

CS2100

<http://www.comp.nus.edu.sg/~cs2100/>

COMPUTER ORGANISATION

Lecture #5a

Arrays, Strings and Structures



NUS
National University
of Singapore

School of
Computing



Questions?

IMPORTANT: DO NOT SCAN THE QR CODE IN THE VIDEO RECORDINGS. THEY NO LONGER WORK

Ask at

<https://sets.netlify.app/module/676ca3a07d7f5ffc1741dc65>

OR

Scan and ask your questions here!
(May be obscured in some slides)



Lecture #5: Arrays, Strings and Structures (1/2)

1. Collection of Data

2. Arrays

2.1 Array Declaration with Initializers

2.2 Arrays and Pointers

2.3 Array Assignment

2.4 Array Parameters in Functions

2.5 Modifying Array in a Function

3. Strings

3.1 Strings: Basic

3.2 Strings: I/O

3.3 Example: Remove Vowels

3.4 String Functions

3.5 Importance of '\0' in a String



Lecture #5: Arrays, Strings and Structures (2/2)

4. Structures

- 4.1 Structure Type
- 4.2 Structure Variables
- 4.3 Initializing Structure Variables
- 4.4 Accessing Members of a Structure Variable
- 4.5 Example: Initializing and Accessing Members
- 4.6 Reading a Structure Member
- 4.7 Assigning Structures
- 4.8 Returning Structure from Function
- 4.9 Passing Structure to Function
- 4.10 Array of Structures
- 4.11 Passing Address of Structure to Functions
- 4.12 The Arrow Operator (->)



1. Collection of Data

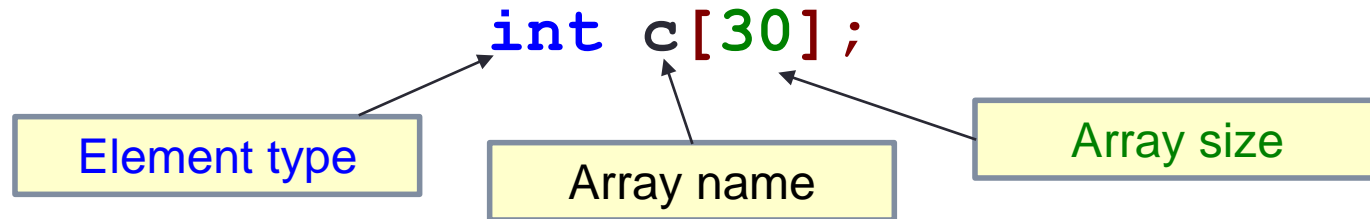
- Besides the basic data types (int, float, double, char, etc.), C also provides means to organise data for the purpose of more logical representation and ease of manipulation.
- We will cover the following in this lecture:
 - Arrays
 - Strings
 - Structures



2. Arrays (1/2)

- An array is a **homogeneous** collection of data
- The declaration of an array includes the **element type**, **array name** and **size** (maximum number of elements)
- Array elements occupy contiguous memory locations and are accessed through **indexing** (from index 0 onwards)

Example: Declaring a 30-element integer array c.



| C[0] | C[1] | C[2] | ... | | | C[28] | C[29] |
|------|------|------|-----|-----|--|-------|-------|
| 10 | 21 | 14 | 20 | ... | | 42 | 7 |



2. Arrays (2/2)

ArraySumV1.c

```
#include <stdio.h>
#define MAX 5

int main(void) {
    int numbers[MAX];
    int i, sum = 0;

    printf("Enter %d integers: ", MAX);
    for (i=0; i<MAX; i++) {
        scanf("%d", &numbers[i]);
    }

    for (i=0; i<MAX; i++) {
        sum += numbers[i];
    }

    printf("Sum = %d\n", sum);
    return 0;
}
```

- Summing all elements in an integer array

ArraySumV2.c

```
#include <stdio.h>
#define MAX 5

int main(void) {
    int numbers[MAX] = {4,12,-3,7,6};
    int i, sum = 0;

    for (i=0; i<MAX; i++) {
        sum += numbers[i];
    }

    printf("Sum = %d\n", sum);
    return 0;
}
```



2.1 Array Declaration with Initializers

- As seen in `ArraySumV2.c`, an array 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};
```

Note what happens when fewer initial values are provided.

- The following initializations are **incorrect**:

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

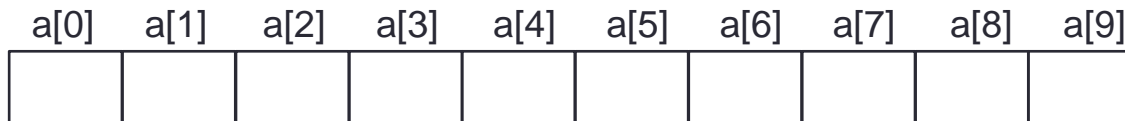
```
int f[5];
```

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



2.2 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).

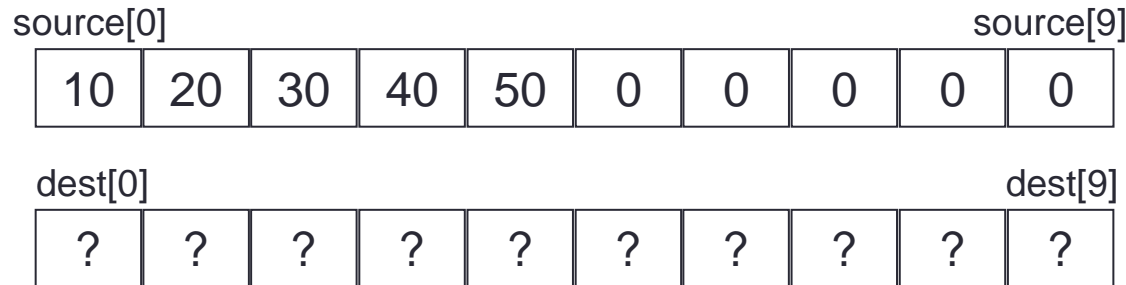


2.3 Array Assignment (1/2)

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

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

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.



2.3 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];
}
```

ArrayCopy.c

| | | | | | | | | | |
|-----------|----|----|----|----|-----------|---|---|---|---|
| source[0] | | | | | source[9] | | | | |
| 10 | 20 | 30 | 40 | 50 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | | | |
|---------|----|----|----|----|---------|---|---|---|---|
| dest[0] | | | | | dest[9] | | | | |
| 10 | 20 | 30 | 40 | 50 | 0 | 0 | 0 | 0 | 0 |

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



2.4 Array Parameters in Functions (1/3)

```
#include <stdio.h>
```

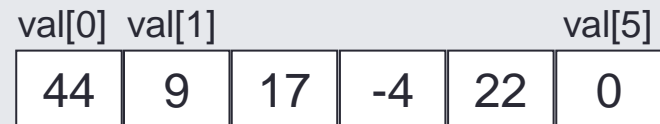
ArraySumFunction.c

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

```
int main(void) {  
    int val[6] = {44, 9, 17, -4, 22};  
    printf("Sum = %d\n", sumArray(val, 6));  
    return 0;  
}
```

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

In main():



In sumArray():

arr

size



2.4 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 in function definition:

- 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*



2.4 Array Parameters in Functions (3/3)

- Since an array name is a pointer, the following shows the alternative syntax for array parameter in function prototype and function header in the function definition

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

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

- Compare this with the `[]` notation

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

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



2.5 Modifying Array in a Function (1/2)

- We have learned that for a function to modify a variable (eg: **v**) outside it, the caller has to pass the address of the variable (eg: **&v**) into the function.
- What about an array? Since an array name is a pointer (address of its first element), there is no need to pass its address to the function.
- This also means that whether intended or not, a function can modify the content of the array it received.



2.5 Modifying Array in a Function (2/2)

```
#include <stdio.h>
```

ArrayModify.c

```
void modifyArray(float [], int);  
void printArray(float [], int);
```

```
int main(void) {  
    float num[4] = {3.1, 5.9, -2.1, 8.8};  
    modifyArray(num, 4);  
    printArray(num, 4);  
    return 0;  
}
```

In main():

| num[0] | num[1] | num[2] | num[3] |
|--------|--------|--------|--------|
| 3.1 | 5.9 | -2.1 | 8.8 |

In modifyArray():

| arr | size |
|---|------|
|  | 4 |

6.20 11.80 -4.20 17.60

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

```
void printArray(float arr[], int size) {  
    int i;  
  
    for (i=0; i<size; i++) {  
        printf("%.2f", arr[i]);  
    }  
    printf("\n");  
}
```

modifyArray() modifies
the array; printArray()
does not.



End of File

