Nick Krisulevicz

Dr. Wang

COSC 120-751

01/25/2021

Lab 10

Lab 10.1

Source Code:

```
//Nick Krisulevicz
//Lab 10.1
//01/25/2021

/*
  Pointers.cpp
  COSC-220 Lab 9
  Based on a lab designed by Thomas Horseman

  Thomas Anastasio
  Created: April 8, 2001
  Current: March 20, 2003
*/

#include <iostream>
#include <stddef.h>     // for NULL definition

int main()
{
  using namespace std;

  int int1 = 1, int2 = 2, int3 = 3;
```

```
 double dub1 = 1.0, dub2 = 2.0, dub3 = 3.0;
 float flt1 = 1.0f;
 char chr1 = 'a', chr2 = 'b';

 ////////////////////////////////////////////////////////////////
 // 1. Experiment with storage sizes and addresses of various data
 // types.
 ////////////////////////////////////////////////////////////////
 //  A.  Add statements to print the addresses of each of the 9
 //  variables declared above and record the addresses.
 //  B.  Add statements to print the sizes (in bytes) of each of the
 //  9 variables  (use the sizeof operator).
 //  C.  Add statements to print the sizes of the addresses of the 9
 //  variables (again, use the sizeof operator).

 //  Write your code here:

cout << "int1 address = " << &int1 << endl;
cout << "int2 address = " << &int2 << endl;
cout << "int3 address = " << &int3 << endl;
cout << "dub1 address = " << &dub1 << endl;
cout << "dub2 address = " << &dub2 << endl;
cout << "dub3 address = " << &dub3 << endl;
cout << "flt1 address = " << &flt1 << endl;
cout << "chr1 address = " << &chr1 << endl;
cout << "chr2 address = " << &chr2 << endl;

cout << endl;
```

```
cout << "Size of int1 = " << sizeof(int1) << endl;
cout << "Size of int2 = " << sizeof(int2) << endl;
cout << "Size of int3 = " << sizeof(int3) << endl;
cout << "Size of dub1 = " << sizeof(dub1) << endl;
cout << "Size of dub2 = " << sizeof(dub2) << endl;
cout << "Size of dub3 = " << sizeof(dub3) << endl;
cout << "Size of flt1 = " << sizeof(flt1) << endl;
cout << "Size of chr1 = " << sizeof(chr1) << endl;
cout << "Size of chr2 = " << sizeof(chr2) << endl;

cout << endl;

cout << "Size of int1 address = " << sizeof(&int1) << endl;
cout << "Size of int2 address = " << sizeof(&int2) << endl;
cout << "Size of int3 address = " << sizeof(&int3) << endl;
cout << "Size of dub1 address = " << sizeof(&dub1) << endl;
cout << "Size of dub2 address = " << sizeof(&dub2) << endl;
cout << "Size of dub3 address = " << sizeof(&dub3) << endl;
cout << "Size of flt1 address = " << sizeof(&flt1) << endl;
cout << "Size of chr1 address = " << sizeof(&chr1) << endl;
cout << "Size of chr2 address = " << sizeof(&chr2) << endl;

 // Record your results here:
 //
 // Variable  Size   Address   Address-Size
 //  int1    4     0x61fe1c  8
 //  int2    4     0x61fe18  8
 //  int3    4     0x61fe14  8
 //  dub1    8     0x61fe08  8
```

```
// dub2    8     0x61fe00  8
// dub3    8     0x61fdf8  8
// flt1    4     0x61fdf4  8
// chr1    1     a         8
// chr2    1     ba        8



// How many bytes of storage is allocated for each data type?  How
// many bytes is allocated for each pointer (address). Notice that
// the data types are not necessarily the same size, but the
// pointers are.
//
// Did the address of the char variables print as you expected?  If
// not, why not?   You can force the printed output to be
// interpreted as a pointer by coercing it (casting) to void* like
// this:
//    cout << (void *) &chr1;
//
// Run the program again, using the coercion for the char
// pointers. Record your data.


//QUESTION ANSWERS PART 1:
//1. There are four bytes allocated for int variables, eight for double, four for float, one for char, and eight for pointers.
//2. The two char variables did not output the address as expected. This was fixed by changing the cout statement to include (void *) which coerced the output to be treated like a pointer.
```

```
 cout << "----------------------------------------" << endl << endl;

 // Write your modified code here:

cout << "int1 address = " << &int1 << endl;
cout << "int2 address = " << &int2 << endl;
cout << "int3 address = " << &int3 << endl;
cout << "dub1 address = " << &dub1 << endl;
cout << "dub2 address = " << &dub2 << endl;
cout << "dub3 address = " << &dub3 << endl;
cout << "flt1 address = " << &flt1 << endl;
cout << "chr1 address = " << (void *)&chr1 << endl;
cout << "chr2 address = " << (void *)&chr2 << endl;

cout << endl;

cout << "Size of int1 = " << sizeof(int1) << endl;
cout << "Size of int2 = " << sizeof(int2) << endl;
cout << "Size of int3 = " << sizeof(int3) << endl;
cout << "Size of dub1 = " << sizeof(dub1) << endl;
cout << "Size of dub2 = " << sizeof(dub2) << endl;
cout << "Size of dub3 = " << sizeof(dub3) << endl;
cout << "Size of flt1 = " << sizeof(flt1) << endl;
cout << "Size of chr1 = " << sizeof(chr1) << endl;
cout << "Size of chr2 = " << sizeof(chr2) << endl;

cout << endl;

cout << "Size of int1 address = " << sizeof(&int1) << endl;
```

```
cout << "Size of int2 address = " << sizeof(&int2) << endl;
cout << "Size of int3 address = " << sizeof(&int3) << endl;
cout << "Size of dub1 address = " << sizeof(&dub1) << endl;
cout << "Size of dub2 address = " << sizeof(&dub2) << endl;
cout << "Size of dub3 address = " << sizeof(&dub3) << endl;
cout << "Size of flt1 address = " << sizeof(&flt1) << endl;
cout << "Size of chr1 address = " << sizeof(&chr1) << endl;
cout << "Size of chr2 address = " << sizeof(&chr2) << endl;

 // Record your results here:
 //
 //  Variable  Size   Address   Address-Size
 //    chr1   1     0x61fdf3  8
 //    chr2   1     0x61fdf2  8

 cout << "----------------------------------------" << endl << endl;

 ////////////////////////////////////////////////////////////////
 // 2. Experiment with pointer variables.
 ////////////////////////////////////////////////////////////////
 //   A. Declare pointer vaiables intPtr1, intPtr2, intPtr3, dubPtr1,
 //   dubPtr2, dubPtr3, fltPtr1, chrPtr1, and chrPtr2 so that they
 //   can be used to point to the  appropriate variables.  Assign the
 //   address of the appropriate variable to them and print their
 //   values.  (The char pointers will have to be coerced as before
 //   for printing).

 //  Write your code here:
 int *intPtr1, *intPtr2, *intPtr3;
```

```
double *dubPtr1, *dubPtr2, *dubPtr3;
float *fltPtr1;
char *chrPtr1, *chrPtr2;

intPtr1 = new int;
intPtr2 = new int;
intPtr3 = new int;
dubPtr1 = new double;
dubPtr2 = new double;
dubPtr3 = new double;
fltPtr1 = new float;
chrPtr1 = new char;
chrPtr2 = new char;

cout << "intPtr1 value = " << intPtr1 << endl;
cout << "intPtr2 value = " << intPtr2 << endl;
cout << "intPtr3 value = " << intPtr3 << endl;
cout << "dubPtr1 value = " << dubPtr1 << endl;
cout << "dubPtr2 value = " << dubPtr2 << endl;
cout << "dubPtr3 value = " << dubPtr3 << endl;
cout << "fltPtr1 value = " << fltPtr1 << endl;
cout << "chrPtr1 value = " << (void *) chrPtr1 << endl;
cout << "chrPtr2 value = " << (void *) chrPtr2 << endl;

//  Report your results here:
//
//   Pointer  Printed
//  Variable   Value
//   intPtr1  0xe417f0
```

```
// intPtr2  0xe41810
// intPtr3  0xe41a50
// dubPtr1  0xe41a70
// dubPtr2  0xe41a90
// dubPtr3  0xe45c60
// fltPtr1  0xe45c80
// chrPtr1  0xe45ca0
// chrPtr2  0xe45cc0

// How do the values of the pointers compare to the values of the
// addresses you got in the previous experiment?

//QUESTION ANSWERS PART 2
//1. The values of the pointer variables are different from the addresses of the variables. This is because the pointers are separate variables
//which store the addresses of the variables, and they have their own addresses.

cout << "----------------------------------------" << endl << endl;

//////////////////////////////////////////////////////////////
// 3. Experiments with NULL pointers and with dereferencing
//////////////////////////////////////////////////////////////
// A. Assign NULL to fltPtr1, then print the value of fltPtr1.
// B. Accessing the storage locations through the relevant
// pointers, assign the following values and print them.
//     int1  12
//     int2  22
//     dub1  10.1
```

```
//      dub2  20.2
//      flt1  30.3

// Write your code here:
//fltPtr1 = NULL;
//cout << "fltPtr1 = " << fltPtr1 << endl;

*intPtr1 = 12;
*intPtr2 = 22;
*dubPtr1 = 10.1;
*dubPtr2 = 20.2;
*fltPtr1 = 30.3;

cout << "intPtr1 dereference = " << *intPtr1 << endl;
cout << "intPtr2 dereference = " << *intPtr2 << endl;
cout << "dubPtr1 dereference = " << *dubPtr1 << endl;
cout << "dubPtr2 dereference = " << *dubPtr2 << endl;
cout << "fltPtr1 dereference = " << *fltPtr1 << endl;


// Your program should have terminated with a run-time error.
// What error was reported?
//
//
//
// Why did it occur?
//
//
```

```
//   C.  Fix the problem and repeat.


// Write your code here:
cout << endl;
*intPtr1 = 12;
*intPtr2 = 22;
*dubPtr1 = 10.1;
*dubPtr2 = 20.2;
*fltPtr1 = 30.3;


cout << "intPtr1 dereference = " << *intPtr1 << endl;
cout << "intPtr2 dereference = " << *intPtr2 << endl;
cout << "dubPtr1 dereference = " << *dubPtr1 << endl;
cout << "dubPtr2 dereference = " << *dubPtr2 << endl;
cout << "fltPtr1 dereference = " << *fltPtr1 << endl;


//QUESTION ANSWERS PART 3
//1. A null pointer points to an address of 0. This is not a valid address and there is nothing stored there to dereference.


cout << "----------------------------------------" << endl << endl;


//////////////////////////////////////////////////////////////////
//  4. Experiments with pointer arithmetic
//////////////////////////////////////////////////////////////////
//    A.  Print the values of intPtr2, (intPtr2+1) and
//    (intPtr2-1). Did you get what you expected?


// Write your code here:
```

```
cout << "intPtr2 = " << intPtr2 << endl;
cout << "intPtr2 + 1 = " << intPtr2 + 1 << endl;
cout << "intPtr2 - 1 = " << intPtr2 - 1 << endl << endl;


// Write your explanation here:
//
//
//
//
//
//


//   B. Print the value of intPtr1.
//   C. Use the increment operator to increment intPtr1.
//   D. Print the value of intPtr1 again.
//   E. Print the value of the memory location pointed to by
//   intPtr1.  Explain what is happening.

// Write your code here:
cout << "intPtr1 value = " << intPtr1 << endl;
cout << "intPtr1 value incremented by 1 = " << intPtr1 + 1 << endl;
cout << "intPtr incrememted address = " << &intPtr1 << endl;

// Write your explanation here.
//
//
```

```
//
//


//QUESTION ANSWERS PART 4
//1. The pointer arithmetic produced an interesting result. Relative to intPtr2, +1 and -1 had a difference of four in the hexadecimal address value.
//Incrementing by 1 in decimal form incremented the pointee address by 4.
//2. The pointer arithmetic applied to intPtr1 added to the address it pointed to. When we referenced intPtr1, it output the address the pointer is located in.
cout << "----------------------------------------" << endl << endl;



//////////////////////////////////////////////////////////////////
//  5. Experiments with dynamic allocation
//////////////////////////////////////////////////////////////////
//   A.  Use the new operator to dynamically allocate an integer.
//   B.  Assign the number 6000 to the new location.
//   C.  Print the address of this new location.
//   D.  Print the value stored in the new location.
//   E.  Comment on the address of the new location compared to the
//   addresses seen in part 2.


//  Write your code here:
int *numPtr;
numPtr = new int;
*numPtr = 6000;


cout << "numPtr value = " << *numPtr << endl;
cout << "numPtr address = " << numPtr << endl;
```

```
//  Write your results and comments here:
//
//  new location address =
//  contents of new location =

//  Comment:
//

//  F.  Use the new operator to allocate an array of size 10 of
//  doubles.  Fill the array with 10.0,20.0,...100.0 and print the array.

// Write your code here:

double *arrayPtr;
arrayPtr = new double[10];
double counter = 0.0;
for(int i = 0; i < 10; i++)
{
   counter += 10.0;
   arrayPtr[i] = counter;
   cout << arrayPtr[i] << endl;
}


//QUESTION ANSWERS PART 5
```

```
  //1. The address of the dynamically created variable is different each time you run the program. Each time, the static variables' addresses remain the same,
  //but the dynamic variables is different.



  delete intPtr1;
  delete intPtr2;
  delete intPtr3;
  delete dubPtr1;
  delete dubPtr2;
  delete dubPtr3;
  delete fltPtr1;
  delete chrPtr1;
  delete chrPtr2;
  delete numPtr;
  delete [] arrayPtr;


  return 0;
}
```

Question Answer:

Exercise 1: Output after implementing statements to output the addresses of all nine variables, size of all nine variables, and size of address of all nine variables is as follows.

int1 address = 0x61fe1c

int2 address = 0x61fe18

int3 address = 0x61fe14

dub1 address = 0x61fe08

dub2 address = 0x61fe00

dub3 address = 0x61fdf8

flt1 address = 0x61fdf4

chr1 address = a

chr2 address = ba

Size of int1 = 4

Size of int2 = 4

Size of int3 = 4

Size of dub1 = 8

Size of dub2 = 8

Size of dub3 = 8

Size of flt1 = 4

Size of chr1 = 1

Size of chr2 = 1

Size of int1 address = 8

Size of int2 address = 8

Size of int3 address = 8

Size of dub1 address = 8

Size of dub2 address = 8

Size of dub3 address = 8

Size of flt1 address = 8

Size of chr1 address = 8

Size of chr2 address = 8

There are four bytes allocated for int variables, eight for double, four for float, one for char, and eight for pointers.

The two char variables did not output the address as expected. This was fixed by changing the cout statement to include (void *) which coerced the output to be treated like a pointer. The correct output for these two statements is as follows.

chr1 address = 0x61fdab

chr2 address = 0x61fdaa

Exercise 2: The exercise then wanted to implement new pointer variables for each of the nine variables and to output their values. It is as follows.

intPtr1 value = 0xd517f0

intPtr2 value = 0xd51810

intPtr3 value = 0xd51a50

dubPtr1 value = 0xd51a70

dubPtr2 value = 0xd51a90

dubPtr3 value = 0xd55c60

fltPtr1 value = 0xd55c80

chrPtr1 value = 0xd55ca0

chrPtr2 value = 0xd55cc0

The values of the pointer variables are different from the addresses of the variables. This is because the pointers are separate variables which store the addresses of the variables, and they have their own addresses.

Exercise 3: After running the program for part three, a runtime error occurred. This is because a null pointer points to an address of 0. This is not a valid address and there is nothing stored there to dereference. Once the null pointer was removed, the other five assigned values into the pointees were output. They are as follows.

intPtr1 dereference = 12

intPtr2 dereference = 22

dubPtr1 dereference = 10.1

dubPtr2 dereference = 20.2

fltPtr1 dereference = 30.3

Exercise 4: The pointer arithmetic produced an interesting result. Relative to intPtr2, +1 and -1 had a difference of four in the hexadecimal address value. Incrementing by 1 in decimal form incremented the pointee address by 4. The output is as follows.

intPtr2 = 0xf41810

intPtr2 + 1 = 0xf41814

intPtr2 - 1 = 0xf4180c

The pointer arithmetic applied to intPtr1 added to the address it pointed to. When we referenced intPtr1, it output the address the pointer is located in.

Exercise 5: The address of the dynamically created variable is different each time you run the program. Each time, the static variables' addresses remain the same, but the dynamic variables is different. Sample output from two different runs of the program are as follows.

numPtr value = 6000

numPtr address = 0x715ce0

numPtr value = 6000

numPtr address = 0x785ce0

There was a next part in exercise 5 that wanted to create a new dynamic array and print the values. They are as follows.

10

20

30

40

50

60

70

80

90

100

Lab 10.2

Source Code:

```
//Nick Krisulevicz
//Lab 10.2
//01/25/2021

#include "ThreeDimPt.h"

#include <iostream>

int GetNumbPts();

ThreeDimPt * MakeThreeDimPtArray(int size);

int main()
{
  int nPts = GetNumbPts();
  int i;

  ThreeDimPt * ptArr;

  ptArr = MakeThreeDimPtArray(nPts);


  int x;
  int y;
```

```
  int z;

  cout << "Enter the first dimension: ";
  cin >> x;
  cout << "Enter the second dimension: ";
  cin >> y;
  cout << "Enter the third dimension: ";
  cin >> z;
  cout << endl;

  ptArr->SetX(x);
  ptArr->SetY(y);
  ptArr->SetZ(z);

  for(i = 0; i < nPts; i++)
  {
     cout << *ptArr[i];
  }



  delete [] ptArr;

  return 0;
}

ThreeDimPt * MakeThreeDimPtArray(int size)
{
    int *threeDimArray = nullptr;
    threeDimArray = new int[size];
```

```
   cout << "Three dim array address = " << threeDimArray << endl;
}

int GetNumbPts()
{
  using namespace std;

  int n;

  cout << "How many points? ";
  cin >> n;

  while (n < 0)
   {
     cout << "Number can't be negative" << endl;
     cout << "How many points? ";
     cin >> n;
   }

  return n;
}
```

Question Answer:

The output is as follows.

How many points? 4

Three dim array address = 0x8265f0

Enter the first dimension: 3

Enter the second dimension: 2

Enter the third dimension: 1