2D Array and Pointers

Computer Science Department

Eastern Washington University

Yun Tian (Tony) Ph.D.

Outline for Today

- malloc
- calloc
- 2D array
- pointers to pointers

Dynamic Allocation

- **char *s = (char *)malloc(numChars * sizeof(char));**
  - allocate memory spaces for numChars characters on the Heap, not the stack.
  - malloc returns a generic or void *, thus we have to do the type casting.

Dynamic Allocation

```
char * myStrcpy( char * s )
{
        char *t = (char *) malloc(strlen(s) + 1);
        strcpy(t, s);
        return t;    //  we can return a pointer from a function because  the
                     // dynamically allocated memory is on the Heap, not on  stack.
                     //memory on heap keep available until you call free().
}
```

Good practice!!

```
char * myStrcpy( char * s )
{
        char t[100];  //static array
        strcpy(t, s);
        return t;       // t is on the stack, can NOT returned from a function.
}                        //after call stack for this function pops up, t is gone.
```

Not Working this Way !!

Deallocation

- calloc works the same way as malloc, but initialize the allocated memory to zero.
- We use **free(void * ptr)** to recollect the memory that is **not useful** any more.
  - Differs from Java (taken care of by garbage collector for you)
  - You have to do it explicitly in C.
    - Otherwise, you get memory leak.

Deallocation

- E.g.
- char *name = (char *) malloc(length * sizeof(char);
- Then you use name in somewhere else, maybe outside of this current function.
- Once you know you **will not** use name any longer,
- free(name); → this free all **length** characters in one statement. You do not need to free each characters individually.

2D Arrays

- We explore the relationship between 2D array and pointers.
  - char multi[5][10];
  - what does this mean?
    - char multi[5][10];
  - The red underlined part means **multi** is an array of 5 elements.
  - For this 2D array, each element in multi[5] is another 1D char array, with size of 10 characters.

2D Arrays

- char multi[5][10];
  - Here, we have an array of 5 arrays, each with 10 characters in it). Each element in mutil[5] is another array.
  - Assume we fill mutil with data in the below.

```
multi[0] = {'0','1','2','3','4','5','6','7','8','9'}
multi[1] = {'a','b','c','d','e','f','g','h','i','j'}
multi[2] = {'A','B','C','D','E','F','G','H','I','J'}
multi[3] = {'9','8','7','6','5','4','3','2','1','0'}
multi[4] = {'J','I','H','G','F','E','D','C','B','A'}
```

2D Arrays

- multi[0][3] → ‘3’; multi[4][0] → ‘J’
- 2D arrays are stored in row major order in the memory in C, same as Java.
- Since arrays are contiguous in memory, our actual memory block for the above should look like:
- 0123456789abcdefghijABCDEFGHIJ9876543210JIHGFEDCBA
  ↑---starting at the address &multi[0][0]

2D Arrays

- Recall that for 1D array,
  - int a[10]; int *p = a;
  - An array name is equivalent to a pointer, except for the fact that array name is considered a constant.
  - p + 1 points to the second element in a.
  - *(p+1) is equal to a[1] or p[1]. Either way is allowed in C program.
  - Question: Go back to the 2D array multi,
    - what does (multi + 1) point to?
    - how about *(multi + 1), and   *(*(multi + 1) + 9) ?

2D Arrays

- char multi[5][10];
  - Here, we have an array of 5 arrays of 10 characters each.
    - Each element in mutil[5] is another array.
  - Assume we fill mutil with data in the below.

multi[0] = {'0','1','2','3','4','5','6','7','8','9'}

multi[1] = {'a','b','c','d','e','f','g','h','i','j'}

multi[2] = {'A','B','C','D','E','F','G','H','I','J'}

multi[3] = {'9','8','7','6','5','4','3','2','1','0'}

multi[4] = {'J','I','H','G','F','E','D','C','B','A'}

2D arrays

- In char multi[5][10];
- multi[1] in multi[5][10]
  - char (multi[1])[10];
  - char (student)[50];
  - Red part as a whole are equivalent. Both are name of 1D array.
- Therefore multi[1] is an address, the start address of the row ONE,
  - equal to &multi[1][0].

2D arrays

- IN char multi[5][10];
- Recall that multi[1] is same as *(multi + 1).
- But multi[1] is the name of a 1D array which can hold 10 characters.
  - *(multi + 1) is also an address or (array name),
  - same as multi[1].
- How To get the address of second element in 1D array that (multi[1]) points to?

2D arrays

- IN char multi[5][10];
- To get the address of the second element in 1D array that (multi[1]) points to,

  (*(multi + 1) + 1) ==== &multi[1][1]

- To get the content(value) of the second element in 1D array that (multi[1]) points to,

  *(*(multi + 1) + 1) ==== multi[1][1]

2D arrays

- Now we access values in 2D array using:
  - *(*(multi + 3) + 1) returns the value at row 3 and column 1, which is char ‘8’.
  - We get the same value as multi[3][1].

```
for (row = 0; row < ROWS; row++)
 {
        for (col = 0; col < COLS; col++)
        {
            printf("\n%d  ",multi[row][col]);
            printf("%d ",  *( *(multi + row) + col) );
        }
  }
```

2D Arrays

Demo of 2d_demo.c

Summary

- **Addresses in 2D array in char multi[5][10].**
- _multi[i] → *(multi + i)_
- _*multi + 1 → *(multi + 0) + 1 → &multi[0][1], But NOT &multi[1][0]._
- _*(multi + i) + j → &multi[i][j]_
- _*(*(multi + i) +j) → multi[i][j]_