Khoi Duong

Prof. Yang

CS360L

8/23/2022

LAB#12

1.

Array< Employee > workerList(100);

The C++ program will initialize an array of objects belonging to the class Employee (Employee object). The array of objects is named "workerList". The program will pass 100 as the argument to tell the constructor to limit the number of objects in the array to 100.

2.

template< typename T > Array< T >::Array(int s)

The C++ program will create a function template in order to create a generic function that can be used for different data types. And the generic function will be applied for the array of object. It will create an array of object of different data types. In this case, the program will create an array of integer data type.

3.

Source code:

Array.h (https://github.com/MynameisKoi/CS360/blob/main/Lab12_3.h)

```
// Array.h
// Array class definition with overloaded operators.
#ifndef ARRAY_H
#define ARRAY_H
#include <iostream>
template <class T> class Array{
  template <class U> friend std::ostream &operator<<( std::ostream &, const Array<U>
& );
  template <class U> friend std::istream &operator>>( std::istream &, Array<U> & );

  public:
    explicit Array( int = 10 ); // default constructor
    size_t getSize() const; // return size
    Array( const Array & ); // copy constructor
    ~Array(); // destructor
    const Array &operator=( const Array & ); // assignment operator
    bool operator==( const Array & ) const; // equality operator
    // inequality operator; returns opposite of == operator
    bool operator!=( const Array &right ) const{
      return ! ( *this == right ); // invokes Array::operator==
    } // end function operator!=
    // subscript operator for non-const objects returns modifiable lvalue
    T &operator[]( int );
    // subscript operator for const objects returns rvalue
    T operator[]( int ) const;

    size_t size; // pointer-based array size
  private:
    T *ptr; // pointer to first element of pointer-based array
}; // end class Array
#endif
```

Array.cpp (https://github.com/MynameisKoi/CS360/blob/main/Lab12_3.cpp)

```
// Array.cpp
// Array class member- and friend-function definitions.
#include <iostream>
#include <iomanip>
#include <stdexcept>
#include "Array.h" // Array class definition
using namespace std;
// default constructor for class Array (default size 10)
template <class T> Array<T>::Array( int arraySize ): size( arraySize > 0 ? arraySize
:
  throw invalid_argument( "Array size must be greater than 0" ) ), ptr( new T[ size
] )
{
  for ( size_t i = 0; i < size; ++i )
    ptr[ i ] = i; // set pointer-based array element
} // end Array default constructor
// copy constructor for class Array;
// must receive a reference to an Array
template  <class  T>  Array<T>::Array(  const  Array  &arrayToCopy  ):  size(
arrayToCopy.size ),ptr( new T[ size ] )
{
  for ( size_t i = 0; i < size; ++i )
    ptr[ i ] = arrayToCopy.ptr[ i ]; // copy into object
} // end Array copy constructor
// destructor for class Array
template <class T> Array<T>::~Array(){
  delete [] ptr; // release pointer-based array space
} // end destructor
 // return number of elements of Array
template <class T> size_t Array<T>::getSize() const{
 return size; // number of elements in Array
} // end function getSize
 // overloaded assignment operator;
template <class T> const Array<T> &Array<T>::operator=( const Array &right )
{
  if ( &right != this )
  {
    // for Arrays of different sizes, deallocate original
    // left-side Array, then allocate new left-side Array
    if ( size != right.size )
    {
      delete [] ptr; // release space
      size = right.size; // resize this object
      ptr = new T[ size ]; // create space for Array copy
    } // end inner if
    for ( size_t i = 0; i < size; ++i )
      ptr[ i ] = right.ptr[ i ]; // copy array into object
  } // end outer if
  return *this; // enables x = y = z, for example
} // end function operator=
 // determine if two Arrays are equal and
 // return true, otherwise return false
template <class T> bool Array<T>::operator==( const Array &right ) const
{
  if ( size != right.size )
```

```
    return false; // arrays of different number of elements
  for ( size_t i = 0; i < size; ++i )
    if ( ptr[ i ] != right.ptr[ i ] )
      return false; // Array contents are not equal
  return true; // Arrays are equal
} // end function operator==
 // overloaded subscript operator for non-const Arrays;
 // reference return creates a modifiable lvalue
template <class T> T &Array<T>::operator[]( int subscript )
{
  // check for subscript out-of-range error
  if ( subscript < 0 || subscript >= size )
    throw out_of_range( "Subscript out of range" );
  return ptr[ subscript ]; // reference return
} // end function operator[]
// overloaded subscript operator for const Arrays
// const reference return creates an rvalue
template <class T> T Array<T>::operator[]( int subscript ) const
{
  // check for subscript out-of-range error
  if ( subscript < 0 || subscript >= size )
    throw out_of_range( "Subscript out of range" );
  return ptr[ subscript ]; // returns copy of this element
} // end function operator[]
 // overloaded input operator for class Array;
 // inputs values for entire Array
template <class U>
istream &operator>>( istream &input, Array<U> &a )
{
  for ( size_t i = 0; i < a.size; ++i )
    input >> a.ptr[ i ];
  return input; // enables cin >> x >> y;
} // end function
 // overloaded output operator for class Array
template <class U>
ostream &operator<< ( ostream &output, const Array<U> &a )
{
  // output private ptr-based array
  for ( size_t i = 0; i < a.size; ++i )
  {
    output << setw( 12 ) << a.ptr[ i ];
    if ( ( i + 1 ) % 4 == 0 ) // 4 numbers per row of output
      output << endl;
  } // end for
  if ( a.size % 4 != 0 ) // end last line of output
```

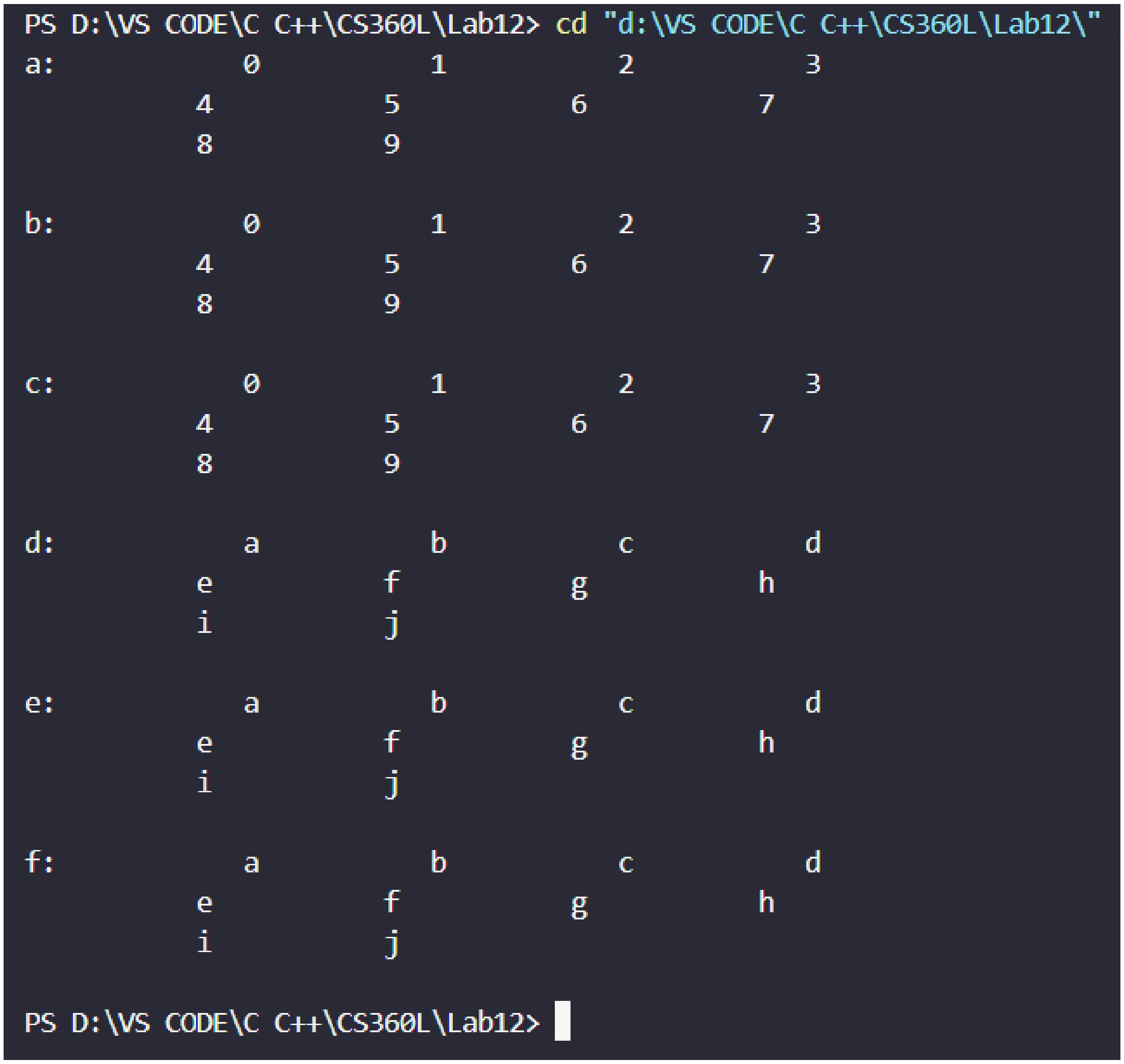
```
    output << endl;
  return output; // enables cout << x << y;
} // end function operator<<

int main(){
  Array<int> a( 10 ); // create Array of 10 ints
  Array<int> b( a ); // create another Array of 10 ints
  Array<int> c( 20 ); // create Array of 20 ints
  c = a; // copy Array a into Array c
  cout << "a: " << a << endl; // output Array a
  cout << "b: " << b << endl; // output Array b
  cout << "c: " << c << endl; // output Array c

  Array<char> d(10); // create Array of 10 chars
  for (size_t j=0; j<d.size; j++) //insert elements
  {
    // create an array of type char
    d[j] = 'a' + j;
  }
  Array<char> e(d); // create another Array of 10 chars
  Array<char> f(20); // create Array of 20 chars
  f = d; // copy Array d into Array f
  cout << "d: " << d << endl; // output Array d
  cout << "e: " << e << endl; // output Array e
  cout << "f: " << f << endl; // output Array f

  return 0;
} // end main
```

Run program & result:



```
PS D:\VS CODE\C C++\CS360L\Lab12> cd "d:\VS CODE\C C++\CS360L\Lab12\"
a:         0           1           2           3
         4           5           6           7
         8           9

b:         0           1           2           3
         4           5           6           7
         8           9

c:         0           1           2           3
         4           5           6           7
         8           9

d:         a           b           c           d
         e           f           g           h
         i           j

e:         a           b           c           d
         e           f           g           h
         i           j

f:         a           b           c           d
         e           f           g           h
         i           j

PS D:\VS CODE\C C++\CS360L\Lab12>
```