 Match each of the vocabulary words at the right with the BEST definition on the left.

* A contiguous block of memory used to hold a set of data of the same type →
  + Array,
* These characters are used when defining an array to enclose its intended size. →
  + [ ],
* A named memory location used to store data that does not change during program execution →
  + constant,
* This looping construct is optimal for working with arrays →
  + for,
* An example of a simple selection statement →
  + if,
* Can be used to store a set of data that is "tabular" in nature →
  + 2 D array,
* Starting Index in an Array →
  + Zero,
* You use this search when looking for something in an ordered array →
  + binary search,
* This sort is good when adding items to a nearly sorted array →
  + Insertion,
* This sort is good when you are looking for the three smallest items, for example →
  + Selection

The Answer search repeatedly divides portion of the array being searched in half when it does not find a value.

* The maximum number of comparisons performed by linear search to find an item in an array of N elements is:
  + N
* How many numbers are placed in order on each pass through the data (each iteration of the inner loop) for the selection sort? (Enter a numeral.)
  + 1
* When storing values in an array, C++ does not allow you to access locations outside the array.
  + False

#include <iostream>  
#include <iomanip>

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

**const** **int** SIZE = 5;

// fill table with multiplication table 1-5  
**void** filler(**int** table[][SIZE])  
{  
    **for** (**int** i = 0; i < SIZE; i++)  
**for**(**int** j = 0; j < SIZE; j++)  
           table[i][j] = (i+1)\*(j+1);  
}

// display table, one row per line  
**void** display(**int** table[][SIZE])  
{  
**for** (**int** i = 0; i < SIZE; i++)  
     {  
**for**(**int** j = 0; j < SIZE; j++)  
        {  
            cout << setw(5) << table[i][j];  
        }  
        cout << endl;  
    }  
        cout << endl;  
}

// define table, fill it, display it  
**int** main()  
{  
**int** table[SIZE][SIZE];

    filler(table);

    display(table);

**return** 0;  
}

Review 2

Test 1

* The de-reference operator
  + \*
* The address of operator
  + &
* The keyword you use to allocate [dynamic memory](https://classes.lanecc.edu/mod/resource/view.php?id=2466732) on the heap
  + New
* The keyword you use to free up [dynamic memory](https://classes.lanecc.edu/mod/resource/view.php?id=2466732) when you are done with it
  + Delete
* These are used to delimit the members of a struct
  + {}
* This is used when you want to free up a dynamically allocated array
  + Delete[]
* This terminates a recursive function
  + Base case
* This operator is used to access the members of a struct
  + .(dot operator)
* The area of a program in which a variable is known and can be used
  + Scope
* These are used to inclose the index when you are accessing an array
  + []
* 1) Need to check base case first (did we reach end of array)
* 2) Base case should be >= length, not > length
* 3) Need to return result of recursive call
* 1) Need to check base case first (did we reach end of array)

#include <iostream>

using namespace std;

struct Student {  
    string name;  
    int grade;  
}; // need semicolon

int main()  
{  
    //struct Student mary;  
    Student mary; // do not need struct keyword

     //Mary:name = "Mary";  
     //Mary:grade = 100;

     mary.name = "Mary"; // variable is mary, not Mary and use dot not colon  
     mary.grade = 100;

     //cout << mary:name  " got a " << mary:grade << endl;

     cout << mary.name << " got a " << mary.grade << endl; // missing << and use dot not colon

    return 0;

}

#include <iostream>

// global constant  
**const** **int** SIZE = 10;

// prototypes  
**int**\* createArray();  
**void** displayArray(**const** **int**\* array);

// main function  
**int** main()   
{

**int**\* array = createArray();

    displayArray(array);

**delete**[] array;

**return** 0;  
}

// create array  
**int**\* createArray()   
{

**int** \*oddArray = **new** **int**[SIZE];

**int** number = 1;

**for** (**int** i = 0; i < SIZE; i++)   
   {

        \*(oddArray + i) = number;

        number += 2;  
    }

**return** oddArray;  
}

// display array  
**void** displayArray(**const** **int**\* array)   
{

**for** (**int** i = 0; i < SIZE; i++)

       std::cout << \*(array + i) << "\n";  
}

Review Test #3

Test 1

The correct answer is:

* The operator used to access methods or public variables of an object → .
  + (dot),
* An instance of a class →
  + object,
* A method that returns the value of an attribute of a class →
  + accessors or getters,
* A constructor that has no parameters. Every class must have one of these →
  + default constructor,
* Every object has a pointer to itself. What is it? → this,
* A keyword that allows you to pass an object (or anything else) by reference, without having to worry if the called function will change the object. →
  + const,
* A constructor with a single parameter that is a reference to an object of that class type →
  + copy constructor,
* A keyword that is used to indicate that a variable or method belongs to the class as a whole, rather than to an individual object →
  + static,
* The default access specifier for variables and methods in a class →
  + private,
* The operator that is used to access methods of an object that is created on the heap using new. →
  + ->,
* This method is used to delete any memory that was allocated in the object. →
  + destructor,
* This keyword is used when you want to allow an outside method or class access the private elements of a given class →
  + friend,
* The method of copying an object that copies what a pointer is pointing to rather than the address that it contains. →
  + deep copy,
* These methods are used to assign values to object variables →
  + mutators or setters

**class** Rainbow{

**private**:  
        **int** numArcs;  
        **bool** isComplete;

**public**:  
        // need default constructor  
       Rainbow() {}

        // constructor must match class name  
        // constructor must be defined, not just declared  
        //rainbow(int arcs, bool complete) : numArcs(arcs), isComplete(complete);  
        Rainbow(**int** arcs, **bool** complete) : numArcs(arcs), isComplete(complete) {}

        // differentiate between class variable and parameter  
        // setters do not return anything, should be void  
        //int setArcs(int numArcs) { numArcs = numArcs; }  
       **void** setArcs(**int** numArcs) { **this**->numArcs = numArcs; }

**int** getArcs() { **return** numArcs; }

**void** setFull(**bool** complete) { isComplete = complete; }

**bool** getFull() { **return** isComplete; }

}; // need semicolon at end of class

#include <iostream>

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

**class** Rug{  
    **private**:  
        **int** length;  
        **int** width;

**public**:  
        // this is a pointer  
        Rug(**int** length = 0, **int** width = 0)  
        {  
            //this.length = length;  
            //this.width = width;  
            **this**->length = length;  
            **this**->width = width;  
        }

**int** getLength() { **return** length; }

**int** getWidth() { **return** width; }

        // area should be length \* width  
        **int** getArea() { **return** length \* width; }

}; // need semicolon at end of class

**int** main()  
{  
    // or you could get rid of pointer and new and not have dereference problem  
    // Rug carpet(10,12);

    Rug \* carpet = **new** Rug(10,12);

    // use object name, not class name  
    // need to dereference or use ->  
    //cout << "The rug area is " << Rug.getArea() << endl;  
    cout << "The rug area is " << (\*carpet).getArea() << endl;

    // need to dereference or use ->  
    //cout << "The length is " << carpet.getLength() << endl;  
    cout << "The length is " << carpet->getLength() << endl;

    // need to dereference or use ->  
    //cout << "The width is " << carpet.getWidth() << endl;  
    cout << "The width is " << carpet->getWidth() << endl;

    // delete the carpet space  
    **delete** [] carpet;

**return** 0;

}

**class** Rectangle{  
    **private**:  
        **int** length;  
       **int** width;

**public**:  
        // constructor with default values  
       Rectangle(**int** length = 0, **int** width = 0) : length(length), width(width) {}

**void** setLength(**int** length) { **this**->length = length; }  
        **void** setWidth(**int** width) { **this**->width = width; }

**int** getLength() { **return** length; }  
        **int** getWidth() { **return** width; }

**int** getArea()  
       {  
           **if** (width == 0)  
                 **return** length \* length;  
            **else**                  **return** length \* width;  
       }  
};

// alternate approach  
**class** RectangleV2{  
    **private**:  
        **int** length;  
        **int** width;

**public**:

        // default constructor  
        RectangleV2() : length(0), width(0) {}  
  
        // square constructor  
        RectangleV2(**int** side) : length(side), width(side) {}  
  
        // rectangle constructor  
        RectangleV2(**int** length, **int** width) : length(length), width(width) {}

        // setters  
        **void** setLength(**int** length) { **this**->length = length; }  
        **void** setWidth(**int** width) { **this**->width = width; }

        // getters  
        **int** getLength() { **return** length; }  
        **int** getWidth() { **return** width; }

**int** getArea() { **return** length \* width; }  
};