Task 1

Create a new project and add a header comment to it which includes your name, email and this assignment number.

Task 2

On the exams this semester, its likely we'll ask questions that require you to exactly state the output of some code fragment. There are plenty of chances for you to make disastrous mistakes! You need to practice these concepts over and over. Here is a code fragment that you should be able to produce the output of. Place the code into your program, but before you do, try to write the answer down on paper first. Oh, just to be nice I've written the answer below ... don't peek.

```
int* p1, p2;
p1 = new int;
*p1 = 42;
p2 = p1;

cout << "*p1 = " << *p1 << endl;
cout << "*p2 = " << *p2 << endl;

*p2 = 53;

cout << "*p1 = " << *p1 << endl;
cout << "*p2 = " << *p2 << endl;

p1 = new int;
*p1 = 88;

cout << "*p1 = " << *p1 << endl;
cout << "*p2 = " << *p2 << endl;

cout << "Hope you got the point of this example!\n";
```

Don't look at this until you have written your answer on paper

Don't look at this until you have written your answer on paper.

Don't look at this until you have written your answer on paper.

Don't look at this until you have written your answer on paper.

Don't look at this until you have written your answer on paper.

Don't look at this until you have written your answer on paper.

Don't look at this until you have written your answer on paper.

The output *should* be:

```
*p1 == 42
*p2 == 42
*p1 == 53
*p2 == 53
*p1 == 88
*p2 == 53
Hope you got the point of this example!
```

Task 3:

If you got this answer, congratulations! ... you are wrong!! How can you be wrong, if you got the "right" answer? Well the code shown above won't compile. Copy the code into your program and try it. Now fix it! The answer shown above is correct ... the code has one small error. Find it and fix it, and explain what the problem is.

Task 4:

Add the following code. Write output statements with appropriate labels to display each value and its location in memory.

```
int a(10);
int b = 20;
int* c;
c = new int(30);
int* d, e;
d = new int(40);
e = 50;
int* f;
f = d;
```

Task 5:

Add the following code. When you are done using a dynamic variable (one created with the new operator), you should return its memory to the heap. You do this with the "delete" operator.

```
int* A;
A = new int(100);
int * B;
B = A;
cout << "The value of A is " << *A << endl;
cout << "A is pointing to the location " << A << endl;
cout << "The value of B is " << *B << endl;
cout << "B is pointing to the location " << B << endl;
cout << "Returning the memory for A to the heap: " << endl;
delete A;
cout << "The value of A is " << *A << endl;
cout << "A is pointing to the location " << A << endl;
cout << "The value of B is " << *B << endl;
cout << "B is pointing to the location " << B << endl;
```

You should notice that even though you only returned the memory given to A to the heap, the values associated with B are also affected. You probably would be smart enough not to try to dereference A after you intentionally destroyed it, but it is much more difficult to avoid accidentally dereferencing B. B will not usually be created right after A! The undefined pointer variables are know as dangling pointers. If you use a dangling pointer, your program may crash. Or maybe it will just give you some odd results.

Task 6:

Add code to create an array (not a vector) of five ints, with values initialized to the square of the array index. Display it.

Task 7:

Allow the user to enter new values for the array. Prompt the user to enter the number of new values he/she wants to create. Get those values from the user. Display the array.

Task 8:

If the user enters five or fewer values, everything is fine. What happens if the user enters 6 values? The last value does not appear in the array. Where did it go??? See if you can find the location of the missing value by using the Debugger.

Task 9:

If we do not know in advance how many values the user will want to store in an array, then we can not create the array in the same way that we did in the previous task. Because the system places the array in the same area of memory as regular variables, we were required to state in advance (before the program runs) how much memory the array would require. Therefore, we had to place some constant expression inside the square brackets in the array declaration. If we need to create the array while the program is running, we must ask the system to give us storage space while the program is running. That is, we need to use dynamic memory for the array. We make this request using the new operator just like we have done for our simpler pointer objects above. Since the name of an array is simply the address of the beginning of the array, this should be reasonable.

Add code to prompt the user to enter the number of values to store in an array. This time, create a dynamic array to hold the values. Display the values.

Task 10:

Remember that a c-string is just an array of characters pointed to by a base pointer and terminated with a null terminating character, ‘\0’. Write a C++ function which will take a c-string (char*) and return true if the c-string is a palindrome. You are NOT given any other parameters to the function and you are allowed, but not required, to create other functions to support your needs.