**Program No:** 01

**Program Topic:** *VECTORS*

**Program Title:**

Write a C++ program doing basic operations of a vector.

**Objective:**

To learn about basic operations of a vector.

**Syntax:**

template < class T, class Allocator = allocator <T>> class vector

**Source Code:**

# include <iostream>

# include <vector>

using namespace std ;

int main ()

{

vector <int > v;

int i;

cout << "Size = " << v. size () << endl ;

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

v. push\_back (i);

cout << "Size now = " << v. size () << endl ;

cout << "Current contents :\n";

for (i =0; i<v. size (); i++)

cout << v[i] << " ";

cout << endl ;

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

v. push\_back (i +10) ;

cout << "Size now = " << v. size () << endl;

cout << "Current contents :\n";

for (i =0; i<v. size (); i++)

cout << v[i] << " ";

cout << endl;

for (i =0; i<v. size (); i++)

v[i] = v[i] + v[i];

cout << "Current contents :\n";

for (i =0; i<v. size (); i++)

cout << v[i] << " ";

cout << endl;

return 0;

}

**Output:**

Size = 0

Size now = 10

Current contents :

0 1 2 3 4 5 6 7 8 9

Size now = 20

Current contents :

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Current contents :

0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38

Process returned 0 (0x0) execution time : 0.180 s

Press any key to continue.

**Explanation:**

The **vector** class supports a dynamic array. This is an array that can grow as needed. in C++ the size of an array is fixed at compile time. Although this is by far the most efficient way to implement arrays, it is also the most restrictive, because the size of the array cannot be adjusted at run time to accommodate changing program conditions. A vector solves this problem by allocating memory as needed. Although a vector is dynamic, it can be still used the standard array subscript notation to access its elements.

**Program No:** 02

**Program Topic:** *VECTORS*

**Program Title:**

Write a C++ program to do basic operations of vectors by using an iterator.

**Objective:**

To learn how to use iterator for vectors.

**Syntax:**

vector <data type>:: iterator p = v. begin ();

// … … …

**Source Code:**

# include <iostream>

# include <vector>

using namespace std ;

int main ()

{

vector <int > v;

int i;

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

v.push\_back (i);

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

cout << v[i] << " ";

cout << endl;

vector <int >:: iterator p = v. begin ();

while (p != v. end ()){

cout << \*p << " ";

p ++;

}

return 0;

}

**Output:**

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

Process returned 0 (0x0) execution time : 0.259 s

Press any key to continue.

**Explanation:**

An array can be accessed either through subscripting or through a pointer. The parallel to this in the STL is the link between vectors and iterators. The members of a vector can be accessed by using subscripting or by using an iterator.

**Program No:** 03

**Program Topic:** *VECTORS*

**Program Title:**

Write a C++ program doing more operations with vector using insert() and erase() function.

**Objective:**

To learn about insert() and erase() function of a vector.

**Syntax:**

v. insert(iterator, no. of insert elements, value);

v. erase(iterator, next no. of elements to be removed);

**Source Code:**

# include <iostream>

# include <vector>

using namespace std ;

int main ()

{

vector <int > v(10, 3);

int i;

cout << "Size = " << v. size () << endl ;

cout << "Original contents :\n";

for (i =0; i<v. size (); i++)

cout << v[i] << " ";

cout << endl << endl;

vector <int >:: iterator p = v. begin ();

p += 2;

v.insert (p, 10, 9);

cout << "Size after insert = " << v. size () << endl;

cout << "Contents after insert :\n";

for (i =0; i<v. size (); i++)

cout << v[i] << " ";

cout << endl << endl;

p = v. begin ();

p += 2;

v. erase (p, p +10);

cout << "Size after erase = " << v. size () << endl ;

cout << "Contents after erase :\n";

for (i =0; i<v. size (); i++)

cout << v[i] << " ";

cout << endl;

return 0;

}

**Output:**

Size = 10

Original contents :

3 3 3 3 3 3 3 3 3 3

Size after insert = 20

Contents after insert :

3 3 9 9 9 9 9 9 9 9 9 9 3 3 3 3 3 3 3 3

Size after erase = 10

Contents after erase :

3 3 3 3 3 3 3 3 3 3

Process returned 0 (0x0) execution time : 0.133 s

Press any key to continue.

**Explanation:**

In addition to putting new values on the end of a vector, Elements can be inserted into the  
middle using the **insert()** function, can also be remove elements using **erase()**.

**Program No:** 04

**Program Topic:** *VECTORS*

**Program Title:**

Write a C++ program storing a class object in a vector.

**Objective:**

To learn how to store a class object in a vector.

**Syntax:**

vector<className > v;

**Source Code:**

# include <iostream>

# include <vector>

using namespace std ;

class D

{

double d;

public :

D () { d = 0.0; }

D ( double x) { d = x; }

D & operator =( double x)

{

d = x;

return \* this ;

}

double getd () { return d; }

};

bool operator <( D a, D b)

{

return a. getd () < b. getd ();

}

bool operator ==( D a, D b)

{

return a. getd () == b. getd ();

}

int main ()

{

vector <D > v;

int i;

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

v. push\_back ( D (i \* 3.0) );

for (i =0; i<v. size (); i++)

cout << v[i]. getd () << " ";

cout << endl ;

for (i =0; i<v. size (); i++)

v[i] = v[i]. getd () \* 2.1;

for (i =0; i<v. size (); i++)

cout << v[i]. getd () << " ";

return 0;

}

**Output:**

0 3 6 9 12 15 18 21 24 27

0 6.3 12.6 18.9 25.2 31.5 37.8 44.1 50.4 56.7

Process returned 0 (0x0) execution time : 0.121 s

Press any key to continue.

**Program No:** 05

**Program Topic:** *LISTS*

**Program Title:**

Write a C++ program using lists.

**Objective:**

To learn about lists.

**Syntax:**

template < class T, class Allocator = allocator <T>> class list

**Source Code:**

# include <iostream>

# include <list>

using namespace std ;

int main ()

{

list <char> l;

int i;

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

l . push\_back (’A’+i);

cout << "Size = " << l . size () << endl ;

list <char >:: iterator p;

cout << "Contents : ";

while (! l . empty ()){

p = l . begin ();

cout << \*p;

l. pop\_front ();

}

return 0;

}

**Output:**

Size = 10

Contents : ABCDEFGHIJ

Process returned 0 (0x0) execution time : 0.263 s

Press any key to continue.

**Explanation:**

The **list** class supports a bidirectional, linear list. Unlike a vector, which supports random  
access, a list can be accessed sequentially only. Because lists are bidirectional, they can be  
accessed front to back or back to front.

**Program No:** 06

**Program Topic:** *LISTS*

**Program Title:**

Write a C++ program to reverse a list.

**Objective:**

To learn how to reverse a list.

**Syntax:**

list<dataType>listName

//… … …

**Source Code:**

# include <iostream>

# include <list>

using namespace std ;

int main ()

{

list <char > l ;

list <char > revl ;

int i;

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

l . push\_back ('N'+i);

cout << "Size of list = " << l . size () << endl ;

cout << "Original contents : ";

list <char >:: iterator p;

while (! l . empty ())

{

p = l . begin ();

cout << \*p;

l . pop\_front ();

revl . push\_front (\*p);

}

cout << endl << endl ;

cout << "Size of reversed list = ";

cout << revl . size () << endl;

cout << "Reversed contents : ";

p = revl . begin ();

while (p != revl . end ())

{

cout << \*p;

p ++;

}

return 0;

}

**Output:**

Size of list = 10

Original contents : NOPQRSTUVW

Size of reversed list = 10

Reversed contents : WVUTSRQPON

Process returned 0 (0x0) execution time : 0.515 s

Press any key to continue.

**Explanation:**

Because lists are bidirectional, elements can be put on a list either at the front or at the  
back.

**Program No:** 07

**Program Topic:** *LISTS*

**Program Title:**

Write a C++ program to sort a list.

**Objective:**

To learn how to sort a list and to learn about sort() function.

**Syntax:**

listName. sort();

… … …

**Source Code:**

# include <iostream>

# include <list>

# include <cstdlib>

using namespace std;

int main ()

{

list <char > lst;

int i;

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

lst.push\_back ('A' + ( rand () %26) );

cout << "Original contents : ";

list <char >:: iterator p = lst . begin ();

while (p != lst . end ()){

cout << \*p;

p ++;

}

cout << endl << endl;

lst . sort ();

cout << "Sorted contents : ";

p = lst . begin ();

while (p != lst . end ()){

cout << \*p;

p ++;

}

return 0;

}

**Output:**

Original contents : PHQGHUMEAY

Sorted contents : AEGHHMPQUY

Process returned 0 (0x0) execution time : 0.061 s

Press any key to continue.

**Explanation:**

A list can be sorted by using a sort function.

**Program No:** 08

**Program Topic:***LISTS*

**Program Title:**

Write a C++ program to merge two lists.

**Objective:**

To learn how to merge two lists.

**Source Code:**

# include <iostream>

# include <list>

using namespace std ;

int main ()

{

list <char > l1 , l2;

int i;

for (i =0; i <10; i +=2)

l1 . push\_back ('A'+i);

for (i =1; i <11; i +=2)

l2 . push\_back ('A'+i);

cout << "Contents of list1: ";

list <char >:: iterator p = l1 . begin ();

while (p != l1 . end ()){

cout << \*p;

p ++;

}

cout << endl << endl ;

cout << "Contents of list2: ";

p = l2 . begin ();

while (p != l2 . end ()){

cout << \*p;

p ++;

}

cout << endl << endl ;

l1 . merge ( l2 );

if(l2 . empty ())

cout << "list2 is now empty \n";

cout << "Contents of list1 after merge :\n";

p = l1 . begin ();

while (p != l1 . end ()){

cout << \*p;

p ++;

}

return 0;

}

**Output:**

Contents of list1: ACEGI

Contents of list2: BDFHJ

list2 is now empty

Contents of list1 after merge :

ABCDEFGHIJ

Process returned 0 (0x0) execution time : 0.111 s

Press any key to continue.