CS1010E Lecture #7

Numeric Arrays

Batch processing

NUS National University of Singapore
Department of Computer Science
School of Computing

Getting to Know Me

Dr. Zhou Lifeng

Senior Lecturer

Office: COM2 #02-56

Email: zhoulifeng@nus.edu.sg

Mid-Semester Review #1

- We have learned 3 control structures
  - Sequence
  - Selection
  - Repetition
- With these, we are able to solve just any computing problem virtually!
- However, writing good programs is more than just learning the syntax:
  - Coding style should be good
  - Logic should be clear
  - Algorithm should be neat

Mid-Semester Review #2

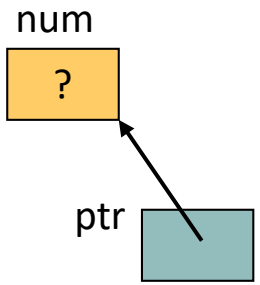
- Pointer is a special type of variable that stores memory address.
  - Define pointer

```
int num;
int *ptr;
```

```
int num, *ptr;
```

  - Assign address to pointer

```
ptr = &num;
```



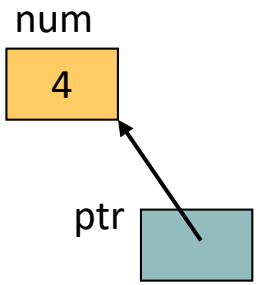
Mid-Semester Review #3

- Pointer is a special type of variable that stores memory address.
  - Assess the value pointer points to

*After address assignment, *ptr is an alias of num*

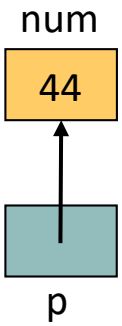
```
int num = 4, *ptr;
ptr = &num;
printf("%d %d", num, *ptr);
```

```
4 4
```



  - Use pointer in function
    - caller passes (copies) address -> callee stores address in pointer

```
int main(void) {
  int num = 4;
  change(&num);
  // ...
}
void change(int *p) { *p = 44; }
```

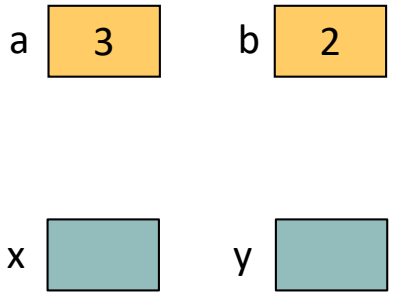


Quiz

- What is printed out by the following C program?

```c
#include <stdio.h>

void f(int x, int y);

int main(void) {

  int a = 3, b = 2;
  a = f(a, b);
  b = f(b, a);

  printf("%d %d\n", a, b);

  return 0;
}
```

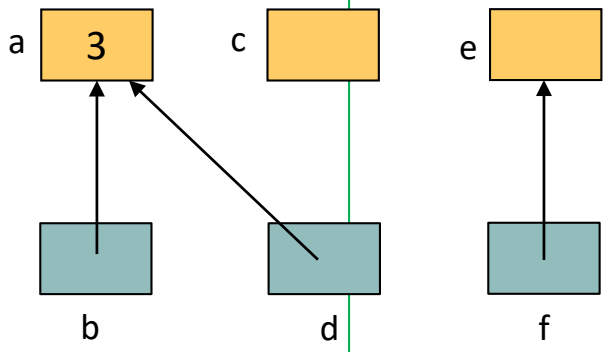
```c
void f(int x, int y) {
  return 3*x – 2*y;
}
```



Quiz

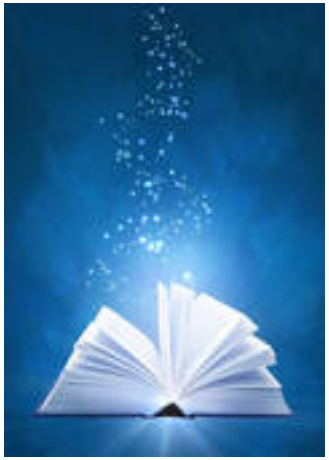
- What is the output of the following program?

```c
#include <stdio.h>

int main(void) {
  int a = 3, c, e;
  int *b, *d, *f;
  b = &a;
  *b = 5;
  c = *b;
  d = b;
  e = *b + c;
  f = &e;
  *f = c;
  a = *f + *b;

  printf("%d %d %d\n", a, c, e);
  printf("%d %d %d\n", *b, *d, *f);
  return 0;
}
```



Learning Objectives

- At the end of this lecture, you should understand:
  - The concept of arrays.
  - How to create and use arrays.
  - How to pass arrays to functions.



Motivating Example : Vote Counting #1

- A student election has just completed with 1000 votes casted for the three candidates: Tom, Dick and Harry.

- Write a program to read in all the votes and display the total number of votes received by each candidate. Each vote has one of the three possible values:
  - 1 for Tom
  - 2 for Dick
  - 3 for Harry

Motivating Example : Vote Counting #2

```c
#include <stdio.h>

int main(void) {

  int i, vote, tom = 0, dick = 0, harry = 0;

  printf("Enter votes:\n");

  for (i = 0; i < 1000; i++) {
    scanf("%d", &vote);
    switch (vote) {
      case 1: tom++;   break;
      case 2: dick++;  break;
      case 3: harry++; break;
    }
  }

  printf("Tom: %d; Dick: %d; Harry: %d\n", tom, dick, harry);
  return 0;
}
```

Q: What if there are 30 instead of 3 candidates?

Motivating Example : Vote Counting #3

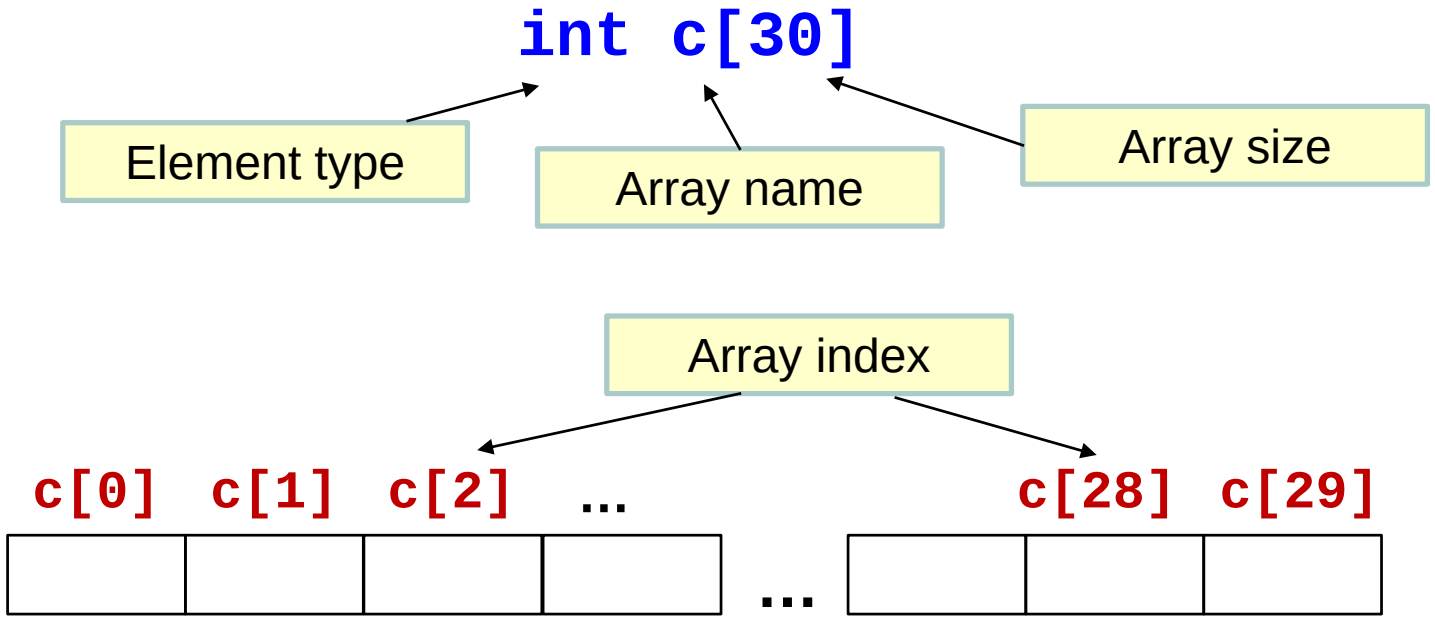
```
#include <stdio.h>

int main(void) {

  int i, vote, c1 = 0, c2 = 0, ..., c30 = 0;

  printf("Enter votes:\n");

  for (i = 0; i < 1000; i++) {
    scanf("%d", &vote);
    switch (vote) {
      case 1:  c1++;  break;
      case 2:  c2++;  break;
      . . .
      case 30: c30++; break;
    }
  }

  . . .
}
```

Q: Can we do it better?

Introducing Array (1/2)

- In the vote counting example, candidates are indexed from 1 to 30.
  - Hence we define and use a set of variables: c1, c2, ..., c30.
  - It's too lame!
- Let's study a new language feature called ARRAY to index candidates $C_0, C_1, \ldots, C_{29}$.

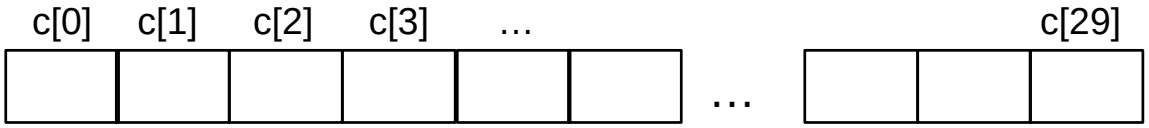
Introducing Array (2/2)



- Array index start from 0.
- **c[0]**, **c[1]**, …, **c[29]** are all integer variables.
- Array provides a way to batch declare variables.

Vote Counting using Array (1/3)

```
int c[30]
```



- For the previous vote counting problem
  - c[0] will store the number of votes for 1st candidate
  - c[1] holds the number of votes for 2nd candidate
  - …
  - c[29] for the 30th candidate.
- If we read in one more vote for candidate 3, we should increase c[2] by 1.

Vote Counting using Array (2/3)

- Pseudo-code for vote counting:

Let *cand* be an array such that $cand_i$ stores the vote count of candidate *i+1*

Initialize *cand[0]* to *cand[29]* to *0*

for every vote read in
    cand[vote-1] ++

Print *cand[0]* to *cand[29]*

Vote Counting using Array (3/3)

```
#include <stdio.h>

int main(void) {

  int i, vote, cand[30];
  for (i = 0; i < 30; i++) {
    cand[i] = 0;
  }

  printf("Enter votes:\n");
  for (i = 0; i < 1000; i++) { // 1000 votes
    scanf("%d", &vote);
    cand[vote-1]++;
  }

  for (i = 0; i < 30; i++) {  // 30 candidates
    printf("Candidate %d: %d\n", i+1, cand[i]);
  }
  return 0;
}
```

```
(data input skipped ...)
Candidate 1: 2
Candidate 2: 4
...
```

Initialize all array elements to 0

Array Declaration : Syntax

SYNTAX	<data type> <array name> [<size>];

- Examples:

```
#define M 5
#define N 10

int arr[10];        // an array of 10 elements
double foo[M*N+8]; // size of array foo is 58
```

Variable-length Array (Disallowed!)

- The following code fragment defines a variable-length array.

```
int i;
scanf("%d", &i);
double bar[i];      // disallowed!
```

- Variable-length array is not supported by ANSI C (which we stick to).
  - **`gcc -pedantic`** gives compilation warning.

- In ANSI C, array is of fixed size and its size is determined at compile time.

Array Maximum Size and Actual Usage

- For problems using arrays, we will indicate the maximum number of data in an array so that you may define array size accordingly.
  - Declared array may be partially used in a given sample run; wastage of the rest array slots usually is not a big concern.
- Example:

```
#define MAX_SIZE 100    // maximum number of data

int main(void) {

  int i, size, data[MAX_SIZE];

  printf("Enter the number of data: ");
  scanf("%d", &size);
  printf("Enter %d data one by one: ", size);
  ...
}
```

Declare array according to the maximum number of data

The actual number of data in a particular sample run

Array Declarations with Initializers

- Array can be initialized with a loop.
- Array can be initialized at the same time of declaration using initializer.

```
int a[3] = {54, 9, 10}; // a[0]=54, a[1]=9, a[2]=10

int b[] = {1, 2, 3};
// size of array b is 3 with b[0]=1, b[1]=2, b[2]=3

int c[5] = {17, 3, 10}; // partial initialization
// c[0]=17, c[1]=3, c[2]=10, c[3]=0, c[4]=0
```

- The following initialization is incorrect:

```
int f[5];
f[5] = {8, 23, 12, -3, 6}; // too late to do this;
                           // compilation error!
```

Vote Counting with Array Initializer

- Now we modify the program to use array initializer.

```
#include <stdio.h>

int main(void) {

  int i, vote, cand[30];
  for (i = 0; i < 30; i++) { cand[i] = 0; }
  int cand[30] = {0};

  printf("Enter votes:\n");
  for (i = 0; i < 1000; i++) {
    scanf("%d", &vote);
    cand[vote-1]++;
  }

  for (i = 0; i < 30; i++) {
    printf("Candidate %d: %d\n", i+1, cand[i]);
  }
  return 0;
}
```

Quiz

- What is printed out by the following C code fragment?

```
int a[] = {2, 5, 4, 1}, i, x = 0;
for (i = 0; i < 4; i++) {
  if (a[i]%3 == 1) {
    break;
  }
  x += a[i];
}
printf("%d\n", x);
```

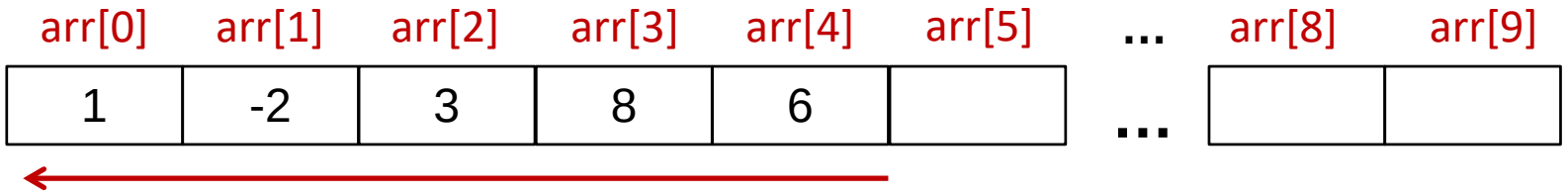
Demo : Reversely Print

This is Problem Set 3
Ex #01 on CodeCrunch

- Write a program reverse_print.c to read a list of numbers (at most 10 of them) into the array, reversely print out the input numbers, each followed by a space.
- Sample run:

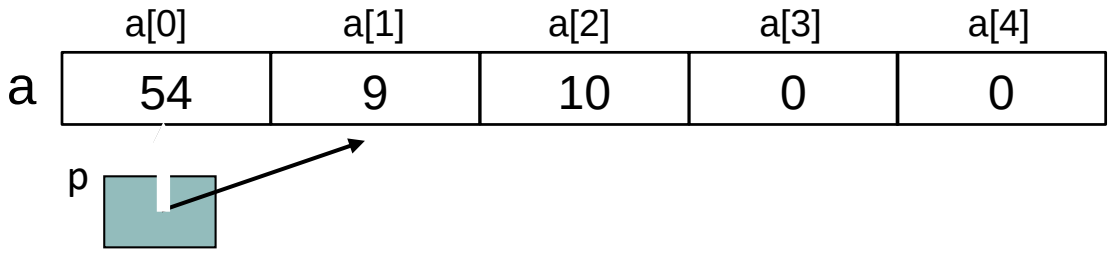
```
Enter the number of integers: 5
Enter 5 integers: 1 -2 3 8 6
Reverse printing: 6 8 3 -2 1
```

- Algorithm?



Arrays and Pointers

- When the array name **`arr`** appears alone in an expression, it means the address of the first element (i.e. **`&arr[0]`**) of that array.



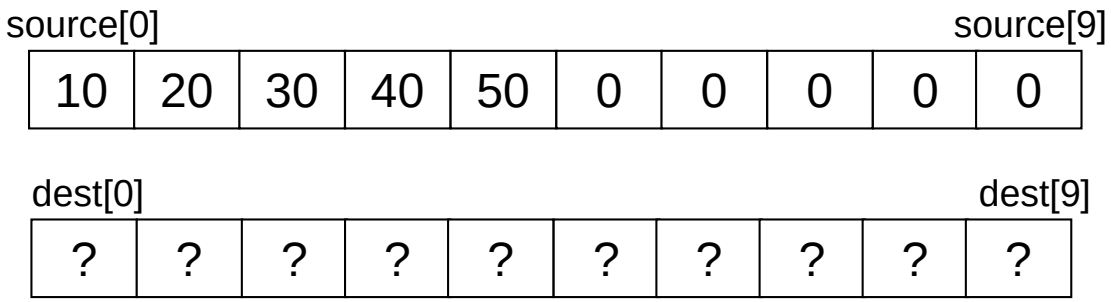

```
int arr[5] = {54, 9, 10}, *p;
p = arr;  // the same as p = &arr[0];
p++;    // make pointer point to the next slot
printf("%d\n", arr[1]);
printf("%d\n", *p);
printf("%p\n", arr);
printf("%p\n", &arr[0]);
printf("%p\n", p);
```

```
9
9
ffbff724
ffbff724
ffbff728
```

Common Error (1/2)

- In C, array cannot be assigned.
- Therefore the following is illegal in C:

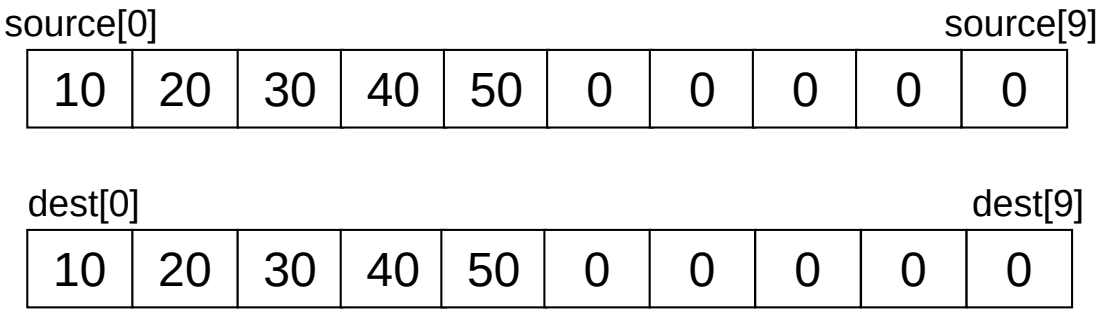
```
int source[10] = { 10, 20, 30, 40, 50 };
int dest[10];
dest = source;  // compilation error!
```



Common Error (2/2)

- To copy values from one array to another:
  - Method 1: use a loop

```
for (i = 0; i < 10; i++) {
  dest[i] = source[i];
}
```



  - Method 2: use C library function **`memcpy()`**
    - Out of the scope of CS1010E

Passing Arrays to Functions (1/5)

```c
#include <stdio.h>
int sum_array(int arr[], int size);

int main(void) {

  int foo[8] = {5, 3, 7, 1, -4, 2};
  printf("sum is %d\n", sum_array(foo, 8));
  printf("sum is %d\n", sum_array(foo, 3));
  return 0;
}

// sum up first 'size' elements in the given array
int sum_array(int arr[], int size) {

  int i, total = 0;
  for (i = 0; i < size; i++) {
    total += arr[i];
  }
  return total;
}
```

Caller specifies array to pass

Q: What is the output?

```
sum is 14
sum is 15
```

*Give **the same array** a new name*

Q: How about this function call?
`sum_array(foo, 9)`

Passing Arrays to Functions (2/5)

- **Caution** 
  - When passing a value representing the number of array elements to be processed, that value must not exceed the actual array size.

```
printf("sum is %d\n", sum_array(foo, 9));
```

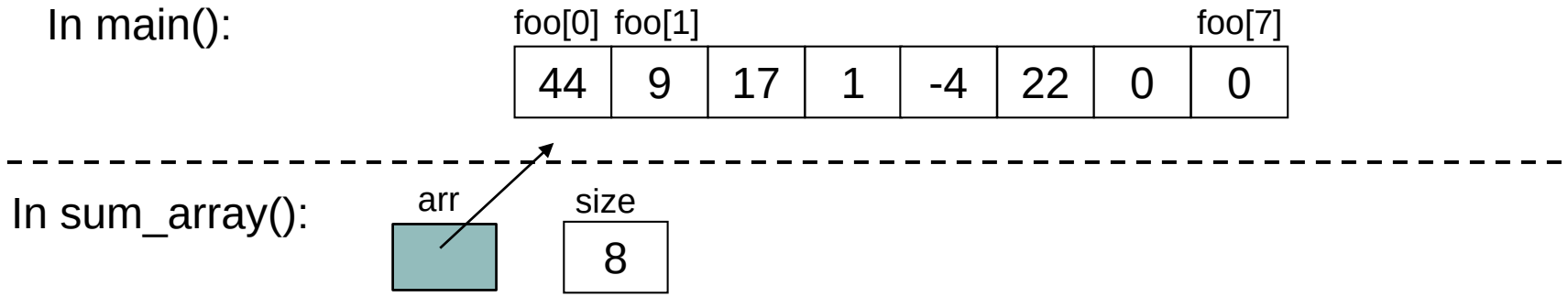
Compiler won't detect this "error".

  - There is NO boundary checking by the compiler.
  - Program execution may produce runtime error such as "segmentation fault".

Passing Arrays to Functions (3/5)

```
int main(void) {
    ...
    printf("sum is %d\n", sum_array(foo, 8));
    ...
}

int sum_array(int arr[], int size) {
    ...
}
```

- Recall that array name is the address of its first element.



Passing Arrays to Functions (4/5)

- Alternative syntax
  - The following shows the alternative syntax for array parameter in function header (and prototype).

```
int sum_array(int *arr, int size) {
```

  - However, we recommend the [ ] notation

```
int sum_array(int arr[], int size) {
```

  - There is also no need to put array size inside [ ].

```
int sum_array(int arr[8], int size) {
```

Ignored by compiler

Actual number of elements you want to process

Passing Arrays to Functions (5/5)

- In C, all parameters are passed by value.
  - In function calls, values are copied from actual parameters to formal parameters.

- When passing an array to a function, the memory address of the first element of the array is copied to that function.
  - Any modification of the elements in the function will affect the actual array.

Quiz

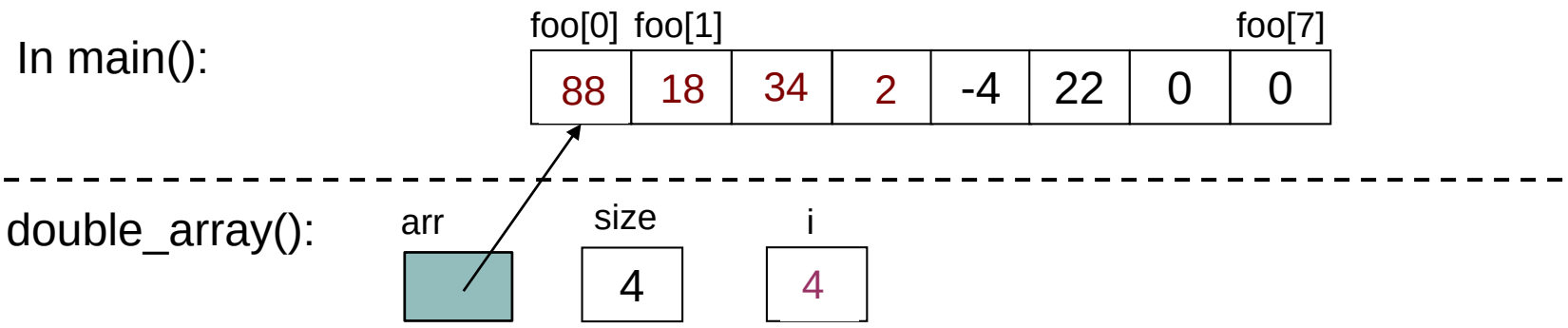
```
// preprocessor directives and function prototypes omitted
int main(void) {
  int foo[8] = {44, 9, 17, 1, -4, 22};
  double_array(foo, 4);
  print_array(foo, 8);
  return 0;
}

void double_array(int arr[], int size) {
  int i;
  for (i = 0; i < size; i++) {
    arr[i] *= 2;
  }
}

void print_array(int arr[], int size) {
  int i;
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
}
```

Q: What is the output?

Exercise #2 : Memory Illustration

- Array may be modified by callee function.

```
int main(void) {
  int foo[8] = {44, 9, 17, 1, -4, 22};
  double_array(foo, 4);
    . . .
}
void double_array(int arr[], int size) {
  int i;
  for (i = 0; i < size; i++) {
    arr[i] *= 2;
  }
}
```



Today's Summary

Arrays I

Arrays

- Array creation : syntax
- Array initializer
- Using array for problem solving
- Relationship between array and pointer
- Passing arrays to functions