ECE 220 Computer Systems & Programming

Lecture 12 – Strings and Multi-dimensional Arrays

Outline

- Chapter 16.3-16.4

- Key concepts
  - Array as Function Parameters
  - Strings
  - Multi-dimensional arrays

Arrays as Function parameters:

- How do we pass array to a function?

C passes arrays by reference
- the address of the array (i.e., address of the first element) is written to the function's activation record
- `[]` indicate to the compiler that the corresponding parameter will be the base address of an array of the specified type.

```c
1  #include <stdio.h>
2  int foo(int x[]);
3  int main()
4  {
5      int y[5] = {1,2,3,4,5};
6      foo(y);
7      printf("%d\n", y[3]);
8      return 0;
9  }
10
11 int foo(int x[])
12 { x[3] = 10;
13 }
```

```c
1  #include <stdio.h>
2  int foo(int *x);
3  int main()
4  {
5      int y[5] = {1,2,3,4,5};
6      foo(y); //y refers to &y[0]
7      printf("%d\n", y[3]);
8      return 0;
9  }
10
11 int foo(int *x)
12 { x[3] = 10;
13 }
```

Review of arrays and pointers

Write a program that takes as input N integers from the user, passes them in an array to a function to compute their average. N is a positive number.

```c
#include <stdio.h>
/* size of the array */
#define N 10
float average(int *input_array, int size);
int main()
{
    int array[N];
    int i;
    float Average = 0.0f;
    /* get array elements from the user */
    for (i = 0; i < N; i++)
    {
        printf("Enter array element %i: ", i);
        scanf("%i", &array[i]);
    }
    Average = average(&array[0], N);
    printf("average=%f\n", Average);
    return 0;
}

float average(int *input_array,int size)
{
    int i;
    float sum = 0.0f;
    float Avg = 0.0f;
    /* sum up the array elements */
    for (i = 0; i < size; i++)
        sum += input_array[i];
    /* average = sum / size */
    Avg = sum/size;
    return Avg;
}
```

Pointer Array Duality

```
char word[10];
char *cptr;
cptr = word; //assign cptr to point to word
```

cptr	word	&word[0]
(cptr + n)	word + n	&word[n]
*cptr	*word	word[0]
*(cptr + n)	*(word + n)	word[n]

Example: Dice Roll Matching Game

Problem Statement:

Roll a dice three times to generate three random numbers.

Use the generated random numbers as indices to pick three random characters from an array of characters.

guess the three random characters and check how many characters you guessed right. Also, check if it was a perfect match. You are allowed to use three trials.

$\Rightarrow$ We will three functions:

$\Rightarrow$ roll_a_dice

$\Rightarrow$ pick_characters

$\Rightarrow$ comp_characters

Note: *function prototype are declared in functions.h

       *functions are defined in dice_roll.c

Strings

Allocate space for a string just like any other array:

```
char outputString[16];
```

Space for string must contain room for <u>terminating zero</u>. [see the simple_string.c code on github]

Special syntax for initializing a string:

```
char outputString[16] = "Result = ";
```

...which is the same as:

```
outputString[0] = 'R';
outputString[1] = 'e';
outputString[2] = 's';
...
```

Null terminating strings – '\0' special sequence that corresponds to the null character.

Simple_string.c

Space for string must contain room for <u>terminating zero</u>. [see the simple_string.c code on github]

```c
1    // C String example
2    #include <stdio.h>
3
4    int main()
5    {
6        char name_scanf[5]="HelloHowAreYou?";
7        printf("\n=============\n");
8        printf("%c\n", name_scanf[5]);
9        printf("%s\n", name_scanf);
10
11       return 0;
12   }
```

//printf ("%s") -- print characters up to terminating zero
// printf("%s\n", name_scanf);

I/O with Strings

printf and scanf use "%s" format character for string

printf -- print characters up to terminating zero

```
printf("%s", outputString);
```

scanf -- read characters until whitespace,
store result in string, and terminate with zero

```
scanf("%s", inputString);
```

gets,fgets – reads line into string; stops when newline character is read

```
gets(str);
fgets(str,10,stdin);
```

Example: gets

```c
#include <stdio.h>
int CountOccurence(char* cptr, char test);

int CountOccurence(char* cptr, char test) {
   int count = 0;
   while(*cptr!='\0') {
      if(*cptr++==test) {
         ++count;
      }
   }
   return count;
}
int main() {
   char text[50];
   char testchar = 'a';
   int count = 0;

   printf("Enter string: ");
   gets(text);

   count = CountOccurence(text, testchar);

   printf("Character a: %d\n", count);
   return 0;
}
```

ILLINOIS

fgets:

```c
// C program to illustrate
// fgets()
#include <stdio.h>
#define MAX 15
int main()
{
    char buf[MAX];
    fgets(buf, MAX, stdin);
    printf("string is: %s\n", buf);

    return 0;
}
```

Multi-dimensional Arrays

int a [2][3];

	Column 1	Column 2	Column 3
Row 1	a[0][0]	a[0][1]	a[0][2]
Row 2	a[1][0]	a[1][1]	a[1][2]

In memory

a[0][0]
a[0][1]
a[0][2]
a[1][0]
a[1][1]
a[1][2]

- 2D arrays
  - <type> <name> [<dim1>][<dim2>];
  - e.g., int a[2][3];

*multi-dimensional array is stored in row-major order
   Flat index = row x (width of row) + col
         (i.e. a[1][1] = 1*3+1 = 4)

Initialize Multi-dimensional Array

```
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
```
or
```
int a[][3] = {{1, 2, 3}, {4, 5, 6}};
```
or
```
int a[2][3] = {1, 2, 3, 4, 5, 6};
```

```
Some expressions:
a[0][2]  ⇔  *(a[0] +2)
a[0]  ⇔  *a
```

Example: Image Storage

Image size is 8x10 pixels (8 rows, 10 pixels in each row)

Image[0][0]

Image[0][9]

Image[2][6]

Image[row#][column#]

Image[7][9]

Image[7][0]

8 pixels 'tall' (height)

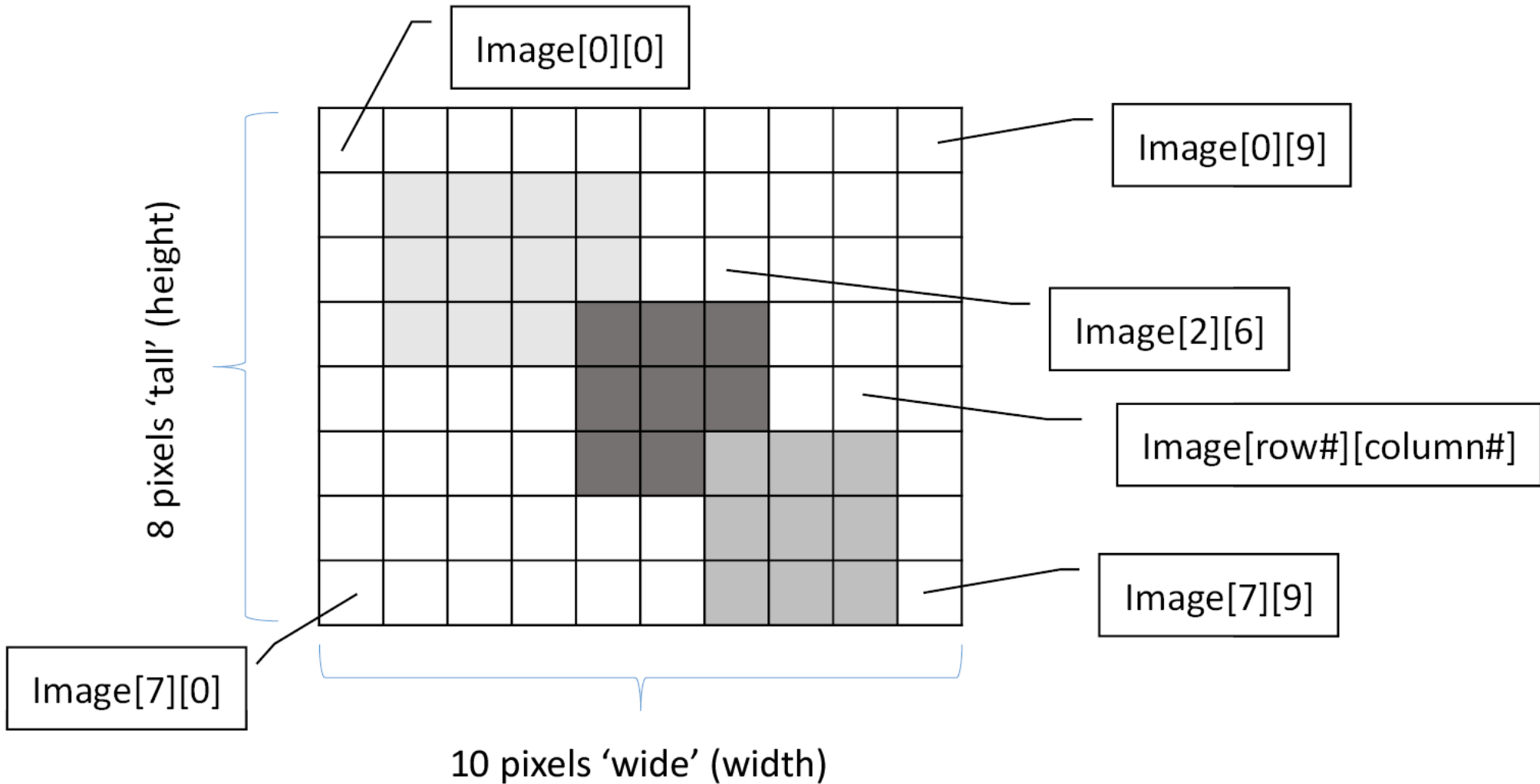10 pixels 'wide' (width)

Image Analysis Example Problem:

Problem statement:

Count the number of 2x2 pixel blocks in an image of size 5x5 pixels that match the reference block. Here both Image and Block are passed as reference to the function Match.

```c
//Match Function
#include <stdio.h>
int Match(int *im, int *ma, int w, int h, int mw, int mh)
{
   int count = 0;
   int x, y, dx, dy;
   int match;

   for(x=0;x<w-mw+1;++x) {
      for(y=0;y<h-mh+1;++y) {
         match = 1;
         for(dx=0;dx<mw;++dx) {
            for(dy=0;dy<mh;++dy) {
            if(im[(x+dx)*w+(y+dy)]!=ma[dx*mw+dy])
            //if (*(im+((x+dx)*w+(y+dy))) != *(ma+(dx*mw+dy)))
               {
                  match = 0;
                  break;
               }
            }
         }

         if (match==0)
               break;
         }

         if(match==1) {
            ++count;
         }
      }
   }
   return count;
}
```

Main Function:

```c
36  int main() {
37      int image[4][4] = {{1,2,3,4},{1,2,3,4},{1,2,1,2},{3,4,1,2}};
38      int match[2][2] = {{1,2},{1,2}};
39      int count = 0;
40      int im_w = 4, im_h = 4;
41      int m_w = 2, m_h = 2;
42
43      printf("\n");
44
45      count = Match(&image[0][0], &match[0][0], im_w, im_h, m_w, m_h);
46
47      printf("Count: %d\n", count);
48
49      return 0;
50  }
51
```

Exercise: implement a function that interchanges two rows of a 5x5 matrix. The functions takes three arguments: pointer to the matrix, row number x and row number y.

```
#define SIZE 5
void row_interchange(int *matrix, int x, int y){
```

```
}
```