CSC 101 Lecture Notes Week 6

More on Arrays and Strings

I. Selected Topics for Program 3

A. The `typedef` declaration.

1. Used for mnemonic type naming.

2. Handy for complicated types, like 2D array.

The `typedef` declaration, cont'd

3. E.g., from `cards.h`:

```
typedef char Deck[DECK_SIZE][CARD_STR_LEN];
```

The `typedef` declaration, cont'd

4. We use `Deck` anywhere we want to define a variable or parameter that's a deck of cards.

The `typedef` declaration, cont'd

5. For example:

```
void shuffle(
        Deck unshuffled_deck,
        Deck shuffled_deck
);
```

The `typedef` declaration, cont'd

Instead of the much bulkier version

```
void shuffle(
  char unshuffled_deck[DECK_SIZE][CARD_STR_LEN],
  char shuffled_deck[DECK_SIZE][CARD_STR_LEN]
);
```

The `typedef` declaration, cont'd

6. Discussed further in Ch 7

Selected Topics for Program 3, cont'd

B. Subdividing a program into `.c` and `.h` files.

1. These are *program files* and *header files*
2. They have extensions "`.h`" and "`.c`".
3. Standard practice for C programs.

Using .c and .h files, cont'd

4. The header files most typically contain type definitions, and function prototypes.

5. The `.c` program files implement the functions.

II. Memory Pictures of Functions with Array Parameters

A. Recall the `read_values` function.

```
int read_values(
    double data[], int max);
```

B. The code is in

`101/examples/arrays/stats-loops-arrays-functions.c`

Memory Pictures of Functions, cont'd

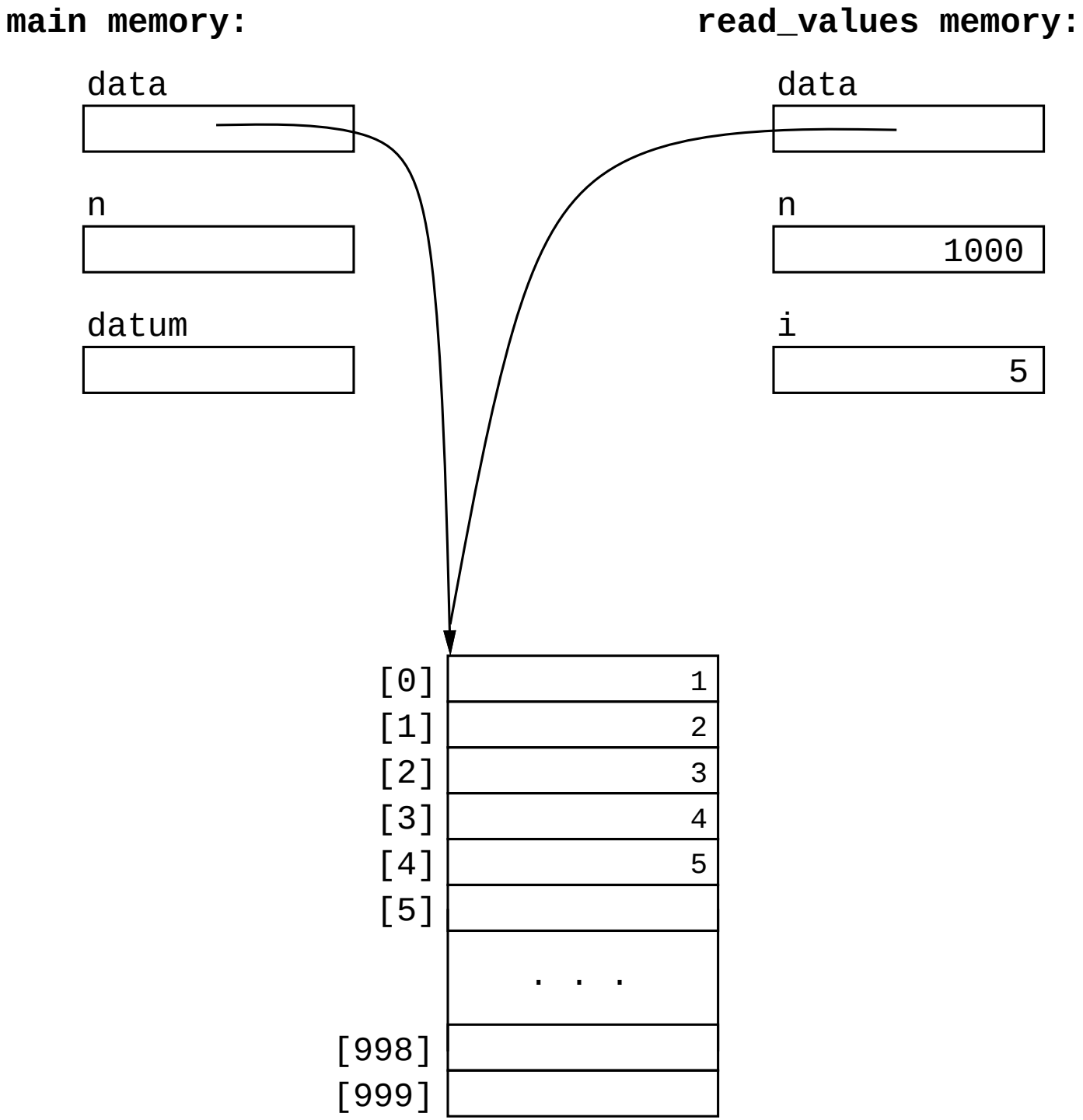
C. Code illustrates array parameters as both ***inputs*** and ***outputs***.

1. `read_values` has array *output* parameter
2. `compute_` functions have array *inputs*
3. In other examples, we'll see array parameters for both input and output.

Memory Pictures of Functions, cont'd

D. Array parameters work as outputs because they're ***pointers*** to shared memory.

Here's a picture:


main memory:
read_values memory:
data
n
datum
data
n
1000
i
5
[0]
1
[1]
2
[2]
3
[3]
4
[4]
5
[5]
. . .
[998]
[999]

III. Precise Steps of a Function Call

A. We've seen a variety of functions.

B. All follow same steps when called.

C. The steps are:

Precise Steps of a Function Call, cont'd

1. evaluate the actual parameters

Precise Steps of a Function Call, cont'd

1. evaluate the actual parameters
2. assign actual values to formal parameters

Precise Steps of a Function Call, cont'd

1. evaluate the actual parameters
2. assign actual values to formal parameters
3. run the function body

Precise Steps of a Function Call, cont'd

1. evaluate the actual parameters
2. assign actual values to formal parameters
3. run the function body
4. return from the function

IV. More on Reading Values into an Array

A. The stats functions do the right thing.

B. It’s pretty easy not to.

C. We’ll now look a bit closer.

More on Reading Values into an Array, cont'd

D. There's a very simple example in

`101/examples/arrays/store.c`

IV. Bounds Checking Arrays

A. `store.c` does NO bounds checking.

IV. Bounds Checking Arrays

A. `store.c` does `!No bounds checking!`

B. I.e., does not check that array is big enough.

IV. Bounds Checking Arrays

A. `store.c` does `!No bounds checking!`

B. I.e., does not check that array is big enough.

C. Fundamentally bad things can happen.

IV. **Bounds Checking Arrays**

A. `store.c` does `!No bounds checking!`

B. I.e., does not check that array is big enough.

C. Fundamentally bad things can happen.

D. Suppose a user gave 6 input values.

IV. **Bounds Checking Arrays**

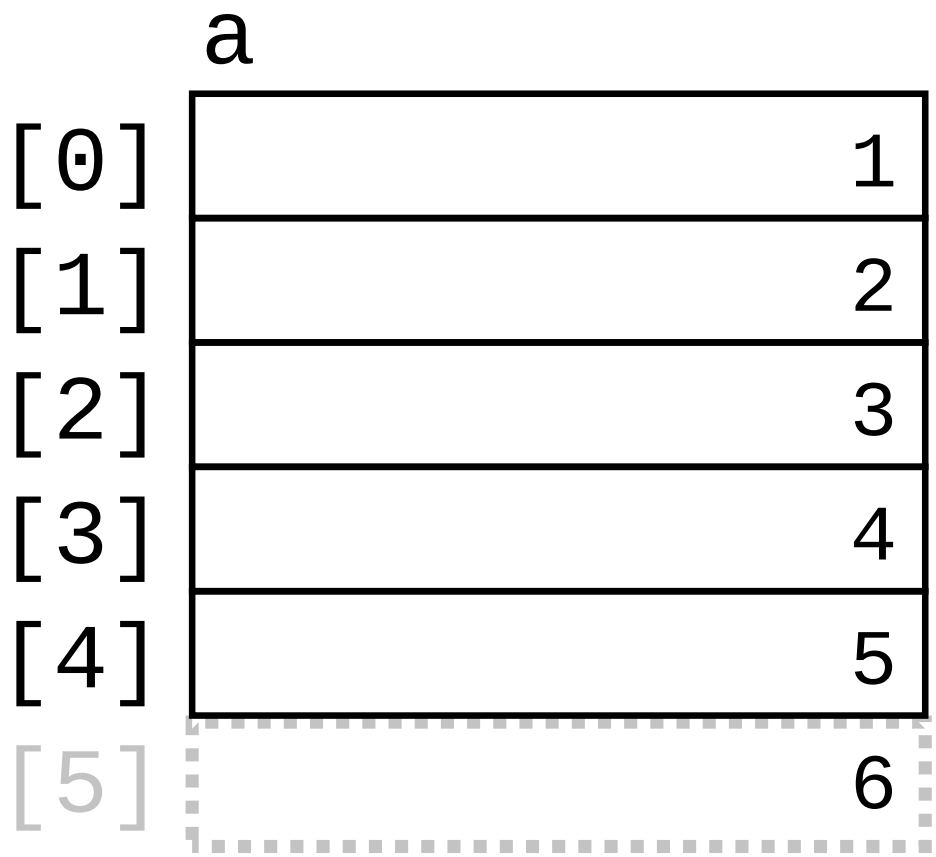
A. `store.c` does `!No bounds checking!`

B. I.e., does not check that array is big enough.

C. Fundamentally bad things can happen.

D. Suppose a user gave 6 input values.

E. Here's a picture of the memory:


a
[0]
1
[1]
2
[2]
3
[3]
4
[4]
5
[5]
6

Bounds Checking, cont'd

F. Fatal error can give these messages:

```
Segmentation violation
Bus error
Illegal instruction
```

1. Exact meaning not important.

2. Messages indicate improper program action.

Bounds Checking, cont'd

Writing past the bounds of an array is a very common C program bug.

Bounds Checking, cont'd

G. C programs should always check bounds.

H. Example in `store-chk.c`.

I. Here's key part of its code:

Bounds Checking, cont'd

```
while (x > 0 && i < 5)
```

versus

```
while (x > 0)
```

V. Some Other Array Examples.

A. On 101 examples page

B. See `101/examples/week5` and `101/examples/week6`