Assignment 3 – Pointers and Arrays

Due: Oct 2, 2014, 8:00 AM

Late penalties will be as described in the syllabus.
Points may be deducted for failure to follow these instructions.

General Objectives:

Learn how to use and apply C++ pointers and their relation to arrays.
Demonstrate ability to think and communicate logically

Graded Part of the Assignment:

The worksheet begins with some background material, after which are several activity and question sections.

1. Fill in all the blanks in the question sections.

2. Type your name into the header blank (above right corner)

What to turn in:

Save a copy of this Word Document named: **A03_LAA_*yourlastname*.docx**
Be certain the items noted above are completed.
Submit the completed document to the appropriate D2L dropbox.

Background Material:

Some content has been derived from Data Structures Using C++, 2nd Edition by D.S. Malik

What are pointers? It is a simple question, perhaps.

Consider the statement:

```
int j  = 77;
```

From this statement the compiler assigns an address to the variable named j. This means j will be stored at one specific address in the computer's (RAM) memory. Exactly which part of the computer's memory depends on various details occurring at runtime as well as at compile time. Things that hold influence include what operating system, what hardware architecture, and similar. The details of this are not, for the moment, that important to the topic at hand.

What is important to understand is that when a variable is declared, memory storage with an address is reserved for it. The memory address is fixed, but the data value of the variable can be changed during execution.

Again referencing the statement:

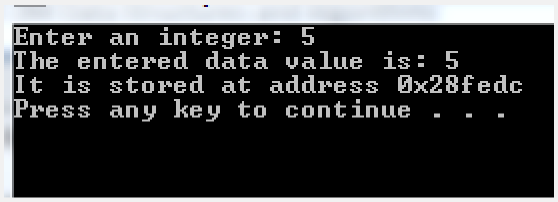
j = 77;

We see the data value 77 is temporarily placed in j and may be used later. In other words, the name of the variable and its address are fixed once the program starts executing, however, its contents may be changed.

In general practice the address of the variable is displayed as a hexadecimal number. This value may be displayed to the screen by use of the **&** (address-of operator). As an experiment, create and run the following program:

```
#include <iostream>
using namespace std;
int main()
{
   int myvar;
   cout << "Enter an integer: ";
   cin >> myvar;
   cout << "The entered data value is: " << myvar << endl;
   cout << "It is stored at address " << &myvar << endl;
   return 0;
}
```

This will produce output similar to the following:



```
Enter an integer: 5
The entered data value is: 5
It is stored at address 0x28fedc
Press any key to continue . . .
```

For 32-bit systems the 0x28fedc = 0x0028fedc hexadecimal will be converted to 32 binary bits
0000 0000 0010 1000 1111 1110 1101 1100

or rather 4 bytes

```
[0    0  ][ 2    8 ] [ f     e  ][ d     c  ]    ← hexadecimal
[0000 0000] [0010 1000] [1111 1110] [1101 1100]  ← binary
```

How does this relate to pointers?

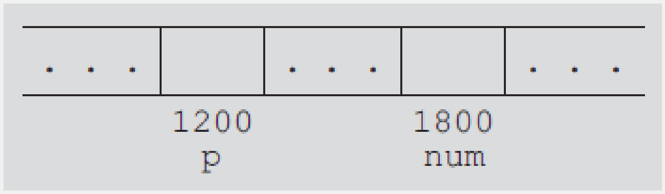
Just as we can create variables to store integer, character and floating point values, we can also create variables to store memory addresses of other variables.

Variables that store memory addresses instead of actual data values are called pointers.

Consider the statements:

int *p;
int num;

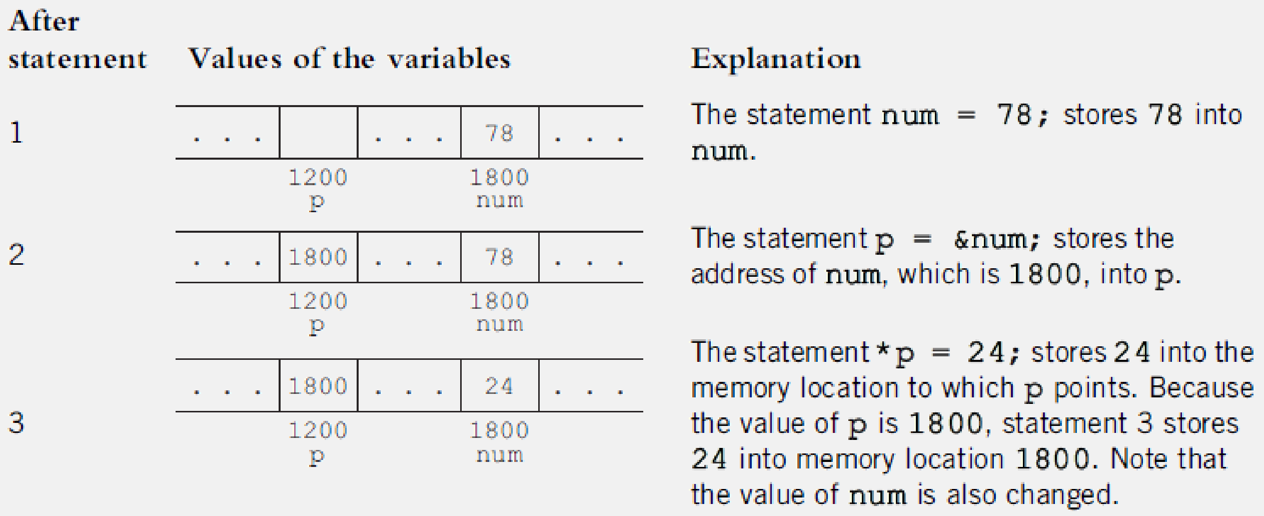
The variable named p is a pointer to an int, and num is a variable of type int. Assume that memory location 1200 is allocated for p and memory location 1800 is allocated for num, as shown in the figure below



Continuing from the previous statements, consider the following additional lines of code:

```
1.      num = 78;
2.      p = &num;
3.      *p = 24;                // recall this de-references p
```

The following shows the values of the variables after the execution of each statement.



After statement	Values of the variables	Explanation
1	`. . .` ` ` `. . .` `78` `. . .` 1200 p 1800 num	The statement `num = 78;` stores 78 into `num`.
2	`. . .` `1800` `. . .` `78` `. . .` 1200 p 1800 num	The statement `p = &num;` stores the address of `num`, which is 1800, into `p`.
3	`. . .` `1800` `. . .` `24` `. . .` 1200 p 1800 num	The statement `*p = 24;` stores 24 into the memory location to which `p` points. Because the value of `p` is 1800, statement 3 stores 24 into memory location 1800. Note that the value of `num` is also changed.

Pointer Arithmetic

We can add or subtract from pointer variables. However, adding a number to a pointer does not add that many bytes but rather adds that number of storage units. Suppose the size of memory allocated for an int variable is 4 bytes, a double variable is 8 bytes, and a char variable is 1 byte. Now consider the statements:

```
int     *p;
double *q;
char    *chPtr;
```

The statement p++; or p = p + 1;
increments the value of p by 4 bytes because p is a pointer to type int.

The statement q++;
increments the value of q by 8 bytes because q is a pointer to type double.

The statement chPtr++;
increments the value of chPtr by 1 byte because it is a pointer to type char.

The statement p = p + 3;
increments the value of p by 12 bytes (the size of three integers)

Relating Arrays to Pointers

When arrays are defined, the array name holds the starting address of the array. Just as when we declare a single variable, the array's address is fixed. Thus the array name is a *constant* pointer. It holds an address but it cannot be made to point to another address.

Consider the following statements:

```
int *p;
int j[ ] = { 2, 6, 3, 7, 4 };
```

Assume the array is allocated starting at memory location 0x2b02:

2b02	2b06	2b0a	2b0e	2b12
2	6	3	7	4
j[0]	j[1]	j[2]	j[3]	j[4]

The integer array, as shown above is allocated with 4 bytes for each slot of the array.

The statement p = j;
would set p to have the value 0x2b02

The statement p = &j[3];
would set p to have the value 0x2b0e

And if followed by p++;
would set p to have the value 0x2b12

The statement j = p;
would result in a compiler error, as you cannot change the value of a constant pointer.

Pointers may also be used to create dynamic arrays.

This requires caution as you can lose the address where the array starts if you are not careful.
You must also de-allocate the memory you dynamically allocate to avoid memory leaks.
The following code segment illustrates how to dynamically allocate an array to match the size requested by the user. The new operator allocates the memory. The delete [] operator de-allocates it.

```
int *intList;
int arraySize;
cout << "Enter array size: ";
cin >> arraySize;
cout << endl;
intList = new int[arraySize];
   //... code skipped here
delete [] intList;
intList = NULL;
```

CS-244 Data Structures and Algorithms Name:

Problem 1

Build and run the following program.
Based on the results, fill in the table as indicated.

```
#include <iostream>

using namespace std;

int main()
{
   int i = 23, j = 55;
   cout << " i = " << i  << ",\t  j = " << j << endl;
   cout << "&i = " << &i << ",\t &j = " << &j << endl;
   return 0;
}
```

Variable Name	Data Value	Address
i		
j		

Problem 2

Build and run the following program. Based on the results, answer the questions as indicated.

```
#include <iostream>
using namespace std;
int main()
{
   int i = 55, j = 100;
   cout << " i = " << i  << ",\t  j = " << j << endl;
   cout << "&i = " << &i << ",\t &j = " << &j << endl;

   // assign new values...
   i = 7;
   j = 8;
   // print again
   cout << " i = " << i  << ",\t  j = " << j << endl;
   cout << "&i = " << &i << ",\t &j = " << &j << endl;

   return 0;
}
```

Part a)

After the second cout add the statement:

&i = 4;

Attempt to compile and run.

Are we allowed to change the addresses of variables?

____

Part b)

After the fourth cout add the statement:

j = &i;

Attempt to compile and run.

Is j a pointer variable?

____

Are we allowed to assign the address of i to j,
using this syntax?

____

Part c)

Replace the code from part b with:

j = (int)&i; *// may want couts after this too*

Attempt to compile and run.

Are we allowed to assign the address of i to j,
using this syntax?

____

Part d)

What do you conclude from parts b and c?

COMPLETE THIS PAGE in the document you turn in to D2L --- with your answers clearly indicated

Problem 3

While integer type variables can store memory address values, the compiler will still treat their data value as just a number. Pointer variables store memory addresses and are recognized by the compiler to be treated in "special" ways.

Build and run the following program. Based on the results, answer the questions and fill in the tables as indicated.

```
#include <iostream>

using namespace std;

int main()
{
   int i = 7;        // Statement 1
   int *pOne;        // Statement 2
   float x = 0.00;

   cout << "i = " << i << "\t&i = " << &i << endl;

   pOne = &i;        // Statement 3
   cout << "pOne = " << pOne << endl;

   return 0;
}
```

Part a)
Complete the table:

Variable Name	Data Value	Address
i		
pOne		

Part b) Try assigning an address of a float as shown below to pOne. Does it work? ________________

```
pOne = &x;
```

COMPLETE THIS PAGE in the document you turn in to D2L --- with your answers clearly indicated

CS-244 Data Structures and Algorithms Name:

Problem 4

This problem set is to demonstrate the relationship between array names, subscripts, and pointer arithmetic.
Suggest: create a table showing the value and memory address for each variable in both arrays.
Consider the following code segment:

```
int main()
{
   int   intArray[10]={2,4,6,8};     // array of 10 integers
   char  chrArray[11];               // array of 11 characters,
   .                                 // room for 10 useable characters
   .                                 // and the null terminator
   .
}
```

Assume the space reserved for the integer array starts at memory address: 4000
Assume the space reserved for the character array starts at memory address: 1800

a) Where does the integer array end? *Hint: assume 32-bit integers* a) ____________________

b) Where does the character array end? b) ____________________

c) What is the value of **intArray**? c) ____________________

d) What is the value of **charArray**? d) ____________________

e) What is the value of **intArray[0]**? e) ____________________

f) What is the value of **intArray[3]**? f) ____________________

g) What is the value of **intArray + 1**? g) ____________________

h) What is the value of **intArray + 9**? h) ____________________

i) "The name of an array is a pointer to the array."
Describe the difference between the expressions **chrArray** and ***chrArray**

____

____

j) Describe the difference between **intArray + 1**, **intArray[1]**, and ***(intArray + 1)**

____

____

k) What is the value of **&chrArray[0]**? k) ____________________

l) What is the value of **&chrArray[1]**? l) ____________________

m) What is the value of **&chrArray**? m) ____________________

COMPLETE THIS PAGE in the document you turn in to D2L --- with your answers clearly indicated
-=-=-=