ECE 220 Computer Systems & Programming

Lecture 12 – Strings and Multi-dimensional Arrays

ECE ILLINOIS

ILLINOIS

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
#include <stdio.h>
int foo(int x[]);
int main()
{
   int y[5] = {1,2,3,4,5};
   foo(y);
   printf("%d\n", y[3]);
   return 0;
}

int foo(int x[])
{ x[3] = 10;
}
```

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

int foo(int *x)
{ x[3] = 10;
}
```

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

```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;
}
```

```c
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]

Strings

Allocate space for a string just like any other array:

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

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

```
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;
}
```

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]

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

In memory

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

*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

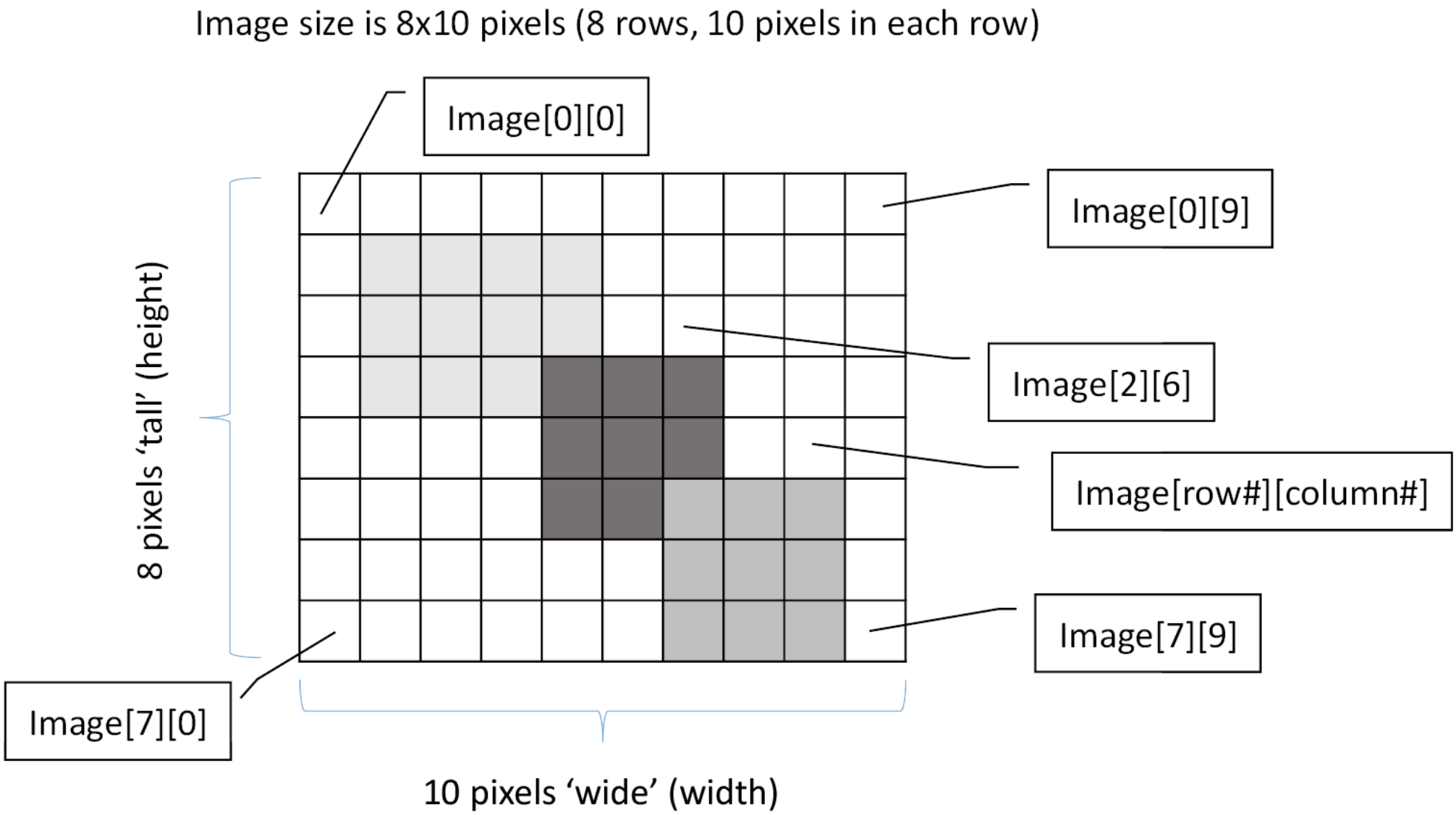

Image Analysis Example Problem:

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.

Code: main

```c
#include <stdio.h>
int Match(int im[][4], int ma[][2], int w, int h, int mw, int mh);

int main() {
  int image[4][4] = {{1,2,3,4},{1,2,3,4},{3,7,1,2},{3,4,1,2}};
  int match[2][2] = {{1,2},{1,2}};
  int count = 0;
  int im_w = 4, im_h = 4;
  int m_w = 2, m_h = 2;

  int *iptr = image[2];

  for(count = 0;count < 4;++count) {
    printf("%d ", iptr[count]);
  }
  printf("\n");

  count = Match(image, match, im_w, im_h, m_w, m_h);

  printf("Count: %d\n", count);

  return 0;
}
```

Match Function:

```
25  int Match(int im[][4], int ma[][2], int w, int h, int mw, int mh) {
26    int count = 0;
27    int x, y, dx, dy;
28    int match;
29
30    for(x=0;x<w-mw+1;++x) {
31      for(y=0;y<h-mh+1;++y) {
32        match = 1;
33        for(dx=0;dx<mw;++dx) {
34          for(dy=0;dy<mh;++dy) {
35            printf("x %d, y %d, dx %d, dy %d\n", x, y, dx, dy);
36            if(im[x+dx][y+dy]!=ma[dx][dy]) {
37              match = 0;
38              break;
39            }
40          }
41        }
42        if(match==1) {
43          ++count;
44        }
45      }
46    }
47
48    return count;
49  }
```

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[SIZE][SIZE], int x, int y){

}
```