

CS1010

<http://www.comp.nus.edu.sg/~cs1010/>

Programming Methodology

UNIT 8

Arrays



NUS
National University
of Singapore

School of
Computing

UNIT 8: Arrays

Objectives:

- Understand the concept and application of arrays
- Problem solving using arrays

Reference:

- Chapter 7: Array Pointers
 - Sections 7.1 – 7.5

UNIT 8: Arrays (1/2)

1. Motivation #1: Coin Change
2. Motivation #2: Vote Counting
3. Arrays
 - 3.1 Array Declaration: Syntax
 - 3.2 Array Variable
 - 3.3 Array Declarations with Initializers
 - 3.4 Demo #1: Using Array Initializer
 - 3.5 Demo #2: Coin Change Revisit
 - 3.6 Array Size

UNIT 8: Arrays (2/2)

4. Arrays and Pointers
5. Array Assignment
6. Array Parameters in Functions
7. Passing Array Arguments
8. Standard I/O Functions for Arrays
9. Modifying Array Arguments
10. Exercise: Up-slopes

1. Motivation #1: Coin Change (1/2)

- Some of the programs we have written are “long-winded”, because we have not learned enough C constructs to do it simpler.
- Consider the **Coin Change** problem (Week 1 Task 2) with 6 denominations 1¢, 5¢, 10¢, 20¢, 50¢, and \$1:

Algorithm 1:

```
input: amt (in cents); output: coins
coins ← 0
coins += amt/100; amt %=100;
coins += amt/50; amt %= 50;
coins += amt/20; amt %= 20;
coins += amt/10; amt %= 10;
coins += amt/5; amt %= 5;
coins += amt/1; amt %= 1;
print coins
```



1. Motivation #1: Coin Change (2/2)

```
int minimumCoins(int amt) { Unit8_CoinChange.c
    int coins = 0;

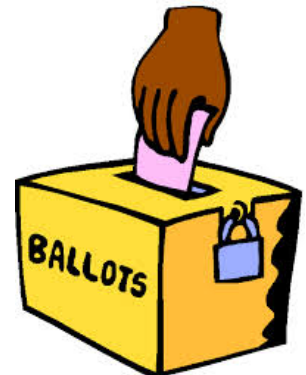
    coins += amt/100;
    amt %= 100;
    coins += amt/50;
    amt %= 50;
    coins += amt/20;
    amt %= 20;
    coins += amt/10;
    amt %= 10;
    coins += amt/5;
    amt %= 5;
    coins += amt/1; // retained for regularity
    amt %= 1;      // retained for regularity

    return coins;
}
```

- Can we do better?

2. Motivation #2: Vote Counting

- A student election has just completed with 1000 votes cast for the three candidates: Tom, Dick and Harry.
- Write a program [Unit8_VoteCount.c](#) to read in all the votes and display the total number of votes received by each candidate. Each vote has one of three possible values:
 - 1: for candidate Tom
 - 2: for candidate Dick
 - 3: for candidate Harry



2. Motivation #2: Votes for 3 candidates

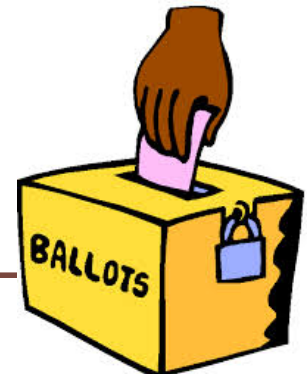
Unit8_VoteCount.c

```
#include <stdio.h>
#define NUM_VOTES 1000 // number of votes

int main(void) {
    int i, vote, tom = 0, dick = 0, harry = 0;

    printf("Enter votes:\n");
    for (i = 0; i < NUM_VOTES; 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;
}
```

What if there were 30
instead of 3 candidates?

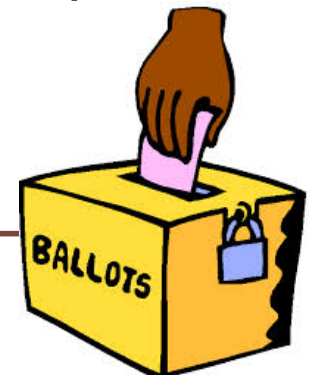


2. Motivation #2: Votes for 30 candidates

```
#include <stdio.h>
#define NUM_VOTES 1000    // number of votes

int main(void) {
    int i, vote, c1 = 0, c2 = 0, ..., c30 = 0;

    printf("Enter votes:\n");
    for (i = 0; i < NUM_VOTES; i++) {
        scanf("%d", &vote);
        switch (vote) {
            case 1: c1++; break;
            case 2: c2++; break;
            . . .
            case 30: c30++; break;
        }
    }
    . . .
}
```



3. Introducing Array (1/4)

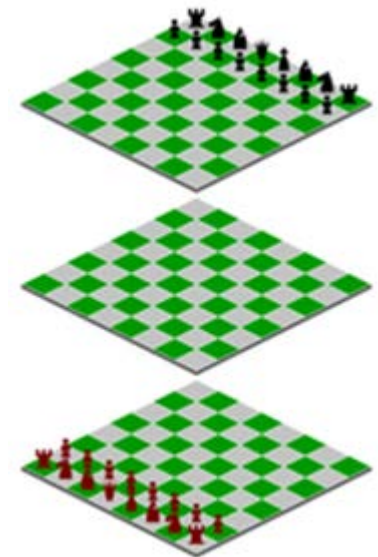
**“If a program manipulates a large amount of data,
it does so in a small number of ways.”**

Alan J. Perlis
Yale University

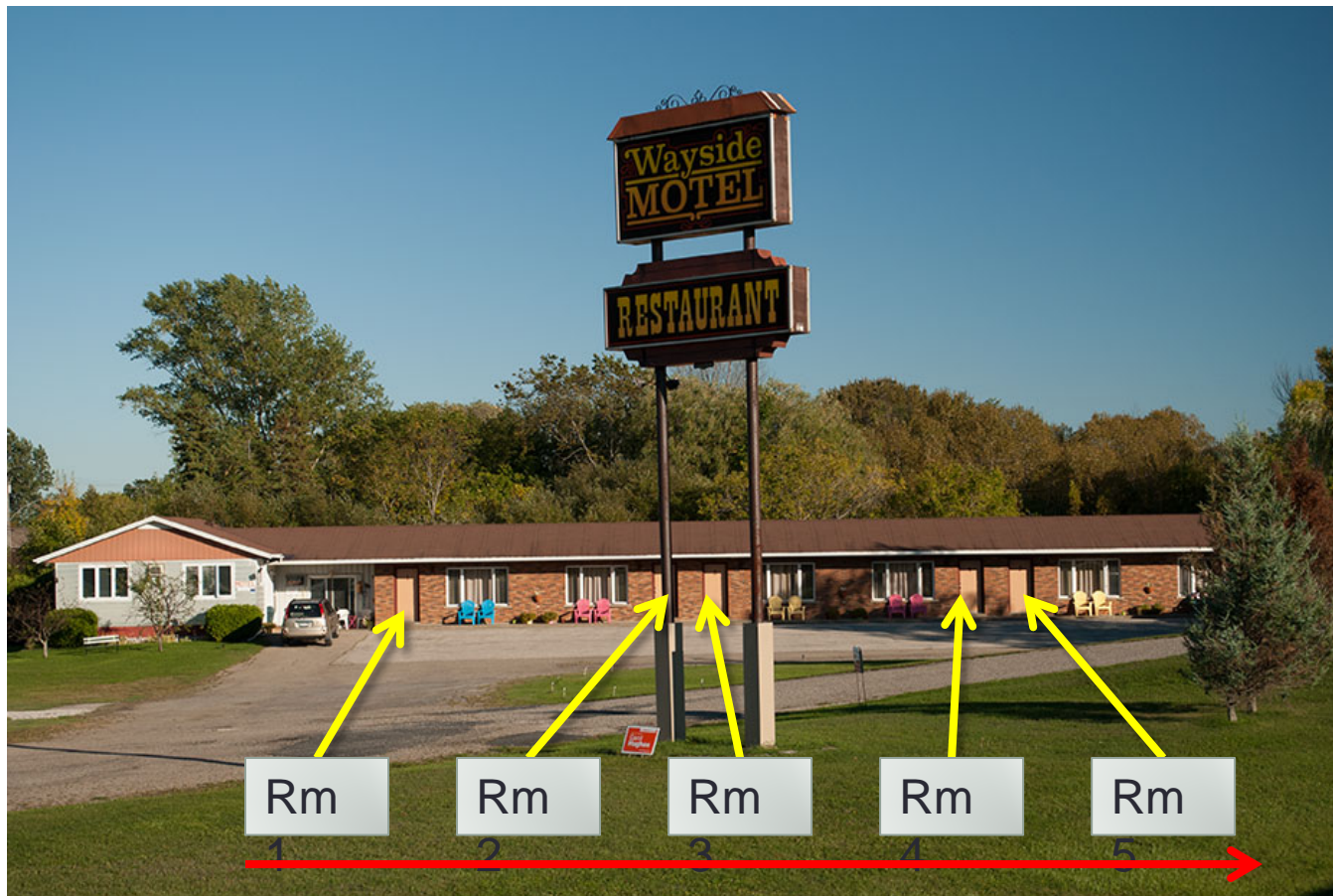
The first recipient of ACM Turing Award

$$\begin{bmatrix} 2 & 1 & 3 \\ -2 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 1 \\ 3 & 2 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 10 \\ 0 & 4 \end{bmatrix}$$

*	2	1			1	*	1
2	*	2	1	1	1	1	1
1	2	3	*	2	1	1	1
1	2	*	3	*	1	1	*
1	*	2	2	1	2	2	2
1	1	1			1	*	1
			1	1	2	1	1
			1	*	1		



Motel in US



One Dimension

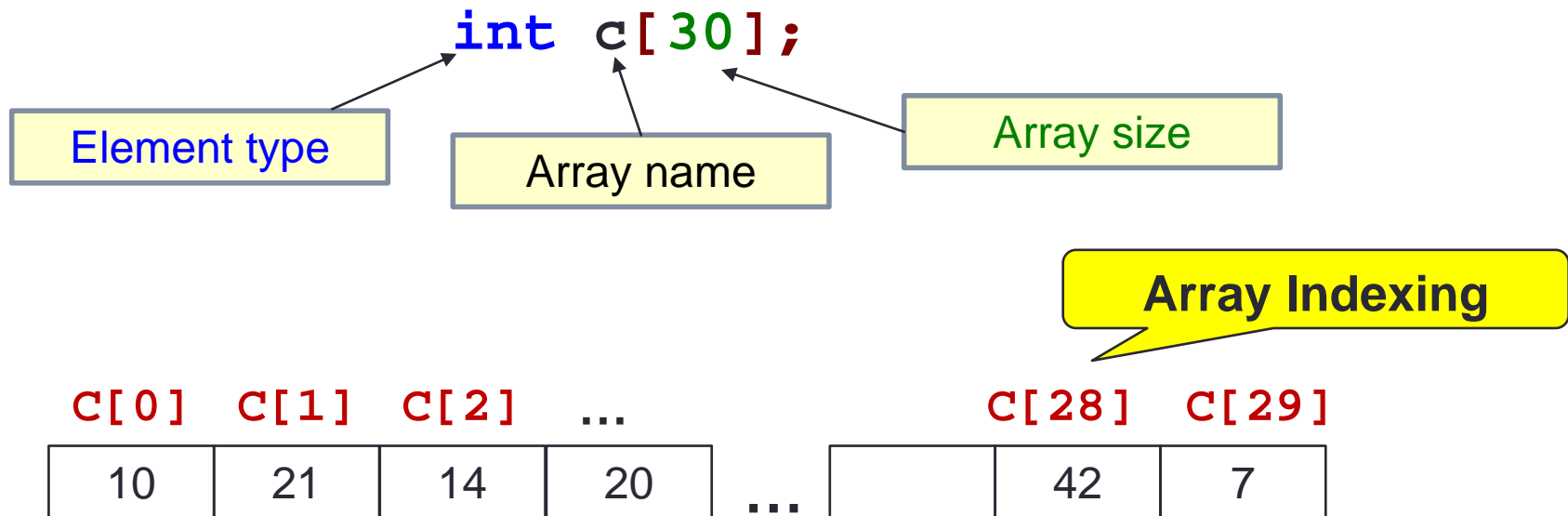
3. Introducing Array (2/4)

- In the vote counting problem, it's inconvenient to define and use a set of variables c_1, c_2, \dots, c_{30} , each for one candidate.
- We handle this problem by indexing the candidates:
 C_0, C_1, \dots, C_{29} (assuming that indexing begins at 0)

	C_0	C_1	C_2		...						C_{28}	C_{29}
Votes for Candidates:	10	21	14	20			...				42	7

3. Introducing Array (3/4)

- The indexing facility is implemented as a programming language feature called **ARRAY**.



3. Introducing Array (4/4)

Pseudo-code for Vote Counting:

Let C be an array such that C_i holds the vote count of Candidate i

for each i such that $0 \leq i < \text{Number_of_candidates}$
 $C_i \leftarrow 0$;

while there is a vote to be counted

$vote \leftarrow$ read a vote

if $1 \leq vote \leq \text{Number_of_candidate}$ then

$j \leftarrow vote - 1$

$C_j \leftarrow C_j + 1$

3.1 Array Declaration: Syntax

T arrname [**E**]

- **arrname** is name/identifier of array (same way you would name a variable)
- T is a data type (e.g., **int**, **double**, **char**, ...)
- **E** is an integer constant expression with a positive value
- Examples:

```
int arr[10]; // size of arr is 10
```

```
#define M 5  
#define N 10  
double foo[M*N+8]; // size of foo is 58
```

```
int i;  
float bar[i]; // variable-length array
```

Not encouraged to use variable-length arrays.
Not supported by ISO C90 standard.
gcc -pedantic will generate warning.

Note

For problem using arrays, we will state the maximum number of elements so there is no need for variable-length arrays.

3.2 Array Variable (1/4)

- In an array of type **T**, each element is a type **T** variable.

Unit8_VoteCountArray.c

```
#define NUM_VOTES 1000
#define NUM_CANDIDATES 30
int main(void) {
    int i, vote, cand[NUM_CANDIDATES];
    for (i = 0; i < NUM_CANDIDATES; i++) // initialize array
        cand[i] = 0;

    printf("Enter votes:\n");
    for (i = 0; i < NUM_VOTES; i++) {
        scanf("%d", &vote);
        cand[vote-1]++;
    }
    for (i = 0; i < NUM_CANDIDATES; i++) {
        printf("candidate %d: total %d, %.2f%%\n",
            i+1, cand[i], (cand[i] * 100.0)/
    }
    return 0;
}
```

What is
%%?

Why 100.0 and
not 100?

3.2 Array Variable (2/4)

- In an array of type **T**, each element is a type **T** variable.

Unit8_VoteCountArray.c

```
#define NUM_VOTES 1000
#define NUM_CANDIDATES 30
int main(void) {
    int i, vote, cand[NUM_CANDIDATES];
    for (i = 0; i < NUM_CANDIDATES; i++) { cand[i] = 0; }

    printf("Enter votes:\n");
    for (i = 0; i < NUM_VOTES; i++) {
        scanf("%d", &vote);
        cand[vote-1]++;
    }
    for (i = 0; i < NUM_CANDIDATES; i++) {
        printf("candidate %d: total %d, %.2f%%\n",
            i+1, cand[i], (cand[i] * 100.0)/NUM_VOTES);
    }
    return 0;
}
```

$0 \leq \text{index} < \text{array size}$

3.2 Array Variable (3/4)

- In an array of type **T**, each element is a type **T** variable.

Unit8_VoteCountArray.c

```
#define NUM_VOTES 1000
#define NUM_CANDIDATES 30
int main(void) {
    int i, vote, cand[NUM_CANDIDATES];
    for (i = 0; i < NUM_CANDIDATES; i++) { cand[i] = 0; }

    printf("Enter votes:\n");
    for (i = 0; i < NUM_VOTES; i++) {
        scanf("%d", &vote);
        cand[vote-1]++;
    }
    for (i = 0; i < NUM_CANDIDATES; i++) {
        printf("candidate %d: total %d, %.2f%%\n",
            i+1, cand[i], (cand[i] * 100.0)/NUM_VOTES);
    }
    return 0;
}
```

Increment the value
of an array element

Assume no invalid vote

3.2 Array Variable (4/4)

- In an array of type **T**, each element is a type **T** variable.

Unit8_VoteCountArray.c

```
#define NUM_VOTES 1000
#define NUM_CANDIDATES 30
int main(void) {
    int i, vote, cand[NUM_CANDIDATES];
    for (i = 0; i < NUM_CANDIDATES; i++) { cand[i] = 0; }

    printf("Enter votes:\n");
    for (i = 0; i < NUM_VOTES; i++) {
        scanf("%d", &vote);
        cand[vote-1]++;
    }
    for (i = 0; i < NUM_CANDIDATES; i++) {
        printf("candidate %d: total %d, %.2f%%\n",
            i+1, cand[i], (cand[i] * 100.0)/NUM_VOTES);
    }
    return 0;
}
```

Print out all vote counts

3.3 Array Declarations with Initializers

- Array variables can be **initialized** at the time of declaration.

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

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

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

- The following initializations are **incorrect**:

```
int e[2] = {1, 2, 3}; // warning issued: excess elements

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



3.4 Demo #1: Using Array Initializer

- Modify the program to use an **array initializer**.

Unit8_VoteCountArrayVer2.c

```
#define NUM_VOTES 1000
#define NUM_CANDIDATES 30
int main(void) {
    int i, vote, cand[NUM_CANDIDATES];
for (i = 0; i < NUM_CANDIDATES; i++) { cand[i] = 0; }
    int cand[NUM_CANDIDATES] = { 0 };

    printf("Enter votes:\n");
    for (i = 0; i < NUM_VOTES; i++) {
        scanf("%d", &vote);
        cand[vote-1]++;
    }
    for (i = 0; i < NUM_CANDIDATES; i++) {
        printf("candidate %d: total %d, %.2f%%\n",
            i+1, cand[i], (cand[i] * 100.0)/NUM_VOTES);
    }
    return 0;
}
```

3.5 Demo #2: Coin Change Revisit (1/2)

- Let's "roll" the common steps in Algorithm 1 into a loop:

Algorithm 1:

```
input: amt (in cents); output: coins
coins ← 0
coins += amt/100; amt %=100;
coins += amt/50; amt %= 50;
coins += amt/20; amt %= 20;
coins += amt/10; amt %= 10;
coins += amt/5; amt %= 5;
coins += amt/1; amt %= 1;
print coins
```

Algorithm 2:

```
input: amt (in cents); output: coins
coins ← 0
From the largest denomination to the smallest:
  coins += amt/denomination
  amt %= denomination
  go to next denomination
print coins
```

Now, we may use a list D to store the denominations – giving rise to an array!

Algorithm 3:

```
input: amt (in cents); output: coins
D is an array with 6 elements
coins ← 0
for i from 0 to 5 // there are 6 denominations
  coins += amt/Di;
  amt %= Di;
print coins
```

3.5 Demo #2: Coin Change Revisit (2/2)

- Compare:

```
int minimumCoins(int amt) {  
    int coins = 0;  
    coins += amt/100;  
    amt %= 100;  
    coins += amt/50;  
    amt %= 50;  
    coins += amt/20;  
    amt %= 20;  
    coins += amt/10;  
    amt %= 10;  
    coins += amt/5;  
    amt %= 5;  
    coins += amt/1;  
    amt %= 1;  
    return coins;  
}
```

Unit8_CoinChange.c

```
int minimumCoins(int amt) {  
    int denoms[] = {100,50,20,10,5,1};  
    int i, coins = 0;  
  
    for (i=0; i<6; i++) {  
        coins += amt/denoms[i];  
        amt %= denoms[i];  
    }  
  
    return coins;  
}
```

Unit8_CoinChangeArray.c

So much more elegant!

3.6 Array Size

- In ANSI C (which we are adopting), the array is of fixed size, and is determined at compile time
 - Hence, we need to specify the size of the array, eg:

```
int arr[8];
```

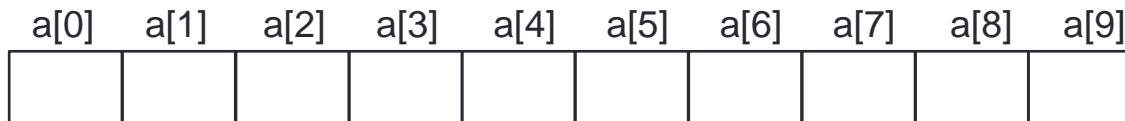
- The following is **not** allowed:

```
int size;  
  
printf("Enter array size: ");  
scanf("%d", &size);  
int arr[size]; // declare array arr with number of  
               // elements provided by user
```

- Hence, for problems on arrays, we will indicate the largest possible size of each array.

4. Arrays and Pointers

- Example: `int a[10]`



- When the array name `a` appears in an expression, it refers to the address of the first element (i.e. `&a[0]`) of that array.

```
int a[3];  
printf("%p\n", a);  
printf("%p\n", &a[0]);  
printf("%p\n", &a[1]);
```

```
ffbff724  
ffbff724  
ffbff728
```

These 2
outputs will
always be
the same.

Output varies from one run to another. Each element is of `int` type, hence takes up 4 bytes (32 bits).

Array a

```
int a[] = {0, 123, 8};
```

`a` is the address of the starting of the array
In this case the value of `a` is `ffbff7d8`

`a` is same as the address of `a[0]`, namely, `&a[0]`.

That's why, `&a[1]` is the address of `a[1]`, in which is 4 bytes away from the address of `a[0]` (or `a`)

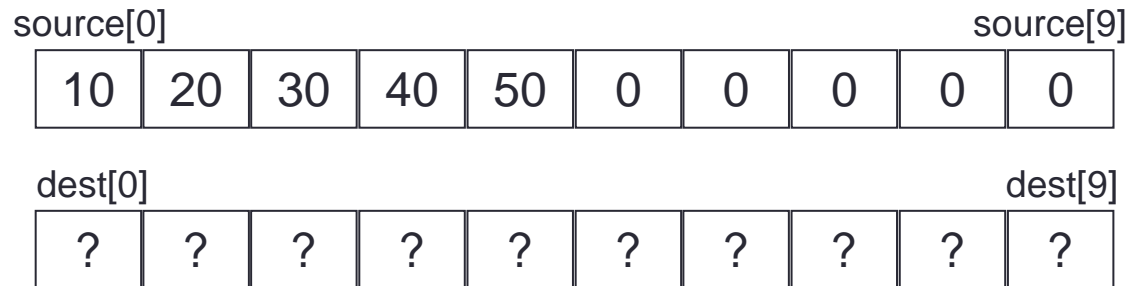
Address	Content	
<code>ffbff724</code>	0	← <code>a[0]</code>
<code>ffbff725</code>		
<code>ffbff726</code>		
<code>ffbff727</code>		
<code>ffbff728</code>	123	← <code>a[1]</code>
<code>ffbff729</code>		
<code>ffbff72a</code>		
<code>ffbff72b</code>		
<code>ffbff72c</code>	8	← <code>a[2]</code>
<code>ffbff72d</code>		
<code>ffbff72e</code>		
<code>ffbff72e</code>		

5. Array Assignment (1/2)

- The following is **illegal** in C:

```
#define N 10
int source[N] = { 10, 20, 30, 40, 50 };
int dest[N];
dest = source; // illegal!
```

Unit8_ArrayAssignment.c



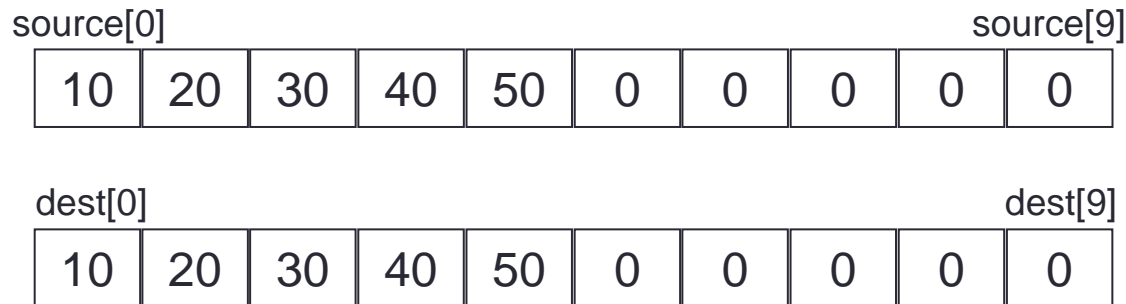
- Reason:
 - An array name is a fixed (constant) pointer; it points to the first element of the array, and this **cannot** be altered.
 - The code above attempts to alter `dest` to make it point elsewhere.

5. Array Assignment (2/2)

- How to do it properly? Write a loop:

```
#define N 10
int source[N] = { 10, 20, 30, 40, 50 };
int dest[N];
int i;
for (i = 0; i < N; i++) {
    dest[i] = source[i];
}
```

Unit8_ArrayCopy.c



- (There is another method – use the `<string.h>` library function `memcpy()`, but this is outside the scope of CS1010.)

6. Array Parameters in Functions (1/3)

Unit8_SumArray.c

```
#include <stdio.h>
int sumArray(int [], int); // function prototype

int main(void) {
    int foo[8] = {44, 9, 17, 1, -4, 22};
    int bar[] = {2, 8, 6};
    printf("sum is %d\n", sumArray(foo, 8));
    printf("sum is %d\n", sumArray(foo, 3));
    printf("sum is %d\n", sumArray(bar, 3));
    return 0;
}

// size of array arr can be unspecified
// need an array size parameter
int sumArray(int arr[], int size) {
    int i, total=0;
    for (i=0; i<size; i++)
        total += arr[i];
    return total;
}
```

```
sum is 89
sum is 70
sum is 16
```

6. Array Parameters in Functions (2/3)

■ Function prototype:

- As mentioned before, name of parameters in a function prototype are optional and ignored by the compiler. Hence, both of the following are acceptable and equivalent:

```
int sumArray(int [], int);
```

```
int sumArray(int arr[], int size);
```

■ Function header:

- No need to put array size inside []; even if array size is present, compiler just ignores it.
- Instead, provide the array size through another parameter.

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

```
int sumArray(int arr[8], int size) { ... }
```

Ignored by compiler

*Actual number of elements
you want to process*

6. Array Parameters in Functions (3/3)

■ Alternative syntax

- The following shows the alternative syntax for array parameter in function prototype and function header (This will be clearer after we cover Pointers later.)

```
int sumArray(int *, int); // fn prototype
```

```
int sumArray(int *arr, int size) { ... }
```

- However, we recommend the `[]` notation


```
int sumArray(int [], int); // fn prototype
```

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

7. Passing Array Arguments (1/3)

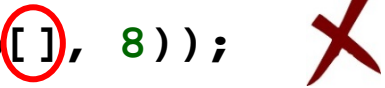
- Need to pass an array argument for **arr** as well as **size**, the number of elements to be processed.

```
int main(void) {  
    ...  
    printf("sum is %d\n", sumArray(foo, 8));  
    ...  
}  
int sumArray(int arr[], int size) {  
    ...  
}
```



- Note that array argument is specified by array name without []

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



Common mistake!



7. Passing Array Arguments (2/3)

■ Caution!

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

```
printf("sum is %d\n", sumArray(foo, 10));
```

Too big!

And compiler won't be able to detect such "error"!
May get "Segmentation Fault (core dumped)"
when you run the program!

7. Passing Array Arguments (3/3)

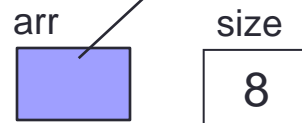
- Recall that the array name is the address of its first element. Hence `foo` means `&foo[0]`.

```
int main(void) {  
    ...  
    printf("sum is %d\n", sumArray(foo, 8));  
    ...  
}  
int sumArray(int arr[], int size) {  
    ...  
}
```

In main():



In sumArray():



8. Standard I/O Functions for Arrays (1/3)

- It might be advisable to write a function to read values into an array, and a function to print values in an array.
- Especially so for the latter, as you probably want to use it to check the values of your array elements at different stages of your program.
- The following illustrates an array `scores` of type `float`.

```
#define SIZE 6

int main(void) {
    float scores[SIZE];
    scanArray(scores, SIZE);
    printArray(scores, SIZE);
    return 0;
}
```

Unit8_ArrayInputOutput.c

8. Standard I/O Functions for Arrays (2/3)

■ Input function:

```
void scanArray(float arr[], int size) {
    int i;
    float value;
    // You may add a prompt for user here
    for (i=0; i<size; i++) {
        scanf("%f", &value);
        arr[i] = value;
    }
}
```

or

```
void scanArray(float arr[], int size) {
    int i;
    // You may add a prompt for user here
    for (i=0; i<size; i++) {
        scanf("%f", &arr[i]);
    }
}
```

8. Standard I/O Functions for Arrays (3/3)

■ Output function:

```
void printArray(float arr[], int size) {  
    int i;  
    // To print all values on one line  
    for (i=0; i<size; i++)  
        printf("%f ", arr[i]);  
  
    printf("\n");  
}
```

or

```
void printArray(float arr[], int size) {  
    int i;  
    // To print each value on one line  
    for (i=0; i<size; i++)  
        printf("%f\n", arr[i]);  
}
```

9. Modifying Array Arguments (1/2)

- Study this program:

Unit8_ModifyArrayArg.c

```
int main(void) {
    int foo[8] = {44, 9, 17, 1, -4, 22};
    doubleArray(foo, 4);
    printArray(foo, 8);
    return 0;
}
// To double the values of array elements
void doubleArray(int arr[], int size) {
    int i;
    for (i=0; i<size; i++)
        arr[i] *= 2;
}
// To print arr
void printArray(int arr[], int size) {
    int i;
    for (i=0; i<size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
```

9. Modifying Array Arguments (2/2)

```
int main(void) {
    int foo[8] = {44, 9, 17, 1, -4, 22};
    doubleArray(foo, 4);
    . . .
}
// To double the values of array elements
void doubleArray(int arr[], int size) {
    int i;
    for (i=0; i<size; i++)
        arr[i] *= 2;
}
```

In main():



In doubleArray():



10. Exercise: Up-slopes (1/3)

You are an avid runner. Given a running route consisting of heights of points at regular interval on the route, you want to find the **number of up-slopes** in the route. An up-slope is a contiguous group of heights of increasing values.



Enter data:

3

3

4

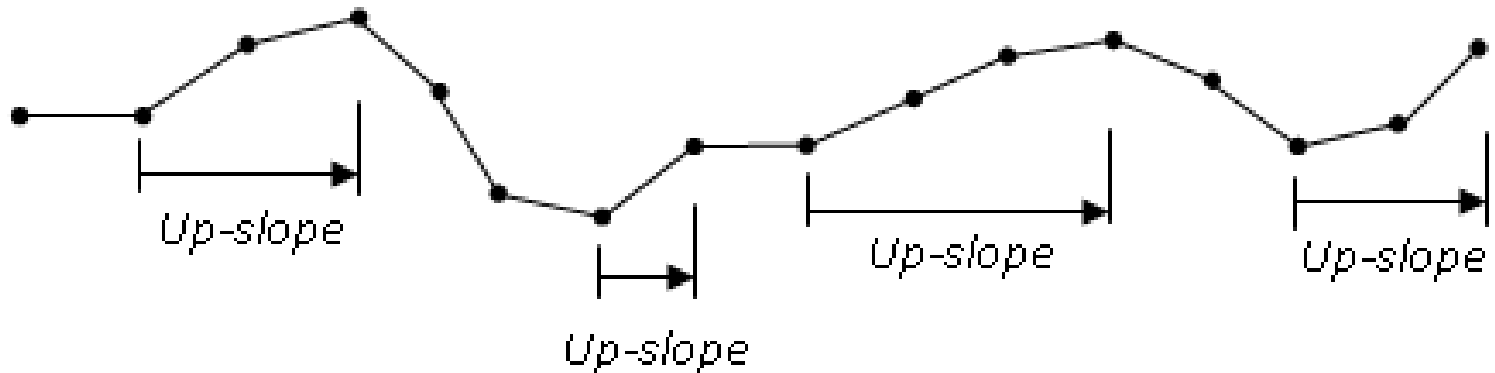
:

2.9

3.65

-1

Number of up-slopes = 4



10. Exercise: Up-slopes (2/3)

Unit8_UpSlopes.c

```
#include <stdio.h>
#define MAX 100 // maximum length of a route
// function prototypes omitted
int main(void) {
    float route[MAX];
    int route_length;

    route_length = read_route(route);
    printf("Number of up-slopes = %d\n", compute_upslopes(route, route_length));
    return 0;
}
// This function reads a list of ...
int read_route(float route[]) {
    float height;
    int length = 0; // length of route; number of values read

    printf("Enter data: ");
    scanf("%f", &height);
    while (height >= 0) {
        route[length++] = height;
        scanf("%f", &height);
    }
    return length;
}
```

10. Exercise: Up-slopes (3/3)

Unit8_UpSlopes.c

```
// This function takes a route and computes the number of upslopes.
```

```
int compute_upslopes(float route[], int size) {  
    int upslopes = 0; // number of upslopes
```

```
    return upslopes;
```

```
}
```

Summary

- In this unit, you have learned about
 - Declaring array variables
 - Using array initializers
 - Relationship between array and pointer
 - How to pass an array into a function

End of File