Name(s):

(15 points) Complete this exercise in pairs. The purpose of this exercise is to work with pointers and dynamic memory allocation. Recall that there are three distinct entities to be aware of when using pointers: the pointer variable (that stores a pointer), the pointer value (an address), and the location being pointed to (the "pointee", which may either be a dynamically allocated anonymous variable or a statically allocated named variable).

For each of the following programs, draw the memory picture created by each program at the points indicated in the comments (under the comment) and predict the output. Then create a project and type in the source code, and build and run the program. (Please do type in the code. You will remember more than if you copy and paste the code into the project.) Observe the actual output and explain any discrepancies between your prediction and the actual output. When you have completed this exercise, hand in these answer sheets.

**Program pointer1.cpp:**

```cpp
#include <iostream>
using namespace std;

int main ()
{
    int *ptr1, *ptr2;
    // (a.) what is the picture here

    ptr1 = new int;
    *ptr1 = 20;
    // (b.) what is the picture here

    cout << (*ptr1) << endl;
    ptr2 = ptr1;
    *ptr2 = *ptr2 + 5;
    // (c.) what is the picture here

    cout << (*ptr2) << endl;
}
```

**Memory pictures:**

- **(a.)** what is the picture here
- **(b.)** what is the picture here
- **(c.)** what is the picture here
cout << (*ptr1) << "  "
    << (*ptr2) << endl;
ptr1 = new int;
*ptr1 = *ptr2;                     // (d.) what is the picture here

cout << (*ptr1) << "  "
    << (*ptr2) << endl;
*ptr2 = 20;                        // (e.) what is the picture here

cout << (*ptr1) << "  "
    << (*ptr2) << endl;
delete ptr1;
delete ptr2;                       // (f.) what is the picture here
return 0;
}  // end main

Predicted output:    Actual output:

Explain any discrepancies:

Pointer variables and values may be used as formal parameters and actual arguments of functions and returned objects, just like any other data object. They may be value parameters or reference parameters. The rules for correspondence are exactly the same.

Program pointer2.cpp:

```
#include<iostream>
using namespace std;

void AddOne (int *ptrParameter);
```

Memory pictures:
int main ()
{
    int *ptr1;

    ptr1 = new int;
    *ptr1 = 10;                      // (a.) what is the picture here

    cout << (*ptr1) << endl;
    AddOne (ptr1);                   // (b.) what is the picture here
                                  //      (after the call)

    cout << (*ptr1) << endl;
    delete ptr1;
    return 0;
} // end main

void AddOne (int *ptrParameter) // (c.) show where ptrParameter points
                                  // to when AddOne is called in
                                  // the program above
{
    *ptrParameter = *ptrParameter + 1;
} // end AddOne

Predicted output: Actual output:

Explain any discrepancies:

Program pointer3.cpp:

```
#include <iostream>
using namespace std;

const int MAX = 10;
```
int main ()
{
    int *ptr1, // for arrays.
        *ptr2; // (a.) what it the picture here
    ptr1 = new int [MAX]; // (b.) what is the picture here
    for (int i = 0; i < 6; i++)
    {
        ptr1[i] = i * 10;
    } // end for
    ptr2 = ptr1; // (c.) what is the picture here
    for (int j = 0; j < 6; j++)
    {
        cout << ptr2[j] << endl;
    } // end for
    delete [] ptr1;
    return 0;
} // end main

Predicted output:                                Actual output:

Explain any discrepancies: