



Arrays and Strings

TECH 3230 – Lecture 6-7
Prof Kohn
Presentation created with a little help from ChatGPT

THE UNIVERSITY OF MEMPHIS

Open (Edge or Chrome): https://www.onlinegdb.com/online_c_compiler

1

What is an Array?

- Think of an array as a numbered list

1	1
2	23
3	34
4	45
5	46

- Questions?
- How many items are in the list?
- What is the 2nd value in the list.



2

What is an Array?

- In computer terms an array is:
 - Collection of same data type
 - Stored in contiguous memory
- Syntax: `data_type name[size]`
- Index starts at 0 ← VERY IMPORTANT!
- Defining an array - Example:


```
int arr[5] = {1,23,34,45,46};
```



3

What is an Array?

Memory Location	0x100	0x101	0x102	0x103	0x104
Value	1	23	34	45	46
Index	0	1	2	3	4

Access elements using index

Example:

```
printf("%d", arr[2]); // Outputs 34
```



4

What is an Array?

Memory Location	0x100	0x101	0x102	0x103	0x104
Value	1	23	34	45	46
Index	0	1	2	3	4

- In our example: `int arr[5] = {1,23,34,45,46};`
- what is `arr` (besides the name of the array)?
 - It is the ADDRESS of the start of the array (the pointer)
- If I do a `printf("%x\n", arr);`
 - the output would be 0x100 (the address of the array)
- If I do a `printf("%d\n", *arr);`
 - the output would be 1 (the value at index 0 of the array)



5

Looping through an Array

```
#include <stdio.h>
int main()
{
    int i;
    int arr[5] = {1,23,34,45,46};

    for(int i=0; i<5; i++)
    {
        printf("%d %d\n", i, arr[i]);
    }
}
```

• Output:

```
0 1
1 23
2 34
3 45
4 46
```



6

Character Arrays

- Character arrays store characters.
- Example:
- `char letters[5] = {'H','e','l','l','o'};`



7

Remember Typical Sizes (Bytes)

- char: 1
- short: 2
- int: 4
- long: 4
- float: 4
- So really in the previous slides, char should have been used instead of int to truly be in consecutive memory locations



8

Strings in C

- Array of characters ending with `'\0'`
- Example:
- `char str[] = "Hello";`
- Note: when defining an array, `[]` means make it the size to fit the data provided.



9

Strings in C

- What is the size of the array in our example:
- `char str[] = "Hello";`
- Answer: 6
 - H, e, l, l, o, and the `\0` character to end the string!



10

String Input/Output

- Example:

```
char name[20];  
scanf("%s", name); // Note: NO & is needed since 'name' is a PTR  
printf("%s", name);
```



11

Finding the Size of an Array in C

- Arrays in C do not store their own length.
- You must calculate the size using `sizeof`.



12

Formula

- `int size = sizeof(array) / sizeof(array[0]);`



13

Example Code

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

```
int main() {  
    int numbers[] = {10, 20, 30, 40, 50};  
    int size = sizeof(numbers) / sizeof(numbers[0]);  
    printf("Array size: %d\n", size);  
    return 0;  
}
```



14

Explanation

- `sizeof(numbers)` → total bytes of array
- `sizeof(numbers[0])` → size of one element
- Divide to get number of elements



15

Important Note

- This method only works in the same scope.
- Arrays decay into pointers when passed to functions.



16

Looping through an Array (Better Version)

```
#include <stdio.h>                Why is this better?
int main()
{
    int i;
    int arr[5] = {1,23,34,45,46};
    int size = sizeof(arr) / sizeof(arr[0]);

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



17

String Functions



18

String Functions

- `#include <string.h>`
- `strlen(str)` - length
- `strcpy(dest, src)` - copy
- `strcat(dest, src)` - concatenate
- `strcmp(s1, s2)` - compare



19

String Functions - Details

B3. String Functions: <string.h>

There are two groups of string functions defined in the header <string.h>. The first have names beginning with `str`; the second have names beginning with `mem`. Except for `memcmp`, the behavior is undefined if copying takes place between overlapping objects. Comparison functions treat arguments as unsigned char arrays.

In the following table, variables `s` and `t` are of type `char *`; `cs` and `ct` are of type `const char *`; `n` is of type `size_t`; and `c` is an `int` converted to `char`.

<code>char *strcpy(s, ct)</code>	copy string <code>ct</code> to string <code>s</code> , including <code>'\0'</code> ; return <code>s</code> .
<code>char *strncpy(s, ct, n)</code>	copy at most <code>n</code> characters of string <code>ct</code> to <code>s</code> ; return <code>s</code> . Pad with <code>'\0'</code> 's if <code>t</code> has fewer than <code>n</code> characters.
<code>char *strcat(s, ct)</code>	concatenate string <code>ct</code> to end of string <code>s</code> ; return <code>s</code> .
<code>char *strncat(s, ct, n)</code>	concatenate at most <code>n</code> characters of string <code>ct</code> to string <code>s</code> ; terminate <code>s</code> with <code>'\0'</code> ; return <code>s</code> .
<code>int strcmp(s, ct)</code>	compare string <code>cs</code> to string <code>ct</code> ; return <code><0</code> if <code>cs < ct</code> , <code>0</code> if <code>cs == ct</code> , or <code>>0</code> if <code>cs > ct</code> .
<code>int strncmp(cs, ct, n)</code>	compare at most <code>n</code> characters of string <code>cs</code> to string <code>ct</code> ; return <code><0</code> if <code>cs < ct</code> , <code>0</code> if <code>cs == ct</code> , or <code>>0</code> if <code>cs > ct</code> .
<code>char *strchr(cs, c)</code>	return pointer to first occurrence of <code>c</code> in <code>cs</code> or <code>NULL</code> if not present.
<code>char *strrchr(cs, c)</code>	return pointer to last occurrence of <code>c</code> in <code>cs</code> or <code>NULL</code> if not present.



20

String Functions - Details

<code>size_t strlen(cs)</code>	return length of prefix of <code>cs</code> consisting of characters in <code>cs</code> .
<code>size_t strspn(cs, ct)</code>	return length of prefix of <code>cs</code> consisting of characters not in <code>ct</code> .
<code>char *strpbrk(cs, ct)</code>	return pointer to first occurrence in string <code>cs</code> of any character of string <code>ct</code> , or <code>NULL</code> if none are present.
<code>char *strxbrk(cs, ct)</code>	return pointer to first occurrence of <code>x</code> in <code>cs</code> or <code>NULL</code> if not present.
<code>size_t strlenn(cs)</code>	return length of <code>cs</code> .
<code>char *strerror(n)</code>	return pointer to implementation-defined string corresponding to error <code>n</code> .
<code>char *strtok(s, ct)</code>	<code>strtok</code> searches <code>s</code> for tokens delimited by characters from <code>ct</code> ; see below.



21

Searching Strings

- Use `strstr()` or manual search
- Example:
 - `char *pos = strstr(str, "lo");`



22

Modifying Strings

- Strings can be modified via indexing
- Example:
 - `str[0] = 'h';`



23

String Manipulation Example

- `#include <string.h>`
- `char str[20] = "Hello";`
- `strcat(str, " World");`
- `printf("%s", str);`



24

Advanced Example

- `#include <stdio.h>`
- `#include <string.h>`
- `int main() {`
- `char str[50] = "Embedded";`
- `strcat(str, " C");`
- `printf("%s", str);`
- `return 0;`
- `}`



25

Memory Management

THE UNIVERSITY OF
MEMPHIS

26

malloc and Arrays in C

- Understanding dynamic memory allocation and arrays



27

What is malloc?

- malloc allocates memory at runtime
- Returns pointer to memory
- Located in stdlib.h
- Returns NULL if fails



28

Static Arrays

- Declared at compile time
- Example: `int arr[5];`
- Stored on stack
- Cannot resize



29

Dynamic Arrays

- Allocated using malloc
- Example:
`int *arr = malloc(5 * sizeof(int));`
- Stored on heap
- Can resize



30

Memory Concept

- Stack: static arrays
- Heap: dynamic arrays
- malloc uses heap memory



31

Example Code

- `#include <stdlib.h>`
- `int *arr = malloc(5 * sizeof(int));`



32

Using the Array

- `arr[i] = *(arr + i)`
- Works like normal array indexing



33

Freeing Memory

- Use `free(arr);`
- Prevents memory leaks



34

Common Mistakes

- Forgetting `sizeof`
- Not checking `NULL`
- Forgetting `free`



35

Resizing with `realloc`

- `arr = realloc(arr, new_size);`
- Can grow or shrink memory



36
