**Logo

Description automatically generated San Francisco Bay University**

**CS360 - Programming in C and C++**

**Homework Assignment #4**

**Due day: 07/19/2022**

**Instruction:**

1. **Push the answer sheets/source code to Github**
2. **Please follow the code style rule like programs on handout.**
3. **Overdue homework assignment submission can’t be accepted.**

**4. Take academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**

1. One nice example of overloading the function call operator () is to allow another form of double-array subscripting popular in some programming languages. Instead of saying

*chessBoard[ row ][ column ]*

for an array of objects, overload the function call operator to allow the alternate form

*chessBoard( row, column )*

Create a class *DoubleSubscriptedArray* that has similar features to class *Array* as following example programs. At construction time, the class should be able to create a *DoubleSubscriptedArray* of any number of rows and columns. The class should supply *operator()* to perform double-subscripting operations. For example, in a *3-by-5* *DoubleSubscriptedArray* called *chessBoard*, the user could write *chessBoard(1, 3)* to access the element at row *1* and column *3*. Remember that operator() can receive *any* number of arguments. The underlying representation of the *DoubleSubscriptedArray* could be a one-dimensional array of integers with *rows \* columns* number of elements. Function *operator()* should perform the proper pointer arithmetic to access each element of the underlying array. There should be two versions of *operator()* - one that returns *int &* (so that an element of a *DoubleSubscriptedArray* can be used as an *lvalue*) and one that returns *int*. The class should also provide the following operators: *==, !=, =, <<* (for outputting the *DoubleSubscriptedArray* in row and column format) and *>>* (for inputting the entire *DoubleSubscriptedArray* contents).

*//Array.h*

*// Array class definition with overloaded operators.*

*#ifndef ARRAY\_H*

*#define ARRAY\_H*

*#include <iostream>*

*class Array{*

*friend std::ostream &operator<<( std::ostream &, const Array & );*

*friend std::istream &operator>>( std::istream &, Array & );*

*public:*

*explicit Array( int = 10 ); // default constructor*

*Array( const Array & ); // copy constructor*

*~Array(); // destructor*

*size\_t getSize() const; // return size*

*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*

*int &operator[]( int );*

*// subscript operator for const objects returns rvalue*

*int operator[]( int ) const;*

*private:*

*size\_t size; // pointer-based array size*

*int \*ptr; // pointer to first element of pointer-based array*

*}; // end class Array*

*#endif*

*//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)*

*Array::Array( int arraySize ): size( arraySize > 0 ? arraySize :*

*throw invalid\_argument( "Array size must be greater than 0" ) ),*

*ptr( new int[ size ] )*

*{*

*for ( size\_t i = 0; i < size; ++i )*

*ptr[ i ] = 0; // set pointer-based array element*

*} // end Array default constructor*

*// copy constructor for class Array;*

*// must receive a reference to an Array*

*Array::Array( const Array &arrayToCopy ): size( arrayToCopy.size ),*

*ptr( new int[ 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*

*Array::~Array(){*

*delete [] ptr; // release pointer-based array space*

*} // end destructor*

*// return number of elements of Array*

*size\_t Array::getSize() const{*

*return size; // number of elements in Array*

*} // end function getSize*

*// overloaded assignment operator;*

*// const return avoids: ( a1 = a2 ) = a3*

*const Array &Array::operator=( const Array &right ){*

*if ( &right != this )// avoid self-assignment*

*{*

*// 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 int[ 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*

*bool Array::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*

*int &Array::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*

*int Array::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*

*istream &operator>>( istream &input, Array &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*

*ostream &operator<<( ostream &output, const Array &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<<*

1. Develop class *Polynomial*. The internal representation of a *Polynomial* is an array of terms. Each term contains a coefficient and an exponent, e.g., the term has the coefficient *2* and the exponent *4*. Develop a complete class containing proper constructor and destructor functions as well as set and get functions. The class should also provide the following overloaded operator capabilities:
   1. Overload the addition operator *(+)* to add two *Polynomials.*
   2. Overload the subtraction operator *(-)* to subtract two *Polynomials*.
   3. Overload the assignment operator to assign one *Polynomial* to another.
   4. Overload the multiplication operator *(\*)* to multiply two *Polynomials*.
   5. Overload the addition assignment operator *(+=)*, subtraction assignment operator *(-=)*, and multiplication assignment operator *(\*=)*.