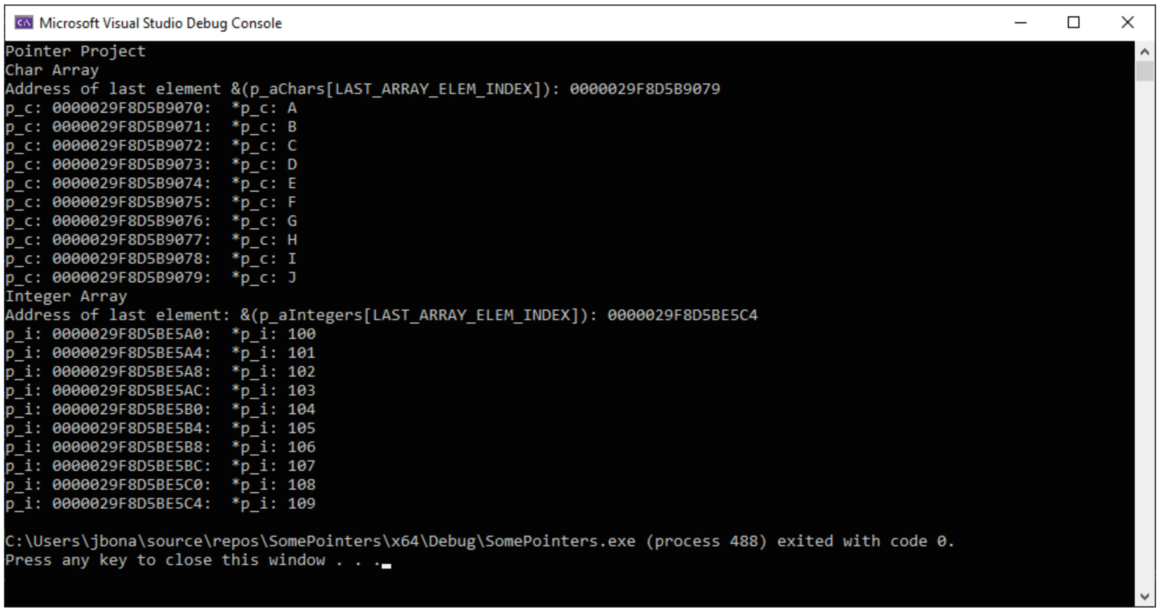
Programming Project: Character and Integer Pointers

Download the file CharIntPointers_incomplete.cpp.

Build and execute the program.

The output will be similar to the following:



```
Pointer Project
Char Array
Address of last element &(p_aChars[LAST_ARRAY_ELEM_INDEX]): 0000029F8D5B9079
p_c: 0000029F8D5B9070:  *p_c: A
p_c: 0000029F8D5B9071:  *p_c: B
p_c: 0000029F8D5B9072:  *p_c: C
p_c: 0000029F8D5B9073:  *p_c: D
p_c: 0000029F8D5B9074:  *p_c: E
p_c: 0000029F8D5B9075:  *p_c: F
p_c: 0000029F8D5B9076:  *p_c: G
p_c: 0000029F8D5B9077:  *p_c: H
p_c: 0000029F8D5B9078:  *p_c: I
p_c: 0000029F8D5B9079:  *p_c: J
Integer Array
Address of last element: &(p_aIntegers[LAST_ARRAY_ELEM_INDEX]): 0000029F8D5BE5C4
p_i: 0000029F8D5BE5A0:  *p_i: 100
p_i: 0000029F8D5BE5A4:  *p_i: 101
p_i: 0000029F8D5BE5A8:  *p_i: 102
p_i: 0000029F8D5BE5AC:  *p_i: 103
p_i: 0000029F8D5BE5B0:  *p_i: 104
p_i: 0000029F8D5BE5B4:  *p_i: 105
p_i: 0000029F8D5BE5B8:  *p_i: 106
p_i: 0000029F8D5BE5BC:  *p_i: 107
p_i: 0000029F8D5BE5C0:  *p_i: 108
p_i: 0000029F8D5BE5C4:  *p_i: 109

C:\Users\jbona\source\repos\SomePointers\x64\Debug\SomePointers.exe (process 488) exited with code 0.
Press any key to close this window . . .
```

The program dynamically allocates two arrays, one of type character and one of type integer:

```
    const int ARRAY_SIZE = 10;
    const int LAST_ARRAY_ELEM_INDEX = 9;

    char* p_aChars = new char[ARRAY_SIZE];
    int* p_aIntegers = new int[ARRAY_SIZE];
```

The arrays are initialized using a “brute force” approach:

```
// Initialize character array
    p_aChars[0] = 'A';
    p_aChars[1] = 'B';
    p_aChars[2] = 'C';
    p_aChars[3] = 'C';
    p_aChars[4] = 'E';
    p_aChars[5] = 'F';
    p_aChars[6] = 'G';
    p_aChars[7] = 'G';
    p_aChars[8] = 'I';
    p_aChars[9] = 'J';

    // Initialize character array
    p_aIntegers[0] = 100;
    p_aIntegers[1] = 101;
    p_aIntegers[2] = 102;
    p_aIntegers[3] = 103;
    p_aIntegers[4] = 104;
    p_aIntegers[5] = 105;
    p_aIntegers[6] = 106;
    p_aIntegers[7] = 107;
    p_aIntegers[8] = 108;
    p_aIntegers[9] = 109;
```

They are then printed out using two while loops. For example, here is the while loop used to print out the character array:

```
    char* p_c = p_aChars;  // Same as p_c = &(p_aChars[0]);


    while (p_c <= &(p_aChars[LAST_ARRAY_ELEM_INDEX])) {

        std::cout << "p_c: " << static_cast<void*>(p_c)
                  << ":  *p_c: " << *p_c << std::endl;

        p_c++;  // Increment the pointer (to point to next array element)
    }
```

Step 1

Comment out the brute force code to initialize the arrays and replace it with two while loops, one while loop to initialize the character array and one to initialize the integer array.

To quickly comment out the brute force initialization code, surround the code using the #if 0 and #endif preprocessor directives:

```
#if 0
    // Initialize character array
    p_aChars[0] = 'A';
    p_aChars[1] = 'B';
    p_aChars[2] = 'C';
    p_aChars[3] = 'C';
    p_aChars[4] = 'E';
    p_aChars[5] = 'F';
    p_aChars[6] = 'G';
    p_aChars[7] = 'G';
    p_aChars[8] = 'I';
    p_aChars[9] = 'J';

    // Initialize character array
    p_aIntegers[0] = 100;
    p_aIntegers[1] = 101;
    p_aIntegers[2] = 102;
    p_aIntegers[3] = 103;
    p_aIntegers[4] = 104;
    p_aIntegers[5] = 105;
    p_aIntegers[6] = 106;
    p_aIntegers[7] = 107;
    p_aIntegers[8] = 108;
    p_aIntegers[9] = 109;

#endif
```

The code will now not be seen by the compiler; the C preprocessor removes it before the intermediate file is presented to the compiler.

Step 2

Next, place your while loop that initializes the character array at the location identified by a TODO comment:

```
 // TODO: Write a while loop that initializes the values of the array pointed to by p_aChars
 //       Use pointer p_c. In the loop use *p_c = value
 //       You may need to introduce additional variables such as char c = 'A'

 // INSERT WHILE LOOP HERE
```

Note that you can do the following:

c = ‘A’;
c++;

The value of c will be ‘B’.

Make sure that you use the * pointer de-referencing operator (e.g. *p_c) when assigning values into the array.

Observe that the value of p_c is set to the first element of the character array just above the place where you insert your while loop:

```
char* p_c = p_aChars;  // Same as p_c = &(p_aChars[0]);
```

Step 3

Finally, place your while loop that initializes the integer array at the location identified by a TODO comment:

```
// TODO: Write a while loop that initializes the values of the array pointed to by p_aIntegers
//       Use pointer p_i.In the loop use *p_i = value where value is 'A', 'B', etc.
//       You may need to introduce additional variables such as int i = 100.

// INSERT YOUR WHILE LOOP
```

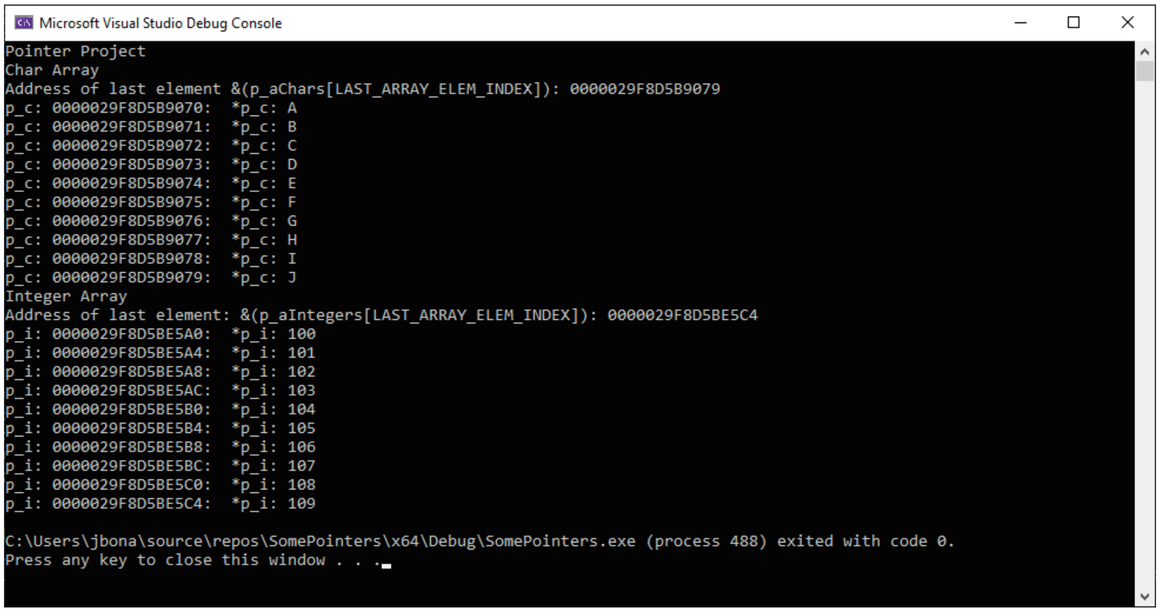
Observe that the value of p_i is set to the first element of the integer array just above the place where you insert your while loop:

```
int* p_i = p_aIntegers;  // Same as p_i = &(p_aOrdinals[0]);
```

Step 4

Build and run your program.

The output of the program should appear nearly identical to your first run except that the values of the addresses (p_c and p_i) may be different:



```
Pointer Project
Char Array
Address of last element &(p_aChars[LAST_ARRAY_ELEM_INDEX]): 0000029F8D5B9079
p_c: 0000029F8D5B9070:  *p_c: A
p_c: 0000029F8D5B9071:  *p_c: B
p_c: 0000029F8D5B9072:  *p_c: C
p_c: 0000029F8D5B9073:  *p_c: D
p_c: 0000029F8D5B9074:  *p_c: E
p_c: 0000029F8D5B9075:  *p_c: F
p_c: 0000029F8D5B9076:  *p_c: G
p_c: 0000029F8D5B9077:  *p_c: H
p_c: 0000029F8D5B9078:  *p_c: I
p_c: 0000029F8D5B9079:  *p_c: J
Integer Array
Address of last element: &(p_aIntegers[LAST_ARRAY_ELEM_INDEX]): 0000029F8D5BE5C4
p_i: 0000029F8D5BE5A0:  *p_i: 100
p_i: 0000029F8D5BE5A4:  *p_i: 101
p_i: 0000029F8D5BE5A8:  *p_i: 102
p_i: 0000029F8D5BE5AC:  *p_i: 103
p_i: 0000029F8D5BE5B0:  *p_i: 104
p_i: 0000029F8D5BE5B4:  *p_i: 105
p_i: 0000029F8D5BE5B8:  *p_i: 106
p_i: 0000029F8D5BE5BC:  *p_i: 107
p_i: 0000029F8D5BE5C0:  *p_i: 108
p_i: 0000029F8D5BE5C4:  *p_i: 109

C:\Users\jbona\source\repos\SomePointers\x64\Debug\SomePointers.exe (process 488) exited with code 0.
Press any key to close this window . . .
```

The values of the pointers (which are addresses) are given in hexadecimal (base 16).

Note that the values of p_c (the pointer to character) increase by 1 each time as p_c is incremented using p_c++:

p_c: 0000029F8D5B90**70**: *p_c: A
p_c: 0000029F8D5B90**71**: *p_c: B
p_c: 0000029F8D5B90**72**: *p_c: C
p_c: 0000029F8D5B90**73**: *p_c: D
…

However, the values of p_i (the pointer to integer) increase by 4 each time p_i is incremented using p_i++:

p_i: 0000029F8D5BE5**A0**: *p_i: 100
p_i: 0000029F8D5BE5**A4**: *p_i: 101

```
p_i: 0000029F8D5BE5A8:  *p_i: 102
p_i: 0000029F8D5BE5AC:  *p_i: 103
…
```

Step 5

Why do character pointers increment their address by 1 yet integer pointer increment their address by 4?
Place your answer in the Canvas comments box.

Note that the pointer values are 48 bits, even though the program is compiled as a 64-bit application and run on a 64-bit Windows machine. **Extra Credit if you can explain why**.

Notes

You may have noticed the use of a static_cast<void *> in the while loop that outputs the character array.

```
while (p_c <= &(p_aChars[LAST_ARRAY_ELEM_INDEX])) {

      std::cout << "p_c: " << static_cast<void*>(p_c) << ":  *p_c: " << *p_c << std::endl;

      p_c++;  // Increment the pointer (to point to next array element)
  }
```

The character pointer is cast to a void * pointer because character pointers have "special meaning" to cout and must be cast to a different type of pointer, and that different type of pointer is a void * pointer. Character pointers are treated as C strings by cout and must be cast to override that interpretation.