LW: Pass By Value

Objectives

- To understand that:
    - argument passing is similar to initialization,
    - when a function is called, each formal argument is initialized by its corresponding actual argument, and
    - when a variable is provided as a formal parameter, the corresponding actual argument's value will be copied into an local object of the called function.

Labwork

1. There are questions at the end of this document that you will respond to for credit.
2. You must compile and run this code on **`build.tamu.edu`**, executing the command `g++ LW_Pass-By-Value.cpp` to compile, and then `./a.out` to run the resultant executable.
3. Download the skeleton code for this lab from **https://drive.google.com/open?id=0B_ouNNuWgNZCQThVWWJMUDZjemM**
4. Upload the source code to your CSCE-drive.
5. Using Putty (PC) or terminal (Mac), log-in to **`build.tamu.edu`**, and navigate to the directory containing `LW_Pass-By-Value.cpp`.
6. Inspect the source code.
    a. You will encounter four calls to a function that you haven't seen before, `vis::print`.
        i. This (i.e., `vis::print`) is a function whose implementation you need not be concerned about: I have written the function `vis::print` to output the elements of a vector for you.
        ii. For instance, when I called `vis::print(v)`, where `v` was defined as a `vector<int>` initialized with `{1,11}` (i.e., `vector<int> v =`

{1,11}), I observed the following output on *my machine*:

```
.--------------------.
| [0] |      1       |
+--------------------+
|   0x7fe002404cb0   |
'--------------------'
           |
.--------------------.
| [1] |     11       |
+--------------------+
|   0x7fe002404cb4   |
'--------------------'
```

- The value of each element of the vector v is contained in its own box:

  ```
  .--------------------.
  | [0] |      1       |
  +--------------------+
  |   0x7fe002404cb0   |
  '--------------------'
             |
  ```

  - On the first line of the box, the integer contained between the brackets (here, [0]) is the index of that respective element in the vector v:

    ```
    .-------
    | [0] |
    +-------
    ```

  - Also on the first line of the box is the value stored at that index in the vector (here, 1):

    ```
    -------------.
    |      1      |
    -------------+
    ```

  - The hexadecimal value shown on the second line of the box is the address in memory where that integer value is being stored (here, v.at(0); this will likely be different each time you run your program):

    ```
    +--------------------+
    |   0x7fa9b9500000   |
    '--------------------'
    ```

b. On line 27, you will see that we declare a vector<int> named vint, initialized with four elements, {2, 4, 6, 8}.

c. On line 30, you will see that we use my vis::print function to print vint's contents after initialization.

d. On line 33, you will see that we are initializing a named object half_sum of type int with the return value from vint_half_sum(vint).

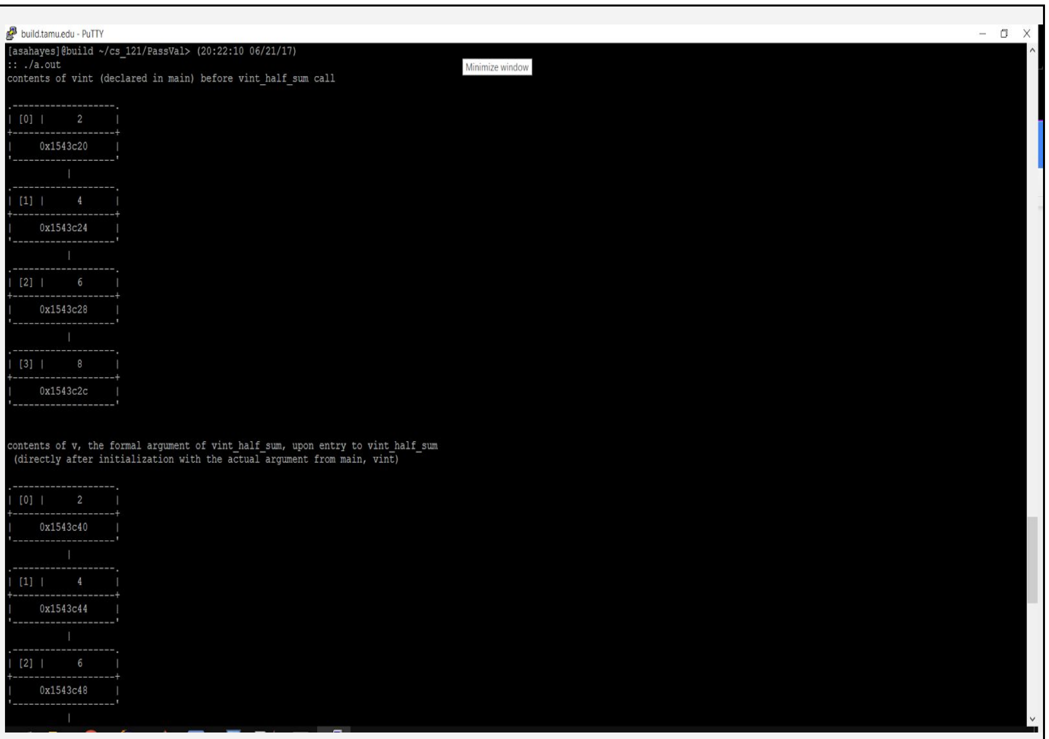
        i. Note: the value that we will use to initialize `half_sum` will be computed by that function call.
    e. Jumping down to the definition of `vint_half_sum` on line 51, we see that when `vint` is provided to `vint_half_sum` as an actual argument. The corresponding formal argument `vector<int> v` will be initialized as an element-wise copy of `vint`.
        i. Here, we can essentially view this process as the occurrence of an implicit `vector<int> v = vint;` statement that is evaluated directly before the execution of the function body of `vint_half_sum.`.
    f. Therefore, from line 33, the formal argument of the function is initialized with the actual argument, and then the execution of the program continues from function body of `vint_half_sum` on line 53.
        i. On line 54, we print `v` using `vis::print`.
        ii. From 55-62, we calculate the "half sum" of `v`.
        iii. On line 65, we use `vis::print` to display the elements of `v` directly before returning the half-sum from the function.
        iv. The return statement on line 66 returns a copy of the value stored in `sum`, which is used to initialize `half_sum`, the int object declared on line 33 in our main function.
    g. On line 37, we print `vint` again using `vis::print` before returning from the program to the operating system.
7. Now that you understand what's going on, compile the code and run it.
8. Carefully, observe the output printed to the screen and then answer the following questions in the fields provided:
    a. Explain why the modification of `v` in `vint_half_sum` does not mutate the actual argument `vint:`

V is separate from vint, in that it is a copy passed down from when vint is input into vint_half_sum. The function makes a copy of vint as to not interfere with anything vint needs to do while in the main function.

    b. What information included in the output produced by the calls to `vis::print` in `main` with `vint` and in `vint_half_sum` supports your response to 8a?

Between the two, the values contained are the same, but the shown memory addresses are different, demonstrating that they are two different objects instead of one modified object.

c. Capture the output written to the terminal window by this program in the form of a screenshot; if you cannot include everything, that's okay. Drag and drop or paste your screenshot to the box below:



```
[asahayes]@build ~/cs_121/PassVal> (20:22:10 06/21/17)
:: ./a.out
contents of vint (declared in main) before vint_half_sum call

.-------------------.
| [0] |      2      |
+-------------------+
|     0x1543c20     |
'-------------------'
          |
.-------------------.
| [1] |      4      |
+-------------------+
|     0x1543c24     |
'-------------------'
          |
.-------------------.
| [2] |      6      |
+-------------------+
|     0x1543c28     |
'-------------------'
          |
.-------------------.
| [3] |      8      |
+-------------------+
|     0x1543c2c     |
'-------------------'


contents of v, the formal argument of vint_half_sum, upon entry to vint_half_sum
 (directly after initialization with the actual argument from main, vint)

.-------------------.
| [0] |      2      |
+-------------------+
|     0x1543c40     |
'-------------------'
          |
.-------------------.
| [1] |      4      |
+-------------------+
|     0x1543c44     |
'-------------------'
          |
.-------------------.
| [2] |      6      |
+-------------------+
|     0x1543c48     |
'-------------------'
          |
```

d. Was the `vis::print` function that I wrote helpful to you?

It allowed for easy access/visualization of the memory addresses, which enabled me to tell the two objects apart.

Submission

- Save this completed labwork as a PDF [File -> Download As -> PDF Document (.pdf)] and submit to gradescope for grading.