OVERLOADING AND TEMPLATES

VGP130 – OOP II

Instructor: Ivaldo Tributino

FUNCTION OVERLOADING

Function's name is overloaded All of the functions in the set have the same name: all functions in the set have different signatures if they have different formal parameter lists.

Example: function that determines the larger of two items. Both items can be integers, floating-point numbers, characters, or strings. You could write several functions as follows:

```
int larger(int x, int y);
char larger(char first, char second);
double larger(double u, double v);
string larger(string first, string second);
```

If the call is larger(5, 3), for example, the first function is executed. If the call is larger('A', '9'), the second function is executed, and so on.

FUNCTION TEMPLATES

C++ simplifies the process of overloading functions by providing function templates.

The syntax of the function template is:

```
template <class Type>
function definition;
```

```
template <class Type>
Type larger(Type x, Type y)
{
    if (x >= y)
        return x;
    else
        return y;
}
```

Define a function template larger, which returns the larger of two items. In the function heading, the type of the formal parameters x and y is Type, which will be specified by the type of the actual parameters when the function is called.

DETERMINE THE LARGER OF THE TWO ITEMS.

main.cpp

main.cpp > main()

```
20
21  template <class Type>
22  Type larger(Type x, Type y);
23
24  int main() {
25
26  //***********************************************
27  //      ---- larger - template fanction ----
28  //***********************************************
29  cout << "---- larger - template fanction ----" << endl;
30
31
32  cout << "Line 1: Larger of 5 and 6 = " << larger(5, 6) << endl;
33  cout << "Line 2: Larger of A and B = " << larger('A', 'B') << endl;
34  cout << "Line 3: Larger of 5.6 and 3.2 = " << larger(5.6, 3.2) << endl;
35
36  string str1 = "Hello";
37  string str2 = "Happy";
38  cout << "Line 6: Larger of " << str1 << " and " << str2 << " = " << larger(str1, str2) << endl;
39
40  return 0;
41
42  }
43
44  template <class Type>
45  Type larger(Type x, Type y) {
46
47  if (x >= y)
48  return x; else
49  return y;
50  }
51
52  template<class Type>
```

Note that the function template larger will work only for those data types for which the operator >= has been defined.

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

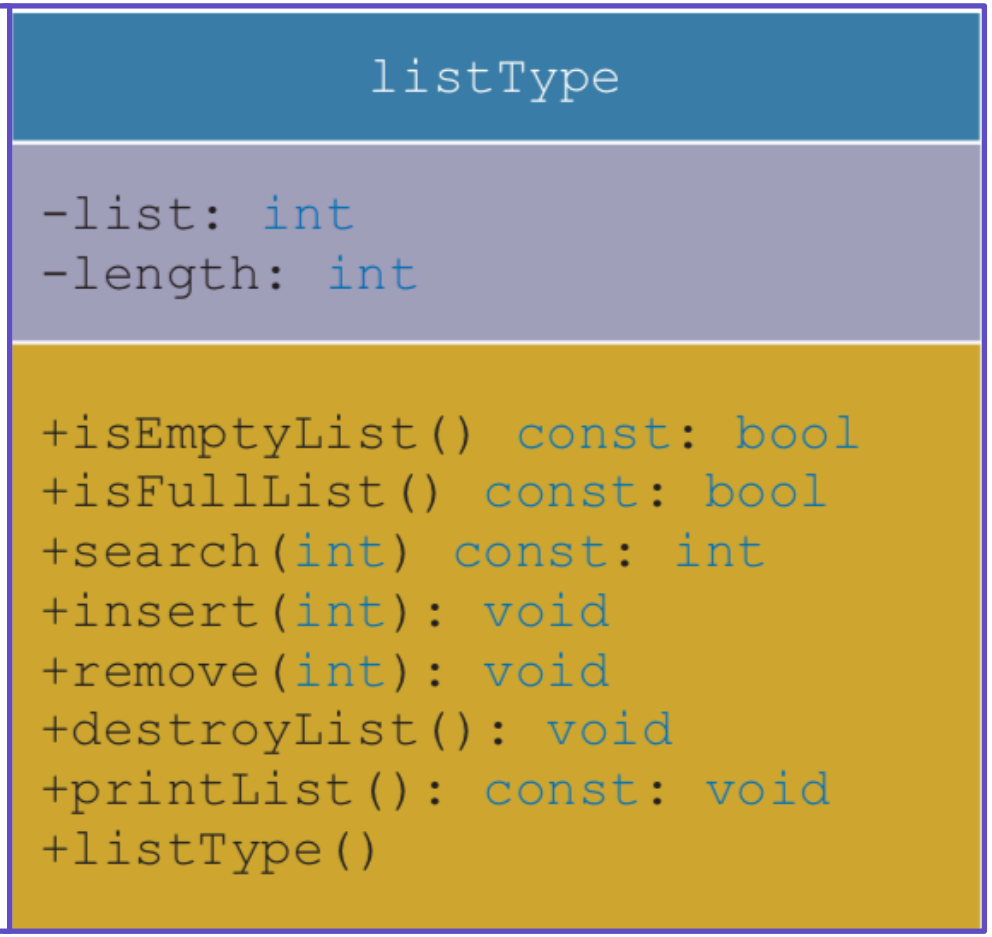
bash

```
---- larger - template fanction ----
Line 1: Larger of 5 and 6 = 6
Line 2: Larger of A and B = B
Line 3: Larger of 5.6 and 3.2 = 5.6
Line 6: Larger of Hello and Happy = Hello
(base) Ivaldo:template admin$
```

master*+ 0 0 Live Share Ln 39, Col 1 Spaces: 4 UTF-8 LF C++ Mac

CLASS LISTTYPE

```
class listType
{
public:
    bool isEmptyList() const;
    bool isFullList() const;
    int search(int searchItem) const;
    void insert(int newElement);
    void remove(int removeElement);
    void destroyList();
    void printList() const;
    listType(); //constructor

private:
    int list[1000];
    int length;
};
```



CLASS TEMPLATES

If the list element type changes from int to, say, char, double, or string, we need to write separate classes for each element type. However, for the most part, the operations on the list and the algorithms to implement those operations remain the same.

Using class templates, we can create a generic class listType, and the compiler can generate the appropriate source code for a specific implementation.

The syntax we use for a class template is::

```
template <class Type>
class declaration;
```

CLASS TEMPLATE

main.cpp ● listType.h ● Polygon.h

listType.h > isEmpty() const

```
18
19
20  template<class elemType>
21  class listType
22  {
23  public:
24      bool isEmpty() const; //determine whether the list is empty
25      bool isFull() const; // determine whether the list is full
26      int getLength() const; // return the number of elements in the list.
27      int getMaxSize() const; // return the maximum number of elements that can be stored in the list.
28      void sort(); // to sort the list
29      void print() const; //Outputs the elements of the list.
30      void insertAt(const elemType& item, int position); //Function to insert item in the list at the location
31      //specified by position.
32
33      /**
34       * Constructor
35       * Creates an array of the size specified by the parameter listSize
36       * the default array size is 50.
37       */
38      listType(int listSize = 50);
39
40      /**
41       * Destructor
42       * Deletes all the elements of the list.
43       * Postcondition: The array list is deleted.
44       */
45
46      ~listType();
47
48
49  private:
50      int maxSize; // variable to store the maximum size
51
52      int length; // variable to store the number of elements in the list
53
54      elemType *list; //pointer to the array that holds the list elements.
55
56  };
57
```

Because we can dynamically allocate arrays, the user will have the option to specify the size of the array. The default array size is 50. We will manipulate a list of integers.

master*+ ⊗ 0 ⚠ 0 Live Share Ln 58, Col 26 Spaces: 4 UTF-8 LF C++ Mac

Attention 1:

The function members of a class template are considered function templates. Thus, we must follow the definition of the function template. For example:

```
template <class elemType>
void listType<elemType> :: insert(elemType newElement)
{
. . .
}
listType<int> intList;
```

When the compiler generates the code for intList, it replaces the word elemType.

Attention 2:

HEADER FILE AND IMPLEMENTATION FILE OF A CLASS TEMPLATE

Passing parameters to a function has an effect at run time, whereas passing a parameter to a class template has an effect at compile time. Because the actual parameter to a class is specified in the user code and because the compiler cannot instantiate a function template without the actual parameter to the template, we can no longer compile the implementation file independently of the user code.
For this reason let's put the class definition and the function definitions in the same header file.

For other solutions see [6].

LISTTYPE.H

main.cpp ● listType.h ● Polygon.h

listType.h > ~listType()

```
58  template <class elemType>
59  bool listType<elemType>::isEmpty() const
60  {
61      return (length == 0);
62  }
63
64  template <class elemType>
65  bool listType<elemType>::isFull() const
66  {
67      return (length == maxSize);
68  }
69
70  template <class elemType>
71  int listType<elemType>::getLength() const
72  {
73      return length;
74  }
75
76  template <class elemType>
77  int listType<elemType>::getMaxSize() const
78  {
79      return maxSize;
80  }
81
82  //Constructor; the default array size is 50
83  template <class elemType>
84  listType<elemType>::listType(int listSize)
85  {
86      maxSize = listSize;
87      length = 0;
88      list = new elemType[maxSize];
89  }
90
91  template <class elemType>
92  listType<elemType>::~listType() //destructor
93  {
94      delete [] list;
95  }
96
```

master*+ ⊗ 0 ⚠ 0 Live Share Ln 95, Col 2 Spaces: 4 UTF-8 LF C++ Mac

HOW TO USE THE CLASS TEMPLATE LISTTYPE.

main.cpp listType.h Polygon.h

main.cpp > main()

```cpp
//*************************************************
//             ---- ListType ----
//*************************************************
cout <<     "---- ListType ----" << endl;

listType<int> intList(100);
int index, number;

cout << "List 5: Processing the integer list" << endl;
cout << "List 6: Enter 5 integers: ";

for(index=0; index<5; index++)
{
    cin >> number;
    intList.insertAt(number, index);
}

cout << endl;
cout << "List: ";
intList.print();
cout << endl;
intList.sort();
cout << "After sorting, intList: ";
intList.print();
cout << endl;

}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

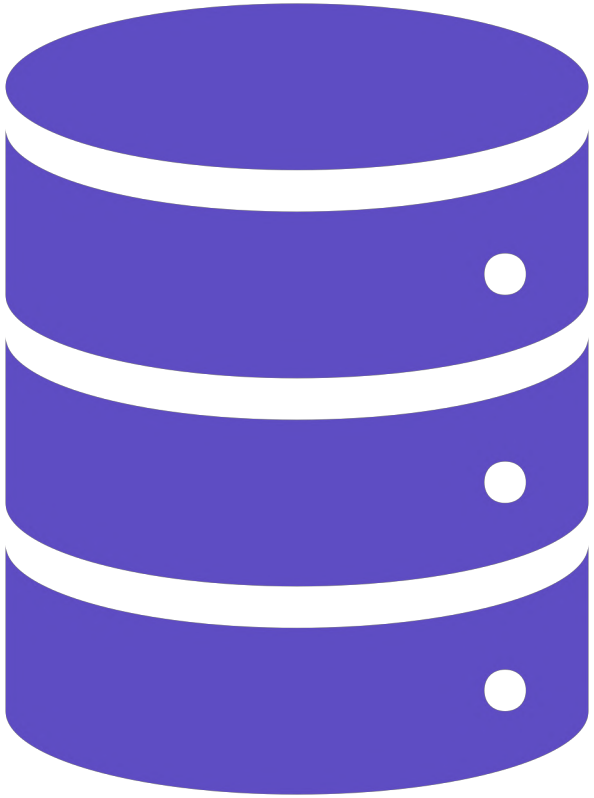
bash

```
---- ListType ----
List 5: Processing the integer list
List 6: Enter 5 integers: 5
7
8
9
0

List: 5 7 8 9 0

After sorting, intList: 0 5 7 8 9
```

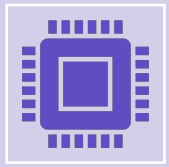
master*+ 0 0 Live Share Ln 67, Col 2 Spaces: 4 UTF-8 LF C++ Mac

STANDARD TEMPLATE LIBRARY

STL provides programmers to store the data efficiently and manipulate the stored data.



COMPONENTS OF THE STL



The main objective of a program is to manipulate data and generate results. This requires the ability to store data in computer memory, access a particular piece of data, and write algorithms to manipulate the data.

More formally, the STL has the following three main components:

Containers -- Iterators --Algorithms

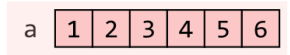
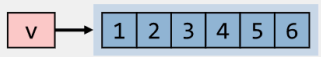
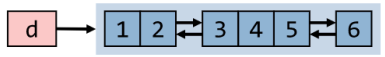
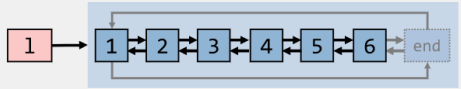
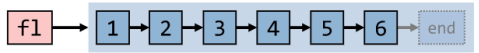
Containers and iterators are templatized classes. Iterators are used to step through the elements of a container. Algorithms are used to manipulate data.

CONTAINERS

A container is a holder object that stores a collection of other objects (its elements). They are implemented as class templates, which allows a great flexibility in the types supported as elements.

The container manages the storage space for its elements and provides member functions to access them, either directly or through iterators (reference objects with similar properties to pointers).

Sequence containers:	arrays, vector, deque, list, forward_list
Container adaptors:	queue, priority_queue, stack
Associative containers:	set, multiset, map, multimap
Unordered associative containers	unordered_set, unordered_multi, unordered_map, unordered_multi

`array<T, size>`	 fixed-size contiguous array
`vector<T>`	 dynamic contiguous array; amortized *O*(1) growth strategy; C++'s "default" container
`deque<T>`	 double-ended queue; fast insert/erase at both ends
`list<T>`	 doubly-linked list; *O*(1) insert, erase & splicing; in practice often slower than vector
`forward_list<T>`	 singly-linked list; *O*(1) insert, erase & splicing; needs less memory than `list`; in practice often slower than vector

source: https://hackingcpp.com/cpp/std/sequence_containers.html

SEQUENCE CONTAINERS

In C++, the sequence containers are a group of template classes to store data elements.

The container classes *array*, *vector*, and *deque* are implemented by using an array data structure. And the container classes, *list* and *forward_list*, are implemented using a linked list data structure.

SEQUENCE CONTAINER: VECTORS

1. Declaring Vector Objects
2. Declaring An Iterator To A Vector Container
3. Member Functions Common To All Containers
4. Member Functions Common To Sequence Containers
5. Copy Algorithm
6. Ostream Iterator And The Function Copy

SEQUENCE CONTAINER: VECTORS

The name of the header file containing the class vector is vector. Thus, to use a vector container in a program, the program must include the following statement:

```
#include <vector>
```

We must specify the type of the object because the class vector is a class template. For example, the statement:

```
vector<int> intList;
```

DECLARING VECTOR OBJECTS

The class vector contains several constructors, including the default constructor. Therefore, a vector container can be declared and initialized several ways. The table below shows ways to declare and initialize an vector.

Statement	*Effect*
vector<elemType> vecList;	Creates the empty vector container vecList.
vector<elemType> vecList(otherVecList);	Creates the vector container vecList, and initializes vecList to the elements of the vector otherVecList.
vector<elemType> vecList(size);	Creates the vector container vecList of size size. vecList is initialized using the default constructor.
vector<elemType> vecList(n, elm);	Creates the vector container vecList of size n. vecList is initialized using n copies of the element elm.
vector<elemType> vecList(beg, end);	Creates the vector container vecList in the range [beg, end). Both beg and end are pointers, called iterators in STL terminology. (Next week, we will see how iterators are used.)

INITIALIZE A VECTOR

Makefile | main.cpp

main.cpp > main()

```cpp
//*********************************************************
cout << "---- Initialize a vector in C++ ----" << endl;

cout << "Create an empty vector: " << endl;
vector<int> vect1;

for(int x : vect1){
    cout << x << ",";
}
cout << "\n";

cout << "Create a vector using the default constructor: " << endl;
vector<int> vect2(10);
for(int x : vect2){
    cout << x << ",";
}
cout << "\n";

cout << "Create a vector of size n with all values as 10: " << endl;
int n = 7;
vector<int> vect3(n, 10);
for (int x : vect3){
    cout << x << ",";
}
cout << "\n";

cout << "Create a vector from another vector: " << endl;
vector<int> vect4(vect3.begin()+2, vect3.end());
for (int x : vect4){
    cout << x << ",";
}
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
---- Initialize a vector in C++ ----
Create an empty vector:

Create a vector using the default constructor:
0,0,0,0,0,0,0,0,0,0,
Create a vector of size n with all values as 10:
10,10,10,10,10,10,10,
Create a vector from another vector:
10,10,10,10,10,
(base) Ivaldo:vector admin$
```

master*+ | 0 ⚠ 0 | Live Share | Ln 55, Col 18 | Spaces: 4 | UTF-8 | LF | C++ | Mac

MANIPULATE THE DATA STORED IN A VECTOR CONTAINER

Operations to Access the Elements of a Vector Container

Expression	*Description*
vecList[index]	Returns the element at the position specified by index.
vecList.front( )	Returns the first element. (Does not check whether the container is empty.)
vecList.back( )	Returns the last element. (Does not check whether the container is empty.)

OPERATIONS TO DETERMINE THE SIZE OF A VECTOR CONTAINER

Expression	*Description*
vecCont.capacity()	Returns the maximum number of elements that can be inserted into the container vecCont without reallocation.
vecCont.empty()	Returns true if the container vecCont is empty, false otherwise.
vecCont.size()	Returns the number of elements currently in the container vecCont.
vecCont.max_size()	Returns the maximum number of elements that can be inserted into the container vecCont.

VARIOUS OPERATIONS ON A VECTOR CONTAINER

Statement	*Effect*
vecList.insert(position, elem)	A copy of elem is inserted at the position specified by position. The position of the new element is returned.
vecList.insert(position, n, elem)	n copies of elem are inserted at the position specified by position.
vecList.insert(position, beg, end)	A copy of the elements, starting at beg until end-1, is inserted into vecList at the position specified by position.
vecList.push_back(elem)	A copy of elem is inserted into vecList at the end.
vecList.pop_back()	Deletes the last element.
vecList.resize(num)	Changes the number of elements to num. If size() increases, the default constructor creates the new elements.
vecList.resize(num, elem)	Changes the number of elements to num. If size() increases, the constructor creates the elements elem.

VECTOR

Makefile | main.cpp

main.cpp > main()

```cpp
//*************************************************
//  -------- MANIPULATE THE DATA --------
//*************************************************
cout << "---- MANIPULATE THE DATA ----" << endl;

vector<int> vecPrime(5);

for (int j = 2; j < 7; j++)
    vecPrime[j-2] = j;

vecPrime.insert(vecPrime.end()-1,7);

// To add elements to intPrimevecPrime, we can use the function push_back as follows:
vecPrime.push_back(11);
vecPrime.push_back(13);
vecPrime.push_back(17);
vecPrime.push_back(19);

vecPrime.resize(11, 23);

for (int x : vecPrime){
    cout << x << ",";
}
cout << "\n";
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
---- MANIPULATE THE DATA ----
2,3,4,5,7,6,11,13,17,19,23,
(base) Ivaldo:vector admin$
```

master*+ 0 0 Live Share Ln 97, Col 5 Spaces: 4 UTF-8 LF C++ Mac

FUNCTIONS COMMON TO SEQUENCE CONTAINERS

Expression	*Description*
seqCont.insert(position, elem)	A copy of elem is inserted at the position specified by the iterator position. The position of the new element is returned.
seqCont.insert(position, n, elem)	n copies of elem are inserted at the position specified by the iterator position.
seqCont.insert(position, beg, end)	A copy of the elements, starting at beg until end-1, are inserted into seqCont at the position specified by the iterator position. Also, beg and end are iterators.
seqCont.push_back(elem)	A copy of elem is inserted into seqCont at the end.
seqCont.pop_back()	Deletes the last element.
seqCont.erase(position)	Deletes the element at the position specified by the iterator position.
seqCont.erase(beg, end)	Deletes all the elements starting at beg until end-1. Both beg and end are iterators.
seqCont.clear()	Deletes all the elements from the container.

VECTOR

Makefile | main.cpp

main.cpp > main()

```
61      cout << "\n";
62      //************************************************
63      //  -------- MANIPULATE THE DATA --------
64      //************************************************
65      cout << "---- MANIPULATE THE DATA ----" << endl;
66
67      vector<int> vecPrime(5);
68
69      for (int j = 2; j < 7; j++)
70          vecPrime[j-2] = j;
71
72      vecPrime.insert(vecPrime.end()-1,7);
73
74      // To add elements to intPrimevecPrime, we can use the function push_back as follows:
75      vecPrime.push_back(11);
76      vecPrime.push_back(13);
77      vecPrime.push_back(17);
78      vecPrime.push_back(19);
79
80      vecPrime.resize(11, 23);
81
82      for (int x : vecPrime){
83          cout << x << ",";
84      }
85      cout << "\n";
86
87      cout << "Erasing 4 and 6 " << endl;
88      vecPrime.erase(vecPrime.begin()+2);
89      vecPrime.erase(vecPrime.begin()+4);
90      for (int x : vecPrime){
91          cout << x << ",";
92      }
93      cout << "\n";
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
---- MANIPULATE THE DATA ----
2,3,4,5,7,6,11,13,17,19,23,
Erasing 4 and 6
2,3,5,7,11,13,17,19,23,
(base) Ivaldo:vector admin$
```

master*+ 0 0 Live Share Ln 83, Col 27 Spaces: 4 UTF-8 LF C++ Mac

FUNCTIONS THAT ARE COMMON TO ALL CONTAINERS.

Member Function	*Description*
Default constructor	Initializes the object to an empty state.
Constructor with parameters	In addition to the default constructor, every container has constructors with parameters. We will describe these constructors when we discuss a specific container.
Copy constructor	Executes when an object is passed as a parameter by value, and when an object is declared and initialized using another object of the same type.
Destructor	Executes when the object goes out of scope.
ct.empty()	Returns true if container ct is empty, false otherwise.

Member Function	*Description*
ct.size()	Returns the number of elements currently in container ct.
ct.max_size()	Returns the maximum number of elements that can be inserted in container ct.
ct1.swap(ct2)	Swaps the elements of containers ct1 and ct2.
ct.begin()	Returns an iterator to the first element into container ct.
ct.end()	Returns an iterator to the position after the last element into container ct.
ct.rbegin()	Reverse begin. Returns a pointer to the last element into container ct. This function is used to process the elements of ct in reverse.
ct.rend()	Reverse end. Returns a pointer to the position before the first element into container ct.
ct.insert(position, elem)	Inserts elem into container ct at the position specified by position. Note that here position is an iterator.
ct.erase(beg, end)	Deletes all the elements between beg...end-1 from container ct. Both beg and end are iterators.
ct.clear()	Deletes all the elements from the container. After a call to this function, container ct is empty.
ct1 = ct2;	Copies the elements of ct2 into ct1. After this operation, the elements in both containers are the same.
ct1 == ct2	Returns true if containers ct1 and ct2 are equal, false otherwise.
ct1 != ct2	Returns true if containers ct1 and ct2 are not equal, false otherwise.

DECLARING AN ITERATOR TO A VECTOR CONTAINER

The class vector contains a typedef iterator, which is declared as a public member. An iterator to a vector container is declared using the typedef iterator. For example, the statement:

vector<int>::iterator intVecIter;

++intVecIter

advances the iterator intVecIter to the next element in the container, and the expression:

*intVecIter

returns the element at the current iterator position.

ITERATOR

Makefile | main.cpp

main.cpp > main()

```
83          cout << x << ",";
84      }
85      cout << "\n";
86
87      cout << "Erasing 4 and 6 " << endl;
88      vecPrime.erase(vecPrime.begin()+2);
89      vecPrime.erase(vecPrime.begin()+4);
90      for (int x : vecPrime){
91          cout << x << ",";
92      }
93      cout << "\n";
94
95      //************************************************
96      //  --------  Iterator --------
97      //************************************************
98      cout << "----  Iterator ----" << endl;
99
100     vector<int>::iterator vecPrimeIter;
101
102     cout <<  "Navigate through a vector using iterators" << endl;
103     for (vecPrimeIter = vecPrime.begin() ; vecPrimeIter != vecPrime.end(); ++vecPrimeIter)
104         cout << *vecPrimeIter << " ";
105     cout <<  '\n';
106
107
108
109
110
111
112
113
114
115
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
2,3,5,7,11,13,17,19,23,
----  Iterator ----
Navigate through a vector using iterators
2 3 5 7 11 13 17 19 23
(base) Ivaldo:vector admin$
```

master*+ 0 0 Live Share Ln 121, Col 1 Spaces: 4 UTF-8 LF C++ Mac

COPY ALGORITHM

In general, the function copy allows us to copy the elements from one place to another. For example, to output the elements of a vector, or to copy the elements of a vector into another vector, we can use the function copy. The prototype of the function template copy is:

```
template <class inputIterator, class outputIterator>
outputItr copy(inputIterator first1, inputIterator last, outputIterator first2);
```

- first1 - specifies the position from where to begin copying the elements;
- Last - specifies the end position;
- first2 - specifies where to copy the elements.

To use the function copy, the program must include the
statement:

```
#include <algorithm>
```

COPY ALGORITHM.

Makefile | main.cpp | libraries.h

main.cpp > main()

```
91      cout << "----  Iterator ----" << endl;
92
93      vector<int>::iterator vecPrimeIter;
94
95      cout <<  "Navigate through a vector using iterators" << endl;
96      for (vecPrimeIter = vecPrime.begin() ; vecPrimeIter != vecPrime.end(); ++vecPrimeIter)
97          cout << *vecPrimeIter << " ";
98      cout <<  '\n';
99
100
101     //***********************************************
102     //  --------  Using copy function --------
103     //***********************************************
104     cout <<  "--------  Using copy function --------" << endl;
105     int primeArray [] = {2,3,5,7,11,13,17,19,23,29};
106     vecPrime.resize(10);
107
108     /*Recall that the array name, intArray, is, in fact, a pointer and contains the base address of the array.
109     Therefore, intArray points to the first component of the array, intArray + 1
110     points to the second component of the array, and so on.*/
111
112     copy(primeArray, primeArray + 10, vecPrime.begin());
113
114     for (vecPrimeIter = vecPrime.begin(); vecPrimeIter != vecPrime.end(); vecPrimeIter++)
115         cout << *vecPrimeIter << " ";
116     cout << endl;
117
118     // Now consider the statement:
119     copy(vecPrime.rbegin() + 2, vecPrime.rend(), vecPrime.rbegin());
120
121     for (vecPrimeIter = vecPrime.begin(); vecPrimeIter != vecPrime.end(); vecPrimeIter++)
122         cout << *vecPrimeIter << " ";
123     cout << endl;
124
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
2 3 5 7 11 13 17 19 23
--------  Using copy function --------
2 3 5 7 11 13 17 19 23 29
2 3 2 3 5 7 11 13 17 19
(base) Ivaldo:vector admin$
```

master*+ | 0 0 | Live Share | Ln 99, Col 1 | Spaces: 4 | UTF-8 | LF | C++ | Mac

OSTREAM ITERATORS

The function copy can also be used to output the elements of a container. In this case, an iterator of type ostream specifies the destination.

When we create an iterator of type ostream, we also specify the type of element the iterator will output.

The following statement illustrates how to create an ostream iterator of type int:

```
ostream_iterator<int> screen(cout, " ");
```

This statement creates screen to be an ostream iterator, and the element type is int. The iterator screen has two arguments: cout and space.

- The iterator screen is initialized using the object cout;
- Iterator outputs the elements, they are separated by a space.

COPY ALGORITHM & OSTREAM_ITERATOR

Makefile | main.cpp | libraries.h

main.cpp > main()

```
118     // Now consider the statement:
119     copy(vecPrime.rbegin() + 2, vecPrime.rend(), vecPrime.rbegin());
120
121     for (vecPrimeIter = vecPrime.begin(); vecPrimeIter != vecPrime.end(); vecPrimeIter++)
122         cout << *vecPrimeIter << " ";
123     cout << '\n';
124
125     //************************************************
126     //  -- Now using ostream_iterator & copy function --
127     //************************************************
128     cout <<  "-- Now using ostream_iterator & copy function --" << endl;
129
130     // ostream_iterator
131     std::ostream_iterator<int> screen(cout, " ");
132
133     copy(primeArray, primeArray + 10, vecPrime.begin());
134     copy(vecPrime.rbegin() + 2, vecPrime.rend(), vecPrime.rbegin());
135     copy(vecPrime.begin(), vecPrime.end(), screen);
136
137
138
139
140
141
142
143
144
145
146
147
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
2,3,5,7,11,13,17,19,23,
----  Iterator ----
Navigate through a vector using iterators
2 3 5 7 11 13 17 19 23
--------  Using copy function --------
2 3 5 7 11 13 17 19 23 29
2 3 2 3 5 7 11 13 17 19
-- Now using ostream_iterator & copy function --
(base) Ivaldo:vector admin$
```

master*+ | 0 ⚠ 0 | Live Share | Ln 147, Col 1 | Spaces: 4 | UTF-8 | LF | C++ | Mac

COPY ALGORITHM & OSTREAM_ITERATOR

Makefile | main.cpp | libraries.h

main.cpp > main()

```
137
138     cout << "-- Now let's create a vector from a text file and print its elements. --" << endl;
139
140     ifstream fileIn("francis.txt");
141
142     if(!fileIn.is_open())
143     {
144         cout << "Failed to open file!\n";
145         return 0;
146     }
147
148     string data;
149
150     // istream_iterator< string > is(fileIn);
151     // istream_iterator< string > eof;
152
153     vector< string > text;
154
155     while (fileIn >> data) // loop until no more input or error
156     {
157         // and remember operator>> can only extract string without spaces!
158         text.push_back(data);
159     }
160
161     // copy( is, eof, back_inserter(text));
162     std::ostream_iterator<string> os(cout, " ");
163     copy( text.begin(), text.end(), os);
164
165
166
167
168     return 0;
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
2 3 2 3 5 7 11 13 17 19
-- Now using ostream_iterator & copy function --
2 3 2 3 5 7 11 13 17 19
-- Now let's create a vector from a text file and print its elements. --
Lord make Me an instrument of Your peace Where there is hatred let me sow love. Where there is injury, pardon. Where there is doubt, faith. Where there is despair, hope.
Where there is darkness, light. Where there is sadness joy. O Divine master grant that I may. Not so much seek to be consoled as to console. To be understood, as to under
stand. To be loved. as to love For it's in giving that we receive And it's in pardoning that we are pardoned And it's in dying that we are born To eternal life. Aaamen\n
} (base) Ivaldo:vector admin$
```

master*+ | 0 ⚠ 0 | Live Share | Ln 138, Col 86 | Spaces: 4 | UTF-8 | LF | C++ | Mac

References:

1) Cplusplus (2022/01/07) Containers, https://www.cplusplus.com/reference/stl
2) Geeksforgeeks (2022/01/06) The C++ Standard Template Library, https://www.geeksforgeeks.org/the-c-standard-template-library-stl/
3) hacking C++(2022/01/07), https://hackingcpp.com/
4) C++ Programming From Problem Analysis to Program Design - D. S. Malik - 2011
5) C++ benchmark – std::vector VS std::list VS std::deque (20/22/01/07) https://baptiste-wicht.com/posts/2012/12/cpp-benchmark-vector-list-deque.html
6) Codeguru (2022/01/07) An Introduction to Sequence Containers in C++ , https://www.codeguru.com/cplusplus/an-introduction-to-sequence-containers-in-c/

Appendix

- Libraries.h
- listType.h
- main.cpp