

CSC209 Week 4 Notes

Hyungmo Gu

May 13, 2020

Introduction to arrays in C 1 of 3

- Array

- **Syntax:** <TYPE >VAR_NAME[ARRAY_SIZE]

```
1  #include <stdio.h>
2
3  int main() {
4      float daytime_high[4];
5  }
6
```

Introduction to arrays in C 2 of 3

- Accessing Array Elements

- C doesn't check if an array access is within the bounds of array
- Overwrites memory location if exists

```
1  #include <stdio.h>
2
3  int main() {
4      float daytime_high[4] = {1,2,3};
5      daytime_high[5] = 999;
6  }
7
```

- Segmentation fault occurs if suitable memory location doesn't exist.

```
1  #include <stdio.h>
2
3  int main() {
4      int daytime_high[4] = {1,2,3};
5      daytime_high[3000] = 999;
6  }
7
```

Introduction to arrays in C 3 of 3

- Iterating Over Arrays

- For loop

* ‘<’ is used over ‘<=’ for the end condition,i.e. $i < 4$ in for ($i = 0; i < 4; i++$).

```

1  #include <stdio.h>
2
3  int main() {
4      float daytime_high[4] = {16.0, 12.8, 14.6, 19.1};
5
6      float average_temp = 0;
7
8      int i;
9      for (i = 0; i < 4; i++) {
10         printf("Adding element %d with value %f.\n", index
, daytime_high[i]);
11         average_temp += daytime_high[i];
12     }
13
14     average_temp = average_temp / 4;
15     printf("average %f\n", average_temp);
16
17     return 0;
18 }
19

```

- Constants

* Combines multiple repeating values into one

* Used to increase maintainability and readability

```

1  #include <stdio.h>
2  #define DAYS 4 // <-- HERE!!
3
4  int main() {
5      float daytime_high[DAYS] = {16.0, 12.8, 14.6, 19.1};
6
7      float average_temp = 0;
8
9      int i;
10     for (i = 0; i < DAYS; i++) {
11         printf("Adding element %d with value %f.\n", index
, daytime_high[i]);
12         average_temp += daytime_high[i];
13     }
14
15     average_temp = average_temp / DAY;
16     printf("average %f\n", average_temp);
17
18     return 0;
19 }
20

```

Pointers in C 1 of 7

- Address in C

- `&<VARIABLE_NAME>`
- Returns memory location of variable

```

1  #include <stdio.h>
2  #define DAYS 4
3
4  int main() {
5      int i;
6      i = 5;
7      printf("Value of i: %d\n", i);
8      printf("Address of i: %p\n", &i);
9  }
10

```

- Pointer

- `<TYPE>* <VARIABLE_NAME>`
- Is used to store memory addresses

```

1  #include <stdio.h>
2  #define DAYS 4
3
4  int main() {
5      int *pt;
6      pt = &i;
7
8      printf("value of pt: %p\n", pt);
9      printf("Address of pt: %p\n", &pt);
10
11     printf("Value pointed to by pt: %d\n", *pt);
12 }
13

```

Pointers in C 2 of 7

- Assigning to Deferred Pointers

- **Syntax:** `TYPE * POINTER_NAME`
- `TYPE * <POINTER_NAME> = VARIABLE_NAME`
 - * Stores memory location of variable to pointer
 - * is the same as

```

1  <TYPE> *<POINTER_NAME>;
2  <POINTER_NAME> = VARIABLE_NAME
3

```

- ***<POINTER_NAME> = VALUE**
 - * changes the value pointed by pointer

Example:

```

1  #include <stdio.h>
2  #define DAYS 4
3
4  int main() {
5      int i = 7;
6      int *pt;
7      pt = &i; // <- stores memory location of i, i.e. 0
x7ffeeab32a28
8      *pt = 9; // <- changes the value of i to 9
9
10     printf("Value of i: %d\n", i);
11     printf("Address of i: %p\n", &i);
12
13     printf("pt points to %d\n", *pt);
14
15     return 0;
16 }
17

```

Pointers in C 3 of 7

- Pointers as Parameters to Functions
 - **Syntax:** ... <FUNCTION_TYPE> (<TYPE> *<VARIABLE_NAME>)
 - Passes variable to function by reference
 - Changing values of variable inside function affects the variable outside of function

```

1  #include <stdio.h>
2
3  void apply_late_penalty(char *grade_ptr) {
4      if (*grade_ptr != 'F') {
5          (*grade_ptr)++;
6      }
7  }
8
9  int main() {
10     char grade_moe = 'B';
11     apply_late_penalty(&grade_moe)
12

```

```

13     return 0;
14 }
15

```

Pointers in C 4 of 7

- Passing Arrays as Parameters

- **Syntax:** ... <FUNCTION_TYPE >(<TYPE >*<ARRAY_VARIABLE_NAME >)

```

1  #include <stdio.h>
2
3  int sum(int *A, int size) {
4      ...
5  }
6
7  int main() {
8      ...
9      printf("total is %d\n", sum(scores, 4));
10
11     return 0;
12 }
13

```

- Passes first element of array to function by reference
- Size of array needs to be passed independently.
 - * *sizeof* measures size of pointer value, not the array
- Array elements are also passed by reference

```

1  #include <stdio.h>
2
3  void change(int *A) {
4      A[0] = 50;
5  }
6
7  int main() {
8      int scores[4] = {4,5,-1,12};
9      ...
10     change(scores);
11     printf("First element in array has value %d\n", scores
[0]); // <- returns 50, instead of 4
12     return 0;
13 }
14

```

Pointers in C 5 of 7

- Pointer Arithmetic
 - **int***
 - * $\text{int}^* + 1$: increases memory size by 4
 - **char***
 - * $\text{char}^* + 1$: increases memory size by 4