**>

**void** **stackType<Type>::push**(**const** **Type**& newItem)

{

list[top] = newItem; //add newItem at the top of the stack

top++; // increment the top

}//end push

**template**<**class** **Type**>

**void** **stackType<Type>::pop**(**Type**& poppedItem)

{

top--; //decrement the top

poppedItem = list[top]; //copy the top element of

//the stack into poppedItem

cout << "Popped item is " << poppedItem << endl;

}//end pop

**#endif**

// template7.cpp

//Program to test the various operations of a stack

//#include "template7.h"

**#include** <iostream>

**using** **namespace** std;

**int** **main**()

{

stackType<**int**> stack(50);

**int** poppedInt;

stack.push(23);

stack.push(45);

stack.push(38);

stackType<**int**> stackCopy(stack);

stack.pop(poppedInt);

stack.pop(poppedInt);

stack.pop(poppedInt);

cout << "stackCopy" << **endl**;

stackCopy.pop(poppedInt);

stackCopy.pop(poppedInt);

stackCopy.pop(poppedInt);

stack.push(33);

stack.push(44);

stack.push(55);

stackType<**int**> secondStack = stack;

stack.push(66);

secondStack.push(77);

cout << "secondStack" << **endl**;

secondStack.pop(poppedInt);

secondStack.pop(poppedInt);

secondStack.pop(poppedInt);

secondStack.pop(poppedInt);

stackType<**float**> floatStack; // floatStack is object of class Stack<float>

**float** poppedFloat;

floatStack.push(1111.1); // push 3 floats, pop 3 floats

floatStack.push(2222.2);

floatStack.push(3333.3);

floatStack.pop(poppedFloat);

floatStack.pop(poppedFloat);

floatStack.pop(poppedFloat);

stackType<**long**> longStack; // longStack is object of class Stack<long>

**long** poppedLong;

longStack.push(123123123L); // push 3 longs, pop 3 longs

longStack.push(234234234L);

longStack.push(345345345L);

longStack.pop(poppedLong);

longStack.pop(poppedLong);

longStack.pop(poppedLong);

**return** 0;

}

}

Output

Popped item is 38

Popped item is 45

Popped item is 23

stackCopy

Popped item is 38

Popped item is 45

Popped item is 23

secondStack

Popped item is 77

Popped item is 55

Popped item is 44

Popped item is 33

Popped item is 3333.3

Popped item is 2222.2

Popped item is 1111.1

Popped item is 345345345

Popped item is 234234234

Popped item is 123123123

// template8i.h

// illustrates using two class type parameters

#include <iostream>

using namespace std;

#include<cstdlib>

template <class T1, class T2>

class Pair

{

public:

// Default constructor

Pair();

Pair (T1 first\_value, T2 second\_value);

// Precondition: position is 1 or 2

// Postcondition: The position indicated has been set to value.

void set\_element( T1 value1, T2 value2);

// Precondition: position is 1 or 2

// Postcondition: The position indicated has been set to value.

void Print();

// Precondition: The position indicated has been set to value.

// Postcondition: Outputs the ordered pair in an order pair format

private:

T1 first; // First position of the ordered pair

T2 second; // Second position of the ordered pair

};

// Default template constructor initializes first and second position to zero.

template<class T1, class T2>

Pair<T1,T2>::Pair()

{

first = 0;

second = 0;

}

// Template constructor initializes first position to T1 first\_value, and

// second position to T2 value.

template<class T1, class T2>

Pair <T1,T2>::Pair (T1 first\_value, T2 second\_value)

{

first = first\_value;

second = second\_value;

}

// This function sets the values in the ordered pair

template <class T1, class T2>

void Pair <T1,T2>::set\_element( T1 value1, T2 value2)

{

first = value1;

second = value2;

}

// This outputs the ordered pair in a order pair format.

template<class T1, class T2>

void Pair <T1,T2>::Print()

{

cout <<"The ordered pair is ("<< first <<", "<< second<<")"<< endl;

}

// template8.cpp

// illustrates using two class type parameters

#include <iostream>

using namespace std;

#include<cstdlib>

#include "template8i.h"

int main()

{

Pair<int, int> intScores;

Pair <char,char> charSeats;

Pair<int, char> intScores2;

Pair <char,int> charSeats2;

intScores.set\_element(3,4);

intScores.Print();

charSeats.set\_element('a','b');

charSeats.Print();

intScores2.set\_element(3,'a');

intScores2.Print();

charSeats2.set\_element('a',3);

charSeats2.Print();

}

output

The ordered pair is (3, 4)

The ordered pair is (a, b)

The ordered pair is (3, a)

The ordered pair is (a, 3)

// template9

// pair::pair example

**#include** <utility> // pair, make\_pair

**#include** <string> // string

**#include** <iostream> // cout

**using** **namespace** std;

**int** **main** () {

pair <string,**double**> product1; // default constructor

pair <string,**double**> product2 ("tomatoes",2.30); // value init

pair <string,**double**> product3 (product2); // copy constructor

pair <string,**double**> product4; // default constructor

product1 = make\_pair(string("lightbulbs"),0.99); // using make\_pair (move)

product2.first = "shoes"; // the type of first is string

product2.second = 49.93; // the type of second is double

product4 = product3; // overloaded = operator

cout << "The price of " << product1.first << " is $" << product1.second << '\n';

cout << "The price of " << product2.first << " is $" << product2.second << '\n';

cout << "The price of " << product3.first << " is $" << product3.second << '\n';

cout << "The price of " << product3.first << " is $" << product3.second << '\n';

**return** 0;

}

Output

The price of lightbulbs is $0.99

The price of shoes is $49.93

The price of tomatoes is $2.3

The price of tomatoes is $2.3

// Vector1.cpp

// Dynamic arrays using the Standard Template Library

// Note the new and delete operators are never called

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<int> intVector;

int inputInt;

while (cin) //read until eof

{

cout <<"Enter a number - enter -1 when ready to input characters" << endl;

cin >> inputInt;

if (inputInt ==-1)

break;

else

intVector.push\_back(inputInt);

// push\_back member function inserts the value at the end vector object

}

cout << "\nNumber of integers inputted were " << intVector.size() << endl;

// reading the vector

cout << "The integers inputted were" << endl;

for (int index=0;index < intVector.size(); index++)

cout << intVector[index] << " " << intVector.at(index) << endl;

vector<char> charVector;

char inputChar;

while (cin) //read until eof

{

cout <<"Enter a character - cntl z when done" << endl;

cin >> inputChar;

if (cin)

charVector.push\_back(inputChar);

// push\_back member function inserts the value at the end vector object

}

cout << "\nNumber of characters inputted were " << charVector.size() << endl;

cout << "The characters inputted were" << endl;

for (int index=0;index < charVector.size(); index++)

cout << charVector[index] << " " << charVector.at(index) << endl;

return 0;

}

output

Enter a number - enter -1 when ready to input characters

3

Enter a number - enter -1 when ready to input characters

5

Enter a number - enter -1 when ready to input characters

8

Enter a number - enter -1 when ready to input characters

2

Enter a number - enter -1 when ready to input characters

-1

Number of integers inputted were 4

The integers inputted were

3 3

5 5

8 8

2 2

Enter a character - cntl z when done

a

Enter a character - cntl z when done

b

Enter a character - cntl z when done

c

Enter a character - cntl z when done

Number of characters inputted were 3

The characters inputted were

a a

b b

c c

// VECTOR2.cpp

// Dynamic arrays using the Standard Template Library

// Note the new and delete operators are never called

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<string> stringVector;

string temp = "Hello World";

stringVector.push\_back(temp);

cout << stringVector.at(0) << endl;

// changing a value in a vector

temp="Hello again";

stringVector.at(0)=temp;

cout << stringVector.at(0) << endl;

return 0;

}

output

Hello World

Hello again

// VECTOR3.cpp

// overloaded the << operator using the standard template library

/\* Include Files \*/

#include <iostream>

#include <vector>

#include <iterator>

using namespace std;

typedef vector<char> charVector;

// overload the << operator

int main()

{

charVector myVector;

int index;

myVector.reserve(10);

// store values into myVector

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

{

myVector.push\_back(index + '0');

}

// get the data

char outVector[10];

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

{

outVector[index]=myVector[index];

}

//output the vector

ostream\_iterator<char> outchar(cout," ");

// print vectors using the ostream objects

cout << "\nThe vector myVector now contains the elements " << endl;

copy (myVector.begin(),myVector.end(),outchar);

return 0;

}

output

The vector myVector now contains the elements

1. 1 2 3 4 5 6 7 8 9

//vector4.h

**#ifndef** VECTOR4\_H\_

**#define** VECTOR4\_H\_

//vector4.h

// create an Item class

**#include** <string>

**#include** <vector>

**#include** <iterator>

**#include** <iostream>

**using** **namespace** std;

**class** Item

{

//

**public**:

**Item**(); // constructor

**Item** (**int** passedId, string passedString);

**bool** **operator==**(**const** Item&obj)**const**;

**bool** **operator!=**(**const** Item&obj)**const**;

**void** **display**(**void**);

**void** **print** (vector<Item> myVector);

**private**:

**int** id;

string name;

};

**#endif**

//Vector4i.cpp

**#include** "vector4.h"

**Item::Item**() // constructor

{

id=0;

name=" " ;

}

**Item::Item** (**int** passedId, string passedString)

{

id=passedId;

name=passedString;

}

**bool** **Item::operator==**(**const** Item&obj)**const**

{

**return** (id==obj.id);

}

**bool** **Item::operator!=**(**const** Item&obj)**const**

{

**return** (id!=obj.id);

}

**void** **Item::display**(**void**)

{

cout << "The id is " << id << **endl**;

cout << "The name is " << name << **endl**;

}

**void** **Item::print** (vector<Item> myVector)

{

vector<Item>::iterator index;;

**for** (index=myVector.begin();index!= myVector.end(); index++)

{

index->display();

}

}

// VECTOR4.cpp

**#include** "vector4.h"

**int** **main**()

{

vector<Item> myVector;

Item myItem1(10,"sam");

Item myItem2(20,"bill");

Item myItem3(30,"jo ann");

Item myItem4(40,"mark");

Item myItem5(50,"mike");

myItem1.display();

myItem4.display();

myVector.push\_back(myItem1);

cout << "\nmyItem1 was put in the vector " << **endl**;

cout << "\nnumber of items in the vector = " << myVector.size() << **endl**;

myVector.push\_back(myItem2);

cout << "\nmyItem2 was put in the vector " << **endl**;

cout << "\nnumber of items in the vector = " << myVector.size() << **endl**;

myVector.push\_back(myItem3);

cout << "\nmyItem3 was put in the vector " << **endl**;

cout << "\nnumber of items in the vector = " << myVector.size() << **endl**;

myVector.push\_back(myItem4);

cout << "\nmyItem4 was put in the vector " << **endl**;

cout << "\nnumber of items in the vector = " << myVector.size() << **endl**;

;

cout << "do another insert" << **endl**;

vector<Item>::iterator index;

index=myVector.end();

myVector.insert(index,myItem5);

cout << "\nnumber of items in the vector = " << myVector.size() << **endl**;

cout << "Print vector now" << **endl**;

myItem3.print(myVector);

**return** 0;

}

output

The id is 10

The name is sam

The id is 40

The name is mark

myItem1 was put in the vector

number of items in the vector = 1

myItem2 was put in the vector

number of items in the vector = 2

myItem3 was put in the vector

number of items in the vector = 3

myItem4 was put in the vector

number of items in the vector = 4

do another insert

number of items in the vector = 5

Print vector now

The id is 10

The name is sam

The id is 20

The name is bill

The id is 30

The name is jo ann

The id is 40

The name is mark

The id is 50

The name is mike

// VECTOR5.cpp

// illustrates using the standard template library

/\* Include Files \*/

**#include** <iostream>

**#include** <vector>

**#include** <algorithm>

**#include** <iterator>

**using** **namespace** std;

**int** **main**()

{

**const** **int** NUMELS = 5;

**int** a[NUMELS] = {1,2,3,4,5};

**char** b[NUMELS]={'a','b','c','d','e'};

**int** index;

// instantiate an integer and character vector

// using a constructor to set the size of each vector

// and initialize each vector with values

vector<**int**> x(a,a+NUMELS);

vector<**char**> y(b,b+NUMELS);

vector<**int**> z(6,5);

vector<**int**>::iterator it;

cout << "\nThe vector x initially contains the elements: " << **endl**;

**for** (index=0;index < NUMELS;index++)

cout << x[index] << " ";

cout << "\nThe vector y initially contains the elements: " << **endl**;

**for** (index=0;index < NUMELS;index++)

cout << y[index] << " ";

// modify elements in the existing list

x.at(3)=6; // set element at position 3 to 6 (remember start at 0)

y.at(3)='g'; // set element at position 3 to 'g' (remember start at 0)

// instantiate two ostream objects

ostream\_iterator<**int**> outint(cout," ");

ostream\_iterator<**char**> outchar(cout," ");

// print vectors using the ostream objects

cout << "\nThe vector x before the insert contains the elements " << **endl**;

copy (x.begin(),x.end(),outint);

cout << "\nThe vector y before the insert contains the elements " << **endl**;

copy (y.begin(),y.end(),outchar);

// add elements to the list

x.insert(x.begin()+4,7); // insert a seven at position 4 (remember start at 0)

y.insert(y.begin()+2,'f'); // insert a 'f' at position 4

// print vectors using the ostream objects

cout << "\nThe vector x now contains the elements " << **endl**;

copy (x.begin(),x.end(),outint);

cout << "\nThe vector y now contains the elements " << **endl**;

copy (y.begin(),y.end(),outchar);

// sort both vectors

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

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

// print vectors using the ostream objects

cout << "\nThe vector x now contains the elements after sorting " << **endl**;

copy (x.begin(),x.end(),outint);

cout << "\nThe vector y now contains the elements after sorting " << **endl**;

copy (y.begin(),y.end(),outchar);

it=find(x.begin(),x.end(),6);

x.erase (it);

cout << "\nThe vector x now contains the elements after the find and erase " << **endl**;

copy (x.begin(),x.end(),outint);

// random shuffle of the existing elements

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

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

// print vectors using the ostream objects

cout << "\nThe vector x after the random shuffle contains the elements " << **endl**;

copy (x.begin(),x.end(),outint);

cout << "\nThe vector y random shuffle contains the elements " << **endl**;

copy (y.begin(),y.end(),outchar);

// sort the first three elements of x vector

sort(x.begin(),x.begin()+3);

cout << "\nThe vector x now contains the elements after sorting " << **endl**;

copy (x.begin(),x.end(),outint);

// use the operators

cout << "\nThe vector z now contains the elements " << **endl**;

copy (z.begin(),z.end(),outint);

// use the vector operators

**if** (x < z)

cout << "\nx is less" << **endl**;

**else**

cout << "z is less" << **endl**;

z=x; // assignment

cout << "\nThe vector z now contains the elements after the assignment operator " << **endl**;

copy (z.begin(),z.end(),outint);

}

output

The vector x initially contains the elements:

1 2 3 4 5

The vector y initially contains the elements:

a b c d e

The vector x before the insert contains the elements

1 2 3 6 5

The vector y before the insert contains the elements

a b c g e

The vector x now contains the elements

1 2 3 6 7 5

The vector y now contains the elements

a b f c g e

The vector x now contains the elements after sorting

1 2 3 5 6 7

The vector y now contains the elements after sorting

a b c e f g

The vector x now contains the elements after the find and erase

1 2 3 5 7

The vector x after the random shuffle contains the elements

7 2 5 3 1

The vector y random shuffle contains the elements

a c e f g b

The vector x now contains the elements after sorting

2 5 7 3 1

The vector z now contains the elements

5 5 5 5 5 5

x is less

The vector z now contains the elements after the assignment operator

2 5 7 3 1

// Vector6.cpp

// comparing size, capacity and max\_size

**#include** <iostream>

**#include** <vector>

**int** **main** ()

{

std::vector<**int**> myvector;

std::cout << "capacity: " << myvector.capacity() << "\n";

// set some content in the vector:

**for** (**int** i=0; i<100; i++) myvector.push\_back(i);

std::cout << "size: " << myvector.size() << "\n";

std::cout << "capacity: " << myvector.capacity() << "\n";

std::cout << "max\_size: " << myvector.max\_size() << "\n";

**return** 0;

}

Output

capacity: 0

size: 100

capacity: 128

max\_size: 1073741823

// Vector7.cpp

// inserting into a vector

// inserting into a vector

//The vector is extended by inserting new elements before the element at the //specified position, effectively increasing the container size by the number of //elements inserted.

**#include** <iostream>

**#include** <vector>

**int** **main** ()

{

std::vector<**int**> myvector (3,100);

std::vector<**int**>::iterator it;

//print out the vector

std::cout << "myvector contains:";

**for** (it=myvector.begin(); it<myvector.end(); it++)

std::cout << ' ' << \*it;

std::cout << '\n';

it = myvector.begin();

it = myvector.insert ( it , 200 );

//print out the vector

std::cout << "myvector contains:";

**for** (it=myvector.begin(); it<myvector.end(); it++)

std::cout << ' ' << \*it;

std::cout << '\n';

myvector.insert (it,2,300);

//print out the vector

std::cout << "myvector contains:";

**for** (it=myvector.begin(); it<myvector.end(); it++)

std::cout << ' ' << \*it;

std::cout << '\n';

it = myvector.begin();

std::vector<**int**> anothervector (2,400);

myvector.insert (it+2,anothervector.begin(),anothervector.end());

//print out the vector

std::cout << "myvector contains:";

**for** (it=myvector.begin(); it<myvector.end(); it++)

std::cout << ' ' << \*it;

std::cout << '\n';

**int** myarray [] = { 501,502,503 };

myvector.insert (myvector.begin(), myarray, myarray+3);

std::cout << "myvector contains:";

**for** (it=myvector.begin(); it<myvector.end(); it++)

std::cout << ' ' << \*it;

std::cout << '\n';

**return** 0;

}

Output

myvector contains: 100 100 100

myvector contains: 200 100 100 100

myvector contains: 200 100 100 100 300 300

myvector contains: 200 100 400 400 100 100 300 300

myvector contains: 501 502 503 200 100 400 400 100 100 300 300

// Vector 8.cpp

// vector::rbegin/rend

//reverse iterator

**#include** <iostream>

**#include** <vector>

**int** **main** ()

{

std::vector<**int**> myvector (5); // 5 default-constructed ints

**int** i=0;

std::vector<**int**>::reverse\_iterator rit = myvector.rbegin();

**for** (; rit!= myvector.rend(); ++rit)

\*rit = ++i;

std::cout << "myvector contains:";

**for** (std::vector<**int**>::iterator it = myvector.begin(); it != myvector.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

**return** 0;

}

Output

myvector contains: 5 4 3 2 1

// Vector 9.cpp

// erasing from vector

**#include** <iostream>

**#include** <vector>

**int** **main** ()

{

std::vector<**int**> myvector;

// set some values (from 1 to 10)

**for** (**int** i=1; i<=10; i++) myvector.push\_back(i);

std::cout << "myvector contains:";

**for** (**unsigned** i=0; i<myvector.size(); ++i)

std::cout << ' ' << myvector[i];

std::cout << '\n';

// erase the 6th element

myvector.erase (myvector.begin()+5);

std::cout << "myvector contains:";

**for** (**unsigned** i=0; i<myvector.size(); ++i)

std::cout << ' ' << myvector[i];

std::cout << '\n';

// erase the first 3 elements:

myvector.erase (myvector.begin(),myvector.begin()+3);

std::cout << "myvector contains:";

**for** (**unsigned** i=0; i<myvector.size(); ++i)

std::cout << ' ' << myvector[i];

std::cout << '\n';

**return** 0;

}

Output

myvector contains: 1 2 3 4 5 6 7 8 9 10

myvector contains: 1 2 3 4 5 7 8 9 10

myvector contains: 4 5 7 8 9 10

// Vector 10

// copy algorithm example

**#include** <iostream> // std::cout

**#include** <algorithm> // std::copy

**#include** <vector> // std::vector

**int** **main** () {

**int** myints[]={10,20,30,40,50,60,70};

std::vector<**int**> myvector (7);

std::copy ( myints, myints+7, myvector.begin() );

std::cout << "myvector contains:";

**for** (std::vector<**int**>::iterator it = myvector.begin(); it!=myvector.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

**return** 0;

}

Output

myvector contains: 10 20 30 40 50 60 70

// Vector 11.cpp

#include <iostream>

#include <string>

#include <vector>

using namespace std;

class base

{

public:

base ();

base (int myType, int myValue);

virtual void show() const;

private:

int type;

int value;

};

base::base ():type(0),value(0)

{

}

base::base (int myType,int myValue):type(myType),value(myValue)

{

}

void base::show () const

{

cout << "base show" << endl;

cout << "type is " << type << endl;

cout <<"value is " << value << endl;

}

class derived : public base

{

public:

derived ();

derived (int myType, int myValue, int myType2,int myValue2);

void show() const;

private:

int type2;

int value2;

};

derived::derived ():type2(0),value2(0)

{

}

derived::derived(int myType, int myValue, int myType2,int myValue2):base(myType,myValue)

{

type2=myType2;

value2=myValue2;

}

void derived::show () const

{

base::show();

cout << "derived show" << endl;

cout << "type is " << type2 << endl;

cout <<"value is " << value2 << endl;

}

typedef vector<base> Vector;

int main()

{

Vector V;

base baseItem(3,6);

derived derivedItem(1,2,9,12);

// insert item

V.push\_back(baseItem);

V.push\_back(derivedItem);

Vector::iterator index;;

for (index=V.begin(); index !=V.end(); index++)

index->show();

}

output

base show

type is 3

value is 6

base show

type is 1

value is 2

//vector 12.cpp

#include <iostream>

#include <string>

#include <vector>

using namespace std;

class base

{

public:

base ();

base (int myType, int myValue);

virtual void show() const;

private:

int type;

int value;

};

base::base ():type(0),value(0)

{

}

base::base (int myType,int myValue):type(myType),value(myValue)

{

}

void base::show () const

{

cout << "base show" << endl;

cout << "type is " << type << endl;

cout <<"value is " << value << endl;

}

class derived : public base

{

public:

derived ();

derived (int myType, int myValue, int myType2,int myValue2);

void show() const;

private:

int type2;

int value2;

};

derived::derived ():type2(0),value2(0)

{

}

derived::derived(int myType, int myValue, int myType2,int myValue2):base(myType,myValue)

{

type2=myType2;

value2=myValue2;

}

void derived::show () const

{

base::show();

cout << "derived show" << endl;

cout << "type is " << type2 << endl;

cout <<"value is " << value2 << endl;

}

typedef vector<base \*> Vector;

int main()

{

Vector V;

base baseItem(3,6);

base \* basePointer= &baseItem;

derived derivedItem(1,2,9,12);

derived \* derivedPointer= &derivedItem;

// insert item

V.push\_back(basePointer); // address

V.push\_back(derivedPointer); //address

Vector::iterator index;;

for (index=V.begin(); index !=V.end(); index++)

(\*index)->show();

}

output

base show

type is 3

value is 6

base show

type is 1

value is 2

derived show

type is 9

value is 12

// VECTOR13.cpp

//============================================================================

/// find example

**#include** <iostream> // std::cout

**#include** <algorithm> // std::find

**#include** <vector> // std::vector

**int** **main** () {

// using std::find with array and pointer:

**int** myints[] = { 10, 20, 30, 40 };

**int** \* p;

p = std::find (myints, myints+4, 30);

**if** (p != myints+4)

std::cout << "Element found in myints: " << \*p << '\n';

**else**

std::cout << "Element not found in myints\n";

// using std::find with vector and iterator:

std::vector<**int**> myvector (myints,myints+4);

std::vector<**int**>::iterator it;

it = find (myvector.begin(), myvector.end(), 30);

**if** (it != myvector.end())

std::cout << "Element found in myvector: " << \*it << '\n';

**else**

std::cout << "Element not found in myvector\n";

**return** 0;

}

Output

Element found in myints: 30

Element found in myvector: 30

// Stack1.cpp

// stacks using STL

#include <iostream>

#include <string>

#include <stack>

using namespace std;

int main()

{

stack<string> stringStack; // stack is LIFO

string myString="first string";

string myString2="second String";

cout << myString << endl;

cout << myString2 << endl;

stringStack.push(myString);

stringStack.push(myString2);

string outputString = stringStack.top();

stringStack.pop();

cout << "popped value is " << outputString << endl;

outputString = stringStack.top();

stringStack.pop();

cout << "popped value is " << outputString << endl;

return 0;

}

output

first string

second String

popped value is second String

popped value is first string

*// tuple1.cpp*

*#include <iostream> // std::cout*

*#include <tuple> // std::tuple, std::get, std::tie, std::ignore*

*int* main ()

{

std::tuple<*int*,*char*> foo (10,'x');

*auto* bar = std::make\_tuple ("test", 3.1, 14, 'y');

std::get<2>(bar) = 100; *// access element*

*int* myint; *char* mychar;

std::tie (myint, mychar) = foo; *// unpack elements*

std::tie (std::ignore, std::ignore, myint, mychar) = bar; *// unpack (with ignore)*

mychar = std::get<3>(bar);

std::get<0>(foo) = std::get<2>(bar);

std::get<1>(foo) = mychar;

std::cout << "foo contains: ";

std::cout << std::get<0>(foo) << ' ';

std::cout << std::get<1>(foo) << '\n';

*return* 0;

}

Output

foo contains: 100

// DEQUE **1.cpp**

// deque::pop\_front

**#include** <iostream>

**#include** <deque>

**int** **main** ()

{

std::deque<**int**> mydeque;

mydeque.push\_back (100);

mydeque.push\_back (200);

mydeque.push\_back (300);

std::cout << "Popping out the elements in mydeque:";

**while** (!mydeque.empty())

{

std::cout << ' ' << mydeque.front();

mydeque.pop\_front();

}

std::cout << "\nThe final size of mydeque is " << **int**(mydeque.size()) << '\n';

**return** 0;

}

Output

Popping out the elements in mydeque: 100 200 300

The final size of mydeque is 0

**//** DEQUE 2**.cpp**

// deque::begin

**#include** <iostream>

**#include** <deque>

**int** **main** ()

{

std::deque<**int**> mydeque;

**for** (**int** i=1; i<=5; i++) mydeque.push\_back(i);

std::cout << "mydeque contains:";

std::deque<**int**>::iterator it = mydeque.begin();

**while** (it != mydeque.end())

std::cout << ' ' << \*it++;

std::cout << '\n';

**return** 0;

}

Output

mydeque contains: 1 2 3 4 5

// DEQUE 3**.cpp**

// inserting into a deque

**#include** <iostream>

**#include** <deque>

**#include** <vector>

**int** **main** ()

{

std::deque<**int**> mydeque;

// set some initial values:

**for** (**int** i=1; i<6; i++) mydeque.push\_back(i); // 1 2 3 4 5

std::deque<**int**>::iterator it = mydeque.begin();

++it;

it = mydeque.insert (it,10); // 1 10 2 3 4 5

// "it" now points to the newly inserted 10

std::cout << "mydeque contains:";

**for** (it=mydeque.begin(); it!=mydeque.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

mydeque.insert (it,2,20); // 1 20 20 10 2 3 4 5

// "it" no longer valid!

std::cout << "mydeque contains:";

**for** (it=mydeque.begin(); it!=mydeque.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

it = mydeque.begin()+2;

std::vector<**int**> myvector (2,30);

mydeque.insert (it,myvector.begin(),myvector.end());

// 1 20 30 30 20 10 2 3 4 5

std::cout << "mydeque contains:";

**for** (it=mydeque.begin(); it!=mydeque.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

**return** 0;

}

**Output**

mydeque contains: 1 10 2 3 4 5

mydeque contains: 1 10 2 3 4 5 20 20

mydeque contains: 1 10 30 30 2 3 4 5 20 20

**//LIST 1.cpp**

// list::begin

**#include** <iostream>

**#include** <list>

**int** **main** ()

{

**int** myints[] = {75,23,65,42,13};

std::list<**int**> mylist (myints,myints+5);

std::cout << "mylist contains:";

**for** (std::list<**int**>::iterator it=mylist.begin(); it != mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

**return** 0;

}

**Output**

mylist contains: 75 23 65 42 13

**//LIST 2.cpp**

// inserting into a list

**#include** <iostream>

**#include** <list>

**#include** <vector>

**int** **main** ()

{

std::list<**int**> mylist;

std::list<**int**>::iterator it;

// set some initial values:

**for** (**int** i=1; i<=5; ++i) mylist.push\_back(i); // 1 2 3 4 5

std::cout << "mylist contains:";

**for** (it=mylist.begin(); it!=mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

it = mylist.begin();

++it; // it points now to number 2 ^

mylist.insert (it,10); // 1 10 2 3 4 5

// "it" still points to number 2 ^

mylist.insert (it,2,20); // 1 10 20 20 2 3 4 5

--it; // it points now to the second 20 ^

std::vector<**int**> myvector (2,30);

mylist.insert (it,myvector.begin(),myvector.end());

// 1 10 20 30 30 20 2 3 4 5

// ^

std::cout << "mylist contains:";

**for** (it=mylist.begin(); it!=mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

**return** 0;

}

**Output**

mylist contains: 1 2 3 4 5

mylist contains: 1 10 20 30 30 20 2 3 4 5

**//LIST 3.cpp**

|  |  |
| --- | --- |
| // splicing lists  #include <iostream>  #include <list>  int main ()  {  std::list<int> mylist1, mylist2;  std::list<int>::iterator it;  // set some initial values:  for (int i=1; i<=4; ++i)  mylist1.push\_back(i); // mylist1: 1 2 3 4  for (int i=1; i<=3; ++i)  mylist2.push\_back(i\*10); // mylist2: 10 20 30  it = mylist1.begin();  ++it; // points to 2  mylist1.splice (it, mylist2); // mylist1: 1 10 20 30 2 3 4  // mylist2 (empty)  // "it" still points to 2 (the 5th element)    mylist2.splice (mylist2.begin(),mylist1, it);  // mylist1: 1 10 20 30 3 4  // mylist2: 2  // "it" is now invalid.  it = mylist1.begin();  std::advance(it,3); // "it" points now to 30  mylist1.splice ( mylist1.begin(), mylist1, it, mylist1.end());  // mylist1: 30 3 4 1 10 20  std::cout << "mylist1 contains:";  for (it=mylist1.begin(); it!=mylist1.end(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  std::cout << "mylist2 contains:";  for (it=mylist2.begin(); it!=mylist2.end(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  return 0;  } |  |

Output:

|  |
| --- |
| mylist1 contains: 30 3 4 1 10 20  mylist2 contains: 2 |

//REVERSE1.cpp

|  |  |
| --- | --- |
| // vector::rbegin/rend  **#include** <iostream>  **#include** <vector>  **int** **main** ()  {  std::vector<**int**> myvector (5); // 5 default-constructed ints  **int** i=0;  std::vector<**int**>::reverse\_iterator rit = myvector.rbegin();  **for** (; rit!= myvector.rend(); ++rit)  \*rit = ++i;  std::cout << "myvector contains:";  **for** (std::vector<**int**>::iterator it = myvector.begin(); it != myvector.end(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  **return** 0;  } |  |

Output:

|  |
| --- |
| myvector contains: 5 4 3 2 1 |
|  |

**//ALGORITHM 1.cpp**

|  |  |
| --- | --- |
| // sort algorithm example  **#include** <iostream> // std::cout  **#include** <algorithm> // std::sort  **#include** <vector> // std::vector  **bool** **myfunction** (**int** i,**int** j) { **return** (i<j); }  **struct** myclass {  **bool** **operator()** (**int** i,**int** j) { **return** (i<j);}  } myobject;  **int** **main** () {  **int** myints[] = {32,71,12,45,26,80,53,33};  std::vector<**int**> myvector (myints, myints+8); // 32 71 12 45 26 80 53 33  // using default comparison (operator <):  std::sort (myvector.begin(), myvector.begin()+4); //(12 32 45 71)26 80 53 33  // using function as comp  std::sort (myvector.begin()+4, myvector.end(), myfunction); // 12 32 45 71(26 33 53 80)  // using object as comp  std::sort (myvector.begin(), myvector.end(), myobject); //(12 26 32 33 45 53 71 80)  // print out content:  std::cout << "myvector contains:";  **for** (std::vector<**int**>::iterator it=myvector.begin(); it!=myvector.end(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  **return** 0;  } |  |

Output:

|  |
| --- |
| myvector contains: 12 26 32 33 45 53 71 80 |

**// ALGORITHM2.cpp**

|  |  |
| --- | --- |
| // count\_if example  **#include** <iostream> // std::cout  **#include** <algorithm> // std::count\_if  **#include** <vector> // std::vector  **bool** **IsOdd** (**int** i) { **return** ((i%2)==1); }  **int** **main** () {  std::vector<**int**> myvector;  **for** (**int** i=1; i<10; i++) myvector.push\_back(i); // myvector: 1 2 3 4 5 6 7 8 9  **int** mycount = count\_if (myvector.begin(), myvector.end(), IsOdd);  std::cout << "myvector contains " << mycount << " odd values.\n";  **return** 0;  } |  |

Output:

|  |
| --- |
| myvector contains 5 odd value |

**//ALGORITHM3.cpp**

// accumulate example

**#include** <iostream> // cout

**#include** <functional> // minus

**#include** <numeric> // accumulate

**#include** <algorithm>

**using** **namespace** std;

**int** **myfunction** (**int** x, **int** y) {**return** x+2\*y;}

**struct** myclass {

**int** **operator()**(**int** x, **int** y) {**return** x+3\*y;}

} myobject;

**int** **main** () {

**int** init = 100;

**int** numbers[] = {10,30,50};

cout << "using default accumulate: ";

cout << accumulate(numbers,numbers+3,init); // 100 + 10 + 30 + 50

cout << '\n';

cout << "using functional's minus: ";

cout << accumulate (numbers, numbers+3, init, minus<**int**>()); // 100 - (10 + 30 + 50)

cout << '\n';

cout << "using custom function: ";

cout << accumulate (numbers, numbers+3, init, myfunction); //100 +2\*(10 + 30 + 50)

cout << '\n';

cout << "using custom object: ";

cout << accumulate (numbers, numbers+3, init, myobject);//100 +3\*(10 + 30 + 50)

cout << '\n';

**return** 0;

}

Output:

|  |
| --- |
| using default accumulate: 190  using functional's minus: 10  using custom function: 280  using custom object: 370 |

**//FUNCTION1.cpp**

//Function Objects

**#include** <iostream>

**#include** <string>

**#include** <algorithm>

**#include** <numeric>

**#include** <iterator>

**#include** <vector>

**#include** <functional>

**using** **namespace** std;

**int** **funcAdd**(plus<**int**>, **int**, **int**);

**int** **main**()

{

plus<**int**> addNum; //Line 1

**int** num = addNum(34, 56); //Line 2

cout << "Line 3: num = " << num << **endl**; //Line 3

plus<string> joinString; //Line 4

string str1 = "Hello "; //Line 5

string str2 = "There"; //Line 6

string str = joinString(str1, str2); //Line 7

cout << "Line 8: str = " << str << **endl**; //Line 8

cout << "Line 9: Sum of 34 and 26 = "

<< funcAdd(addNum, 34, 26) << **endl**; //Line 9

**int** list[8] = {1, 2, 3, 4, 5, 6, 7, 8}; //Line 10

vector<**int**> intList(list, list + 8); //Line 11

ostream\_iterator<**int**> screenOut(cout, " "); //Line 12

cout << "Line 13: intList: "; //Line 13

copy(intList.begin(), intList.end(), screenOut); //Line 14

cout << **endl**; //Line 15

//accumulate function

**int** sum = accumulate(intList.begin(),

intList.end(), 0); //Line 16

cout << "Line 17: Sum of the elements of "

<< "intList = " << sum << **endl**; //Line 17

**int** product = accumulate(intList.begin(),

intList.end(),

1, multiplies<**int**>()); //Line 18

cout << "Line 19: Product of the elements of "

<< "intList = " << product << **endl**; //Line 19

**return** 0;

}

**int** **funcAdd**(plus<**int**> sum, **int** x, **int** y)

{

**return** sum(x, y);

}

Output

Line 3: num = 90

Line 8: str = Hello There

Line 9: Sum of 34 and 26 = 60

Line 13: intList: 1 2 3 4 5 6 7 8

Line 17: Sum of the elements of intList = 36

Line 19: Product of the elements of intList = 40320