PDS Lab Section 15

January 28, 2021

General Instructions

- Add header in front of your program

```
/*
*     Section 15
*     Roll No  :  20CS30010
*     Name     :  Your Name
*     Assignment No : 3
*     Description : Program to check points
*/
```

- Name your file with assgnX_Y.c, where X is the assignment number and Y is your roll no.
  - assgn9_20ME10010, assgn10_20CE30014,…..

Indenting your program

- Give proper indentation (space in front of a line) in your program to make your program look nice
- Indent every block in your program consistently
  - What is a block? – statements inside if, else, for, while etc…(for now)
  - Idea is to be able to see which statements are part of which if or which else or which for/while etc. easily
    - Helps in making less mistakes in missing braces etc. when you write programs
    - Helps in easier debugging if your program does not work

Badly indented program

```
int main()
{
int i, j,k,n =10,sum, count;
for (i=0; i<n; i++)
{
sum=0; count=0;
for (j=0;j<n;j++)
{ scanf(“%d”,&k);
if (k>0)
{sum = sum + k;
 count = count + 1;
}
printf(“Sum=%d Count=%d\n”, sum, count);
}
}
}
```

Properly Indented Program

```
int main()
{
    int i, j,k,n = 10, sum, count;
    for (i=0; i<n; i++)
    {
        sum=0;
        count = 0;
        for (j=0; j<n; j++)
        {
            scanf(“%d”,&k);
            if (k>0)
            {
                sum = sum + k;
                count = count + 1;
            }
            printf(“Sum=%d Count=%d\n”, sum, count);
        }
    }
}
```

Commenting a function

- Before each function declaration, add a comment with the following
  - The name of the function
  - A one line description for each parameter
  - A one line description of the return value
  - A small description of what the function does

- Example:

```
/****************
Function name: FindMin
Parameters:
            A : array of integers
            n : no. of integers in the array A
Return value: The minimum integer in array A
Description: This function finds the minimum value of a set of
                    integers stored in array A
****************/

int FindMin(int A[ ], int n)
{
   ….
}
```

Intermediate Submission for the First Assignment

- For the first assignment, you will see two submission links in moodle
  - Assignment 13 Submission (Intermediate)
  - Assignment 13 Submission (Final)
- You **must submit** the .c file containing whatever you have done so far for Assignment 11 (first assignment today) in the link "Assignment 13 Submission (Intermediate)" strictly between 10-30 am to 10-45 am (**the link will open at 10-30 am and close at 10-45 am**)
  - Submit whatever you have done till then, we want to see how regularly you are progressing
  - **Not submitting will incur 30% penalty irrespective of what your final submitted version is**
- Submit the final .c file using the link "Assignment 13 Submission (Final)" as usual anytime before the lab ends (**must submit for grading, only the final version will be graded**)

Today’s Topic: Pointers

Basics of Pointers

Pointers

- Meant to hold the address of a location in memory where a variable value is stored
- *char *p* : *p* is a pointer to char
  - can hold the address of a location where a char type variable's value will be stored
  - We say that p points to a char
- Similarly can have
  - *int *p, double *q*, etc…
- **Very Very Important to remember**
  - A pointer by itself points to nothing meaningful
  - You must initialize it to point to something

- If *x* is a pointer type variable
  - **x* gives the content of the location pointed to by *x*
  - You can use **x* as a variable wherever you want to use a variable in a program
- If *y* is any variable, *&y* gives the address of the location where the value of the variable *y* is stored

```
int *p, x;
x = 5;
printf("Content of location pointed to by
  p is %d", *p);
```

- **WRONG**, as *p* is not initialized to anything.
  - When you run your program, in CodeBlock, either you will see no value printed for *p, or a garbage value printed
  - Run it and see for yourself what gets printed so that you will know during lab when you see it

```
int *p, x;
x = 5; p = &x;
printf(“Content of location pointed to by
  p is %d”, *p);
```

- CORRECT, will print 5

Equivalence of Arrays and Pointers

- Arrays can be accessed with pointers
- *int A[100]*
  - Array *A* is a block of 100 integers, stored one after the other in memory
  - *A* is a pointer of type int that points to the first location in the array
  - So if you print *A* and *&A[0]*, they will print the same value (print it and check)
  - Similarly, **A* and *A[0]* will print the same value (print it and check)
  - The next location in the array can be obtained by incrementing the pointer
    - *A*+1 and *&A[1]* will be the same, *A*+2 and *&A[2]* will be same
    - *(*A*+1) and *A*[1] will be same, *(*A*+2) and *A[2]* will be same

```
int A[100], *p, i;

p = &A[0];
for (i=0; i<100; i++)
        scanf("%d", p + i);  /* no &, as already a pointer */

for (i=0; i<100, I++)
        printf("%d  ", *(p+i));
```

Pointer Arithmetic

- If *p* is a pointer to variable of type *X* (*X* can be char, int, float, double etc.), then
  - *p* + 1 will print *p* + (the no. of bytes needed to store a variable of type *X*)
- Example:
  - If *p* = 1000, and *p* is pointer to int, then *p*+1 will be 1004 (since an integer takes 4 bytes)
  - If *p* = 1000, and *p* is pointer to char then *p* + 1 will be 1001 (since a char takes 1 byte)
- How to know which type takes how many bytes? Can call predeclared functions
  - *sizeof(int), sizeof(char)*,…

Passing Pointers as Parameters

Passed the same way as declaring pointers

```
int FIndMax (int *p, int *q)
{
    if (*p > *q ) return (*p);
    else return (*q);
}
```

When you call the function, make sure that you pass a pointer to an int

```
int main()
{
   int a, b, max;
   scanf(“%d%d”, &a, &b);
   max = FindMax(&a, &b);  /* pass pointers */
   printf(“The max of two numbers is %d\n”, max);
}
```

```
int main()
{
   int *a, *b, max;
   a = (int *) malloc (sizeof(int));
   b = (int *) malloc (sizeof(int));
   scanf("%d%d", a, b); /*a, b are already pointers */
   max = FindMax(a, b);
   printf("The max of two numbers is %d\n", max);
}
```

Dynamic Allocation

- Allocate memory to hold a variable, array etc. when needed
- No need to declare the array beforehand
- Free the memory when no longer needed
- Use the malloc() and free() function already provided
  - We will only introduce you to malloc() today

malloc

- Allocate a contiguous block of memory during running
- Returns a void pointer, need to typecast it to appropriate type
- Parameter of malloc: how many bytes to allocate

```
int *p;
p  = (int *)malloc(sizeof(int));
*p = 5;
printf(“Value stored at p is %d\n”, *p);
```

```
int *p, n, I;
printf("how many integers in array: ");
scanf("%d", &n);
p = (int *)malloc( n * sizeof(int));
/* now you can use it using pointers,
  or just use this as an array *.
for (i=0; i<n; i++)
  scanf("%d", &p[i]);
  /* could have used p+i also, but array
  is more natural */
```

- Can malloc memory for any number of any type of variable, including structures and user-defined types

```
struct complex *p;

/* allocate memory for one struct complex type variable */
p = (struct complex *) malloc(sizeof(struct complex));

/* allocate array of 10 struct complex type variables */
p = (struct complex *) malloc(10 * sizeof(struct complex));
```

Now can access the structures in the array as p[0], p[1], p[2],…(members accessed as p[0].re, p[1].im,…)

*Can also be accessed as *p, *(p+1), *(p+2)… (members as (*p).re, (*(p+1)).im,…) but again, array notation is more natural to understand, so use it*

Practice Problem 1

- Read an integer n
- Dynamically allocate an array A of n floating point numbers
- Read in n floating point numbers in the array A
- Print the n numbers read in array A using array notation (A[ ])
- Print the n numbers read in array A again using pointer notation (*A etc.) and verify that the same numbers are printed

Practice Problem 2

- Read in two strings s1 and s2 using %s (assume each string has less than 100 chars)
- Dynamically allocate memory to store a string s3 that will be the concatenation of s1 and s2. You should not allocate more storage than necessary. To do this
  - Find the lengths of s1 and s2
  - Use malloc to allocate memory equal to sum of the lengths + 1 (don’t forget the space for the ‘\0’)
- Store the concatenation of s1 and s2 in s3
  - First copy s1 into s3
  - Next copy s2 after s1 in s3
  - Finally put the ‘\0’ at end

Returning more than one value from a function

Returning more than one value from a function

- The return statement can return only one value
- What if you want to return more than one thing from a function?
  - Return the max and the index where it is found in an array
  - Return both the min and max in an array
- Solution: return one value using a return statement as usual, return the others in variables passed as pointers

```
int FindMaxInArray (int A[ ], int n, int *index)
{
    int i, max = A[0], k = 0;
    for (i=1; i < n; i++)
    {
        if (A[i] > max)
        {
                max = A[i];
                k = i;
        }
    }
    *index = k;
    return (max);
}
```

```
int main()
{
      int i, n, *A, pos, max;
      scanf("%d", &n);
      A = (int *)malloc(n*sizeof(int));
      for (i=0; i < n; i++)
         scanf("%d", &A[i]);
      max = FindMaxInArray(A, n, &pos);
      printf("Max is %d, index of max is %d\n", max, pos);
      return 0;
}
```

When to pass a parameter as a pointer?

- Look at each parameter, say x
- If the purpose of x is ONLY to pass a value FROM the calling function (for ex., from main( )), no need to pass x as a pointer
- If the value of x will be modified in the called function (ex. FindMaxInArray()), and the changed value is to be used in the calling function (ex., main( )), pass x as a pointer (or the change will not be there after the function returns)
- Arrays are always passed as pointers by default, so if the array is changed inside the function, the change will stay on return, no extra thing for you to do

A Classic Example

```
void swap (int x, int y)
{   int temp;
    temp = x;   x = y;  y=temp;
}

void main( )
{
     int a=10, b =5;
     swap(a, b);
     printf(“Value of a and b are %d and %d\n”, a, b);
}
```

This will still print a=10 and b=5 !! No swapping !!!

Correct Swap Function

Problem is x and y are not passed as pointer (values are modified inside function, but since not passed as pointers, change is lost on return)

```
void swap (int *x, int *y)
{
    int temp;
    temp = *x;  * x = *y;  *y = temp;
}
```

Call with &a, &b from main( )

Now it will print a=5 and b=10;

Practice Problem 1

- Write a function that will return both the sum of elements and the number of elements > 10 in an integer array of n elements.
  - Think what the parameters of the function should be based on what is told in the slides before and the example shown.
- Write a main function that reads in an integer n, then dynamically allocates an array of n integers, reads n integers in the array from the keyboard, and then calls the function to find the sum of the elements and the the number of elements > 10 in the array

Practice Problem 2

- Write a function that takes a null-terminated string, and returns its length and the number of non-alphabet characters (i.e., characters not in ‘a’-’z’ or ‘A’-’Z’) in it.
  - Think what the parameters of the function should be based on what is told in the slides before and the example shown.
- Write a main function that reads in a string, and then calls the function to find the length and number of non-alphabet characters in it.

Summary

- Basic pointer usage
  - If you have declared a pointer *p*, and are doing **p* anywhere, always check that *p* has been assigned something before that with a *p= ….* statement
    - Either another pointer directly like *p=&x;*
    - Or the result of a malloc like *p= (int *)malloc(…)*
- Dynamic allocation of arrays
  - If you are dynamically allocating an array A, do not declare the array as you were doing for static arrays (so NO *int A[100]* etc. needed). Declare a pointer variable like *int *A*
  - Then malloc and assign to *A* as shown earlier
  - For malloc(), do not forget to typecast (the *(int *), (float *),…*part before malloc as shown
  - After you have malloc'ed the array *A*, use it just as a normal static array that you have seen so far using *A[ ]* notation.
    - Can use pointer notation also, but that will cause more problems for you at this stage.

- Passing pointers to functions
  - Remember the syntax
  - Make sure to pass a pointer that is actually pointing to something when you call the function (like *&x* instead of just *x* for a integer variable *x*, or a pointer that is actually allocated something)
- Returning more than one value from a function
  - Return one value same as before using the return statement
  - For any other value to return, have one pointer for each as parameter
  - Decide whether a parameter has to be a pointer or not as told before
  - Inside the function, compute fully using local variables first. At the end, just before returning, assign properly to the pointer values. This will help you to debug errors faster as rest of the program is same as what you have done so far.

Assignments

Assignment 13

Consider an array A of integers. You have to write a C program to find the two numbers in A with the smallest absolute difference among all such pairs of number in A.

For example, if A = 2, 5, 0, 7, 3, then the two required numbers are 2 and 3 (with absolute difference = 1, which is smaller than the absolute difference of any other pair of numbers that you can choose in A).

- Write a C function

  int FindDiff(int *X, int n, int *x1, int *x2)

  that takes as parameters an integer array X with n elements, and does the following:
  - Returns the minimum absolute difference between any pair of numbers in X
  - Returns the two numbers with this minimum difference using x1 and x2
  - If there are more than one such pair of numbers, you can return any one

- Write a main function that does the following:
  - Read in an integer n
  - Dynamically allocate an integer array of size n using malloc()
  - Read in n integers in the array A
  - Call FindDiff() to find the minimum absolute difference
  - Print the difference, and the corresponding two numbers, with an appropriate message

- If the array is 30, 10, 5, 20, 45, then your program should print

The minimum absolute difference is 5 and the corresponding numbers are 10 and 5

- If the array is 30, 20, 5, 20, 35, then your program should print

The minimum absolute difference is 0 and the corresponding numbers are 20 and 20

Assignment 14

- Consider a string of characters with no space between them (for example, a word in English), terminated by '\0'. You will write a C program to count and print the maximum consecutive occurrence of the same character, and the corresponding character. If there are multiple characters that occur consecutively the same number of times, you can print any one such character. For example,

- If the string is “aabbbccbbd”, then ‘a’ occurs two times consecutively, ‘b’ occurs maximum of three times consecutively, ‘c’ occurs two times consecutively, and ‘d’ occurs one time consecutively. So you should output the character ‘b’ and 3, since ‘b’ occurs the maximum number of times consecutively among all characters

- If the string is “pdslab”, there is no character that occurs more than once consecutively. So in this case, you should output the character ‘p’ and 1 (meaning that p occurred the max times (=1) consecutively). Note that you can also output any of the other characters in the string (instead of ‘p’), as they all occur exactly once

- Write a C function int CountChar(char C[ ], char *mc) that takes a string C of characters (terminated by '\0') and returns the count of the maximum number of consecutive occurrence of the same character. It also puts the character with this maximum in mc.
- Write a main function that does the following:
  - Reads in an integer n
  - Dynamically allocates a character array of size n
  - Read in a string using %s in scanf (assume size of string will be < n)
  - Call CountChar() to find the maximum number of consecutive occurrence of the same character in the string.
  - Prints the maximum count and the character with the maximum count.

- If the string read is “aabbbccbbd”, your program should print

The character with maximum consecutive occurrence is b, with count 3

- If the string read is “aabbbccc”, your program should print either

The character with maximum consecutive occurrence is b, with count 3

Or

The character with maximum consecutive occurrence is c, with count 3

as both b and c occur 3 times consecutively

Teaser problem (not to be submitted)

- Let A be an n × n two-dimensional array in which all the rows and all the columns are sorted in ascending (non-decreasing) order from smaller to larger indices (that is left to right and top to bottom). Your task is to efficiently sort all the elements present in the array, in the same array.