CSI 310 Data Structures    Spring 2003    Professor Chaiken
Lab Exercise 2    Using DDD to see Pointers, program shift of array contents.    Due: Feb. 14

1 Prelab

Read pages 141-145 of DSO and this entire handout before you begin this lab. Refer to Lab1 for ddd and other details.

You may structure your program either with a class and member functions or in any simpler way you care to.

2 Begin

In a new, empty directory, begin a C++ program with an integer array A two size_t variables: LAST and USED, and int variable temp. You may declare and use additional variables.

Write a build script.

Make (by reiterating build and editing) your rather trivial program compile and link.

3 Read and Print Half the Array

Code two loops to run successively.

1. Read\(^1\) integers to fill the array approximately half way.

   The code to put each integer into the array should have the form:
   ```
   int temp;
   cin >> temp;
   A[USED] = temp;
   USED = USED + 1;
   ```
   rather than the (better)
   ```
   cin >> A[USED++];
   ```
   because a future exercise will require calling a function to insert the integer. The loop must also update LAST.

   After this loop, the value of LAST must satisfy the invariant:
   ```
   // (The value of) LAST is the index of the last array entry filled by the loop, if there is any.
   ```

2. Print the integers just read from the array, in the order they were read.

   Build and test your program until it works.

4 Prepare for the Experiment with the Array

Run your program under ddd and make it stop just before second loop, where the invariant is true.

Details: (1) Start ddd with ddd executableFileName. (2) Set the breakpoint at main. (3) Activate ddd's “run” operation. (4) Set a new breakpoint in front of the second loop by clicking on a spot in front of that second loop's first line, and then clicking the stop sign “breakpoint” icon. Observe the new stop sign and message about breakpoint 2 in the (gdb) interaction window. (4) Activate ddd's “continue” operation, either from the “Program” pull-down menu or by typing cont(inue).

You will of course have to type in the integers.

\(^1\)All I/O is to be done with standard input and output using the elementary “>>>” and “<<<” operators belonging to cin and cout.
When your program stops at the second breakpoint, put the array contents on dynamic display: Edit \( A[0...28] \) (replace 28 by the last legal index for your array \( A \)) into the upper left edit box and then click the Flashlight “display” icon.

You should see the numbers you typed in followed by “garbage” (or zeros if you used a static array.) Scroll and/or enlarge the display window so you can see them.

5 Use the print Command to evaluate expressions

The debugger’s “print” command evaluates and then prints the value of C++ expressions you type in. Try in the (gdb) window:

```c
print 3
print 2003+3
print 5*7
print 5/7
print (double)5/7
print ((double)(5)/7
print (int)((double)(5)/7
print (double)(5)/7
```

(The last 4 expressions include a C-style cast. For example, \((\text{double})5\) is the double precision floating point number representation resulting from converting integer 5 to double precision.)

Each value printed is also stored in a debugger history variable of the form \$integer. Try:

```c
print $1
print $2
etc...
print $1+$2
print $198
```

Besides doing calculations, the print command lets you examine the process’s state by printing the values of variables. Try a few now.

6 Printing array element VALUES AND ADDRESSES, Dereference expressions

This material will be new to most of you!! It’s a core topic in Data Structures!

Try each command below, others like them, and all variants you can think of. For each one, copy onto this sheet what gdb printed out. For each one, try to FIGURE OUT WHY it printed what you see.

6.1 The “address-of” & operator produces ADDRESS or POINTER values

The C/C++ & operator evaluates to the ADDRESS OF, in other words, the POINTER TO its operand, which must be a variable. Try:

```c
print LAST
See the VALUE of the variable named LAST
print &LAST
(0x... indicates an integer number written in hexadecimal, which is base 16, with the symbols a...f used for digits with values ten...fifteen.) See the ADDRESS of or the POINTER to the variable named LAST.
print &938
```
See that it is illegal (in fact meaningless!) to request the address of a constant which is not in memory.

A memory address or pointer IS A NUMBER. Examples of such numbers 998, 997, 996, etc. are printed in the figure on page 141 of DSO.

Programmers find it convenient to express numerical address or pointer values in hexadecimal notation. Therefore, gdb (by default) uses the hexadecimal system to display a number when the number comes from an expression denoting an address. But numbers, whether expressed in decimal, roman numerals, tally marks, binary, English (e.g. thirty seven) or hexadecimal are still numbers!

Try:
print 938
print /x 938
print 0x3aa
print /x 0x3aa

Take the result of print &LAST and type it after print
Then compare it to the result from print /d &LAST

### 6.2 Array Elements

Now, expressions for array elements like A[0], A[1] and A[LAST] all denote VARIABLES. AS SUCH, the address-of operator will produce their addresses. Try:

print &A[0]
print /d &A[0]
print A[0]

The last command prints the VALUE of the variable denoted by A[0].

Try it for other elements in the array:

print &A[1]
print /d &A[1]
print &A[2]
print /d &A[2]
print &A[LAST]
print /d &A[LAST]
print A[1]
print A[2]
print A[LAST]

### 6.3 Dereferencing

Look back over the contents of the (gdb) prompt window and find some results when print was commanded to print an address. Try to print what is stored at that address by giving the print command on the dereference operator * applied to the address.

Practice this with an address you retype in hex. Also practice this with debugger history variables ($numbers).

What if you try “print * 0”, trying to dereference the NULL pointer?

Try alternations like print &**something, &***, ***, ***, etc.

For the first Lab2 credit (L2.1), the TA will see your build script work, review your notes, and ask you to print similar expressions and then explain their results.
Given `float * P;` what each means

- **&P**: the VALUE of &P is the address of the variable named P.
- **P**: pointer variable P is what’s stored HERE. It is an address or pointer, because P has type pointer.
- ***P**: variable denoted by *P is the VALUE of *P is what’s stored HERE. It’s type is float, the type P is declared to point to.

To dereference P means to access the data in HERE.

---

## 7 Part 2

After you master the implementation of the bag class of chapter 2, do this exercise:

Design a function to be called in place of the line `A[USED] = temp;` to insert an integer into array `A` above. Instead of inserting it at the end of the used part of the array, your function should insert the given integer somewhere between positions 0 and USED of `A` so that the integers remain in non-decreasing order. Here’s an example of a sequence of integers in non-decreasing order: -34, -32, 0, 1, 5, 7, 7, 8, 99.

When this function is used in the reading loop to insert the integers, all the copies of integers will be printed in sorted order by the printing loop.

This function requires

1. Finding where to insert the new integer.
2. “Pushing” the integers at and right of the insertion point to the right, to make room for the new integer without losing any of the old ones.

When you’re done, you can get credit for part 2 either by demonstrating it to a staff member or submitting it to the project `lab2`. 
CSI310 Spring 2003 Lab Exercise 2 Credit Sheet.

Your Name ________________________________

Checkpoint L2.1: Demonstrate to the staff member your build script operating and starting ddd on the newly built program described above. Show your notes written on this lab sheet. Then, the staff member will ask you to command the debugger to print various kinds of expressions and ask you to explain the results.

(L2.1) Recorded: ___________ Signature ___________________________ Date Comments: 

TA: Make sure student’s name is written in INK above.

Checkpoint L2.2: Part 2 was done as described above and works correctly.

(L2.2) Recorded: ___________ Signature ___________________________ Date Comments: 

or record here yourself when you submitted this work to lab2.