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 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;
}
```

```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]

cptr = cptr + 1 (acceptable)

word = word + 1 (Compiler error!)

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 next page]

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.

```
1   // C String example
2   #include <stdio.h>
3
4   int main()
5   {
6       //char name_scanf[6]="Hello";
7       char name_scanf[6]={'H','e','l','l','o','\0'};
8       printf("\n==============\n");
9       printf("%c\n", name_scanf[5]);
10      printf("%s\n", name_scanf);
11
12      return 0;
13  }
```

//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;
}
```

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;
}
```

sscanf:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main () {
    int seed, count;
    char post[5], seed_str[200];
    strcpy( seed_str, "1989 lmnop" );

    //The sscanf statement below reads the integer into seed.

    count=sscanf (seed_str, "%d%1s", &seed, post);

    printf("the seeed is %d and count %d", seed,count);
    printf("\n%s",post);

   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};
```

Initialize Multi-dimensional Array

```
int a[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

or

```
int a[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

or

```
int a[3][3] = {{1, 2, 3, 4, 5, 6, 7, 8, 9};
```

```
*a       → {1,2,3} (addr of 1)
a[0]     → {1,2,3} (addr of 1)
*(a+1)   → {4,5,6} (addr of 4)
a[1]     → {4,5,6} (addr of 4)
**a      → 1       *(*(a+2)+1)   → 8
a[0][0]  → 1       a[2][1]       → 8
```

Initialize Multi-dimensional Array

```
int a[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

or

```
int a[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

or

```
int a[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

```
int *p[2];    → p is an array of int type pointers

p[0] = &a[0][0];
p[1] = &a[1][1];


*p[0] → 1

*p[1] → 5
```

Exercise: (i) Exchange two rows in matrix
(ii) Transpose a matrix and
(iiii) print the result

```
// Swap x-th row and y-th row
void swap(int x, int y, int a[][col])
{

}
```

```
// Transpose a[N][M] matrix and store in b[M][N] matrix
void transpose(int a[N][M], int b[M][N])
{

}
```

Function declaration and main function

```c
#include<stdio.h>
#define row 4
#define col 3
void swap(int x, int y, int a[][col]);
void transpose(int a[][col],int b[][row]);
void print_2D(int a[][col]);

void print_2D_ptr(int r, int c, int *a);

int main()
{
int array[row][col]= {{1,2,3},{4,5,6},{7, 8, 9},{10, 11, 12}};
int array_t[col][row];
print_2D(array);
swap(1,2,array);
print_2D(array);
transpose(array,array_t);
print_2D_ptr(col, row, &array_t[0][0]);
return 0;
}
```

swap and transpose functions:

```
void swap(int x, int y, int a[][col])
{
    int temp,i;
    for (i=0; i<col; i++)
    {
        temp=a[x][i];
        a[x][i]=a[y][i];
        a[y][i]=temp;
    }
}

void transpose(int a[][col], int b[][row])
{
    int i, j;
    for (i=0; i<row; i++){
        for(j=0; j<col; j++)
        {
            b[j][i]=a[i][j];

        }
    }
}
```

Print Functions

```c
//Allows us to use 2D indexing format
void print_2D(int a[][col]){
int i, j;
for(i=0; i<row; i++){
for(j=0; j<col; j++){
printf("%d ", a[i][j]);
}
printf("\n");
}
printf("\n");
}

//Allows us to print Matrix of any dimension:
void print_2D_ptr(int r, int c, int *a){
int i, j;
for(i=0; i<r; i++){
for(j=0; j<c; j++){
printf("%d  ", a[i*c+ j]);
}
printf("\n");
}
printf("\n");
}
```