CSC209 Week 4 Notes

Hyungmo Gu

May 13, 2020

Introduction to arrays in C 1 of 3

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

```
#include <stdio.h>

int main() {
    float daytime_high[4];
}

```

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

```
#include <stdio.h>

int main() {
    float daytime_high[4] = {1,2,3};
    daytime_high[5] = 999;
}

```

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

```
#include <stdio.h>

int main() {
    int daytime_high[4] = {1,2,3};
    daytime_high[3000] = 999;
}

```

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++).

```c
#include <stdio.h>

int main() {
    float daytime_high[4] = {16.0, 12.8, 14.6, 19.1};

    float average_temp = 0;

    int i;
    for (i = 0; i < 4; i++) {
        printf("Adding element %d with value %f.\n", index
, daytime_high[i]);
        average_temp += daytime_high[i];
    }

    average_temp = average_temp / 4;
    printf("average %f\n", average_temp);

    return 0;
}

```

  - Constants
    - Combines multiple repeating values into one
    - Used to increase maintainability and readibility

```c
#include <stdio.h>
#define DAYS 4 // <-- HERE!!

int main() {
    float daytime_high[DAYS] = {16.0, 12.8, 14.6, 19.1};

    float average_temp = 0;

    int i;
    for (i = 0; i < DAYS; i++) {
        printf("Adding element %d with value %f.\n", index
, daytime_high[i]);
        average_temp += daytime_high[i];
    }

    average_temp = average_temp / DAY;
    printf("average %f\n", average_temp);

    return 0;
}

```

Pointers in C 1 of 7

- Address in C
  - **& <VARIABLE_NAME >**
  - Returns memory location of variable

```c
#include <stdio.h>
#define DAYS 4

int main() {
    int i;
    i = 5;
    printf("Value of i: %d\n", i);
    printf("Address of i: %p\n", &i);
}

```

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

```c
#include <stdio.h>
#define DAYS 4

int main() {
    int *pt;
    pt = &i;

    printf("value of pt: %p\n", pt);
    printf("Address of pt: %p\n", &pt);

    printf("Value pointed to by pt: %d\n", *pt);
}

```

Pointers in C 2 of 7

- Assigning to Deferenced Pointers
  - **Syntax:** $TYPE * POINTER\_NAME$
  - **TYPE *<POINTER_NAME > = VARIABLE_NAME**
    - Stores memory location of variable to pointer
    - is the same as

```
    <TYPE> *<POINTER_NAME>;
    <POINTER_NAME> = VARIABLE_NAME

```

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

Example:

```
#include <stdio.h>
#define DAYS 4

int main() {
    int i = 7;
    int *pt;
    pt = &i; // <- stores memory location of i, i.e. 0
x7ffeeab32a28
    *pt = 9; // <- changes the value of i to 9

    printf("Value of i: %d\n", i);
    printf("Address of i: %p\n", &i);

    printf("pt points to %d\n", *pt);

    return 0;
}

```

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

```
#include <stdio.h>

void apply_late_penalty(char *grade_ptr) {
    if (*grade_ptr != 'F') {
        (*grade_ptr)++;
    }
}

int main() {
    char grade_moe = 'B';
    apply_late_penalty(&grade_moe)

```

```
        return 0;
    }

```

Pointers in C 4 of 7

- Passing Arrays as Parameters
  - **Syntax:** ... <FUNCTION_TYPE >(<TYPE >*<ARRAY_VARIABLE_NAME >)

```
#include <stdio.h>

int sum(int *A, int size) {
    ...
}

int main() {
    ...
    printf("total is %d\n", sum(scores, 4));

    return 0;
}

```

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

```
#include <stdio.h>

void change(int *A) {
    A[0] = 50;
}

int main() {
    int scores[4] = {4,5,-1,12};
    ...
    change(scores);
    printf("First element in array has value %d\n", scores
[0]); // <- returns 50, instead of 4
    return 0;
}

```

Pointers in C 5 of 7

- Pointer Arithmetic
  - Moves the memory location by x amount, where x is int.
  - **General Formula:** $p = p + (\text{sizeof(type)} \cdot n)$, given

```
    type k;
    type *p = &k;

    int n;
    p = p + n;

```

  - Example
    - int* var1 + 1 : increases memory size by 4
    - char* var2 + 1 : increases memory size by 3
- Pointer Arithmetic in Array
  - Moves the memory location by x amount, where x is int.
  - **General Formula:** $p[k] == p + k$, given

```
    int n = //Arbitrary positive int value;
    type A[n];
    type *p = &A;

    int k = //Arbitrary int value with size less than n;
    print(*p);
    print(*(p+k));

```