1. B, C, E, F, H, I
2. Cannot convert a std::string to a ‘const char\*’ with both functions

3.  
\*p++; the values in array stay the same(1,2,3) \*p becomes address of index 1

(\*p)++; index 0 increments, everything else stays the same(2,2,3)

\*p++; (\*p)++; pointer becomes address of index 1 and increments (1,3,3)

//-----THIS IS MY HEADER FOR ALL MY CODE

#include <iostream>

#include <string.h>

#include <stdio.h>

#include <array>

using namespace std;

4.a

void sumArray(int intArray[], int arraySize){

int \*p = intArray;  
int sum = 0;

for(int i=0; i < arraySize; i++){

sum += \*p;

p++;

}

cout << sum;

delete p;

}  
//Main  
int main(){  
int SIZE = 9;  
int intArray[SIZE] = {1, 2,3,4,5,6,7,8,9};  
sumArray(intArray, SIZE);  
}

4.b

void removeOdds(int intArray[], int arraySize){

int \*p = intArray;

int counterForNewArray = 0;

//turns value of odds to zero

for(int i=0; i < arraySize; i++){

if(\*p %2 !=0){

\*p = 0;

//\*(p+1) = 0;

counterForNewArray++;

}

cout << \*p << endl;

p++;

}

counterForNewArray--;

//create new array to store evens and remove zeros

int newArray[counterForNewArray];

int \*p2 = newArray;

p = intArray;

for(int i=0; i < arraySize; i++){

if(\*p !=0){

\*p2 = \*p;

p2++;

}

//cout << \*p2 << endl;

p++;

}

//displays content of new array

p2 = newArray;

for(int i=0; i < counterForNewArray; i++){

cout << \*p2 << endl;

p2++;

}

//return pointer for new array

}

//MAIN  
int main(){

int SIZE = 9;

int intArray[SIZE] = {1, 2,3,4,5,6,7,8,9};

removeOdds(intArray, SIZE);

}

Would it be easier to write this function if the array was unordered? No because you would have to create a sort function that swaps values and returns the ordered list before displaying it.

6. if(p==q) compares the values of the pointers, which is a memory address.  
if(\*p==\*q) compares the values of the variable the pointers are pointing too.

Int main(){

int i = 7;

int j = 7;

int \*p = &i;

int \*q = &j;

if(p==q) //compares the values of the pointers, which is a memory address.

cout << " compares the values of the pointers, which is a memory address";

if(\*p==\*q) //compares the values of the variable the pointers are pointing too.

cout << " compares the values of the variable the pointers are pointing too.";

}

8. Private - only functions within the class can access the variable. For example: get & set functions,  
to retrieve the value or assign a new one you must use these functions. You can not access the variable from outside the class directly.

Protected – can be accessed from objects of the same class or derived classes. For my example I created a function that passes in an object of the same class.

class BaseClass{

public:

void passBaseClass(BaseClass &joint) {

joint.level = 77;

cout << joint.level << endl << this->level << endl;

}

protected:

int level = 55;

};

int main(){

BaseClass b, passMe;

b.passBaseClass(passMe);

}

//Output:  
77   
55

Public – The variable of an object can be accessed anywhere in the main program where the object is in scope.

Int main(){

BaseClass b;  
b.length = 99;

}