Embedded Systems Lab

CPE 325-02

Data Types in C

By: David Thornton

Lab Date: August 27, 2020

Lab Due: September 7, 2020

Demonstration Due: September 7, 2020

Introduction

This lab reviews basic C programming skills as well as an in depth look at data types. Two concepts from MA 244, INTRO TO LINEAR ALGEBRA, are implemented: dot product and matrix multiplication. This lab also allows the student to expand on the assignment, if desired.

Theory

Topic 1: Different data types

There are several data types in C. There are signed data types, unsigned data types, and floating point types. Signed data types include negative and positive values, unsigned data types only include positive values as well as zero, and floating point types include positive decimal values of various precisions.

Topic 2: Size limit of data types.

Each data type has a size limit based on the number of bits each type uses in memory. The number of values that can be stored is equal to $2^{\#\text{ of bits}}$. There are 8 bits in a byte.

Topic 3: Endianness

“A big-endian system stores the most significant byte of a word at the smallest memory address and the least significant byte at the largest. A little-endian system, in contrast, stores the least-significant byte at the smallest address.” - Endianness. MSP430 is little endian.

Lab Assignment

1. Write a C program that will print the sizes and ranges of common data types char, short int, int, long int, long long int, unsigned char, unsigned short int, unsigned int, unsigned long int, unsigned long long int, float, and double. Your program’s output should be like the following:

Data Type	Size (in bytes)	Minimum	Maximum
char	1	0	255
short int	2	-32768	32767
(additional data types)			

Note: You should use definitions given in the limits.h and float.h header files for the ranges of data types. For float and double, display positive minimum value in Minimum column.

2. On a paper, using a pen, compute the maximum and minimum values of a data-type whose size is 2 bytes. Perform this computation considering the data-type to be (a) Unsigned data-type (b) Signed data-type. Take a picture of your solution and paste it in your report document. Alternately, you can solve in word document which can be added to the submission report.

In the list of data type that you printed in Q1, which data type is 2-bytes. Does your maximum and minimum values match with your output in Q1?

3. Write a C program that declares and initializes two integer arrays x and y. They should have at least 5 elements. You are required to compute a dot product of these two arrays. For example, one sample run output can look as following:

Input Array X: [-1 2 5 3 -5 6]
Input Array Y: [-7 8 23 13 23 28]
Dot Product is: 230

4. (Bonus: Up to 5 pts) Write a C program that performs the matrix multiplication on two 8x8 matrices. Display your input matrices and final result matrix.

Flow Chart

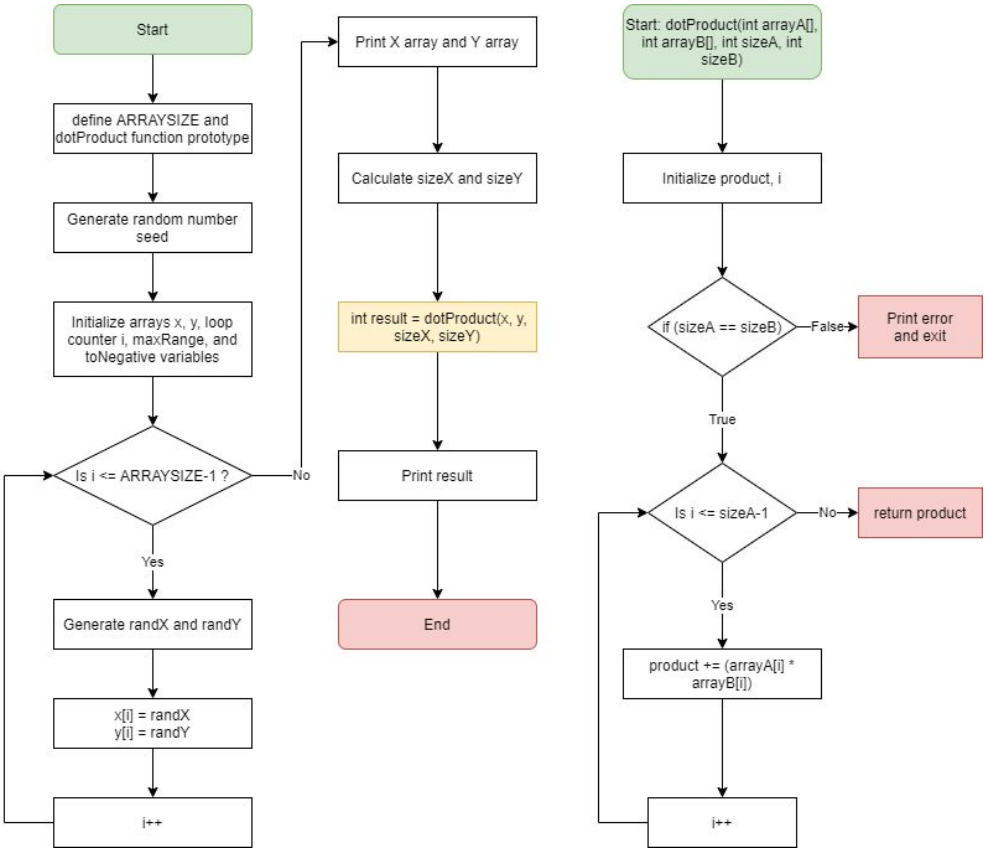

Figure 1:
Lab 2
Q 3

Observations

All programs work as intended, and the scope of some programs was expanded.

Conclusion

This lab expanded my understanding of dot product, matrix multiplication, as well as increased my knowledge of the C programming language, Code Composer Studio, and how to run and debug a program on the MSP-EXP430F5529LP. The most significant issues I faced during this lab were related to the extra credit, Q4. Implementing the 3 layer nested for loop took a while to figure out. I also struggled with Q3 figuring out the size of an array.

Demo link

Appendix

Appendix 1: Lab2_Q1.c

```
/*-------------------------------------------------------
* File: Lab2_Q1.c
* Description: C program will print the sizes and ranges
*              of common data types.
* Input: float.h, limits.h
* Output: Data type sizes and ranges
* Author: David Thornton
* Lab Section: 2
* Date: September 7, 2020
* *-------------------------------------------------------*/
#include <stdio.h> // For printf
#include <msp430.h>
#include <limits.h> // For min/max variables
#include <float.h> // For min/max variables

#define ZERO 0 // Eliminate magic numbers

int main()
{
   WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer to prevent time out reset
   const char hyphen = '-'; int i = 0;
   for(i = 1; i <= 76; i++) { putchar(hyphen); } printf("\n"); // Prints 76 '-'
   printf("| Data Type          | Bytes |        Minimum       |       Maximum        |\n");
```

```
    for(i = 1; i <= 76; i++) { putchar(hyphen); } printf("\n"); // Prints 76 '-'
    printf("| char               | %3d   | %20d | %20d |\n", sizeof(char), SCHAR_MIN,
SCHAR_MAX);
    printf("| short              | %3d   | %20hd | %20hd |\n", sizeof(short int), SHRT_MIN,
SHRT_MAX);
    printf("| int                | %3d   | %20d | %20d |\n", sizeof(int), INT_MIN, INT_MAX);
    printf("| long               | %3d   | %20ld | %20ld |\n", sizeof(long int), LONG_MIN,
LONG_MAX);
    printf("| long long          | %3d   | %20lld | %20lld |\n", sizeof(long long int), LLONG_MIN,
LLONG_MAX);
    printf("| unsigned char      | %3d   | %20d | %20u |\n", sizeof(unsigned char), ZERO,
UCHAR_MAX);
    printf("| unsigned short     | %3d   | %20d | %20hu |\n", sizeof(unsigned short int), ZERO,
USHRT_MAX);
    printf("| unsigned int       | %3d   | %20d | %20u |\n", sizeof(unsigned  int), ZERO,
UINT_MAX);
    printf("| unsigned long      | %3d   | %20d | %20lu |\n", sizeof(unsigned long int), ZERO,
ULONG_MAX);
    printf("| unsigned long long | %3d   | %20d | %20llu |\n", sizeof(unsigned long long int),
ZERO, ULLONG_MAX);
    printf("| float              | %3d   | %20g | %20g |\n", sizeof(float), FLT_MIN, FLT_MAX);
    printf("| double             | %3d   | %20g | %20g |\n", sizeof(double), DBL_MIN, DBL_MAX);
    for(i = 1; i <= 76; i++) { putchar(hyphen); } // Prints 76 '-'
    return 0;
}

/* Calculate the range of the data type for a given size.
 * Number of Bytes * 8 = Number of Bits
 * Size = 2 ^ (bits)
 */
```

Appendix 2: Lab2_Q2.txt

1. Unsigned data type
    a. 1 bytes = 8 bits
    b. 2 bytes = 16 bits
    c. $2^{16} = 65536$
    d. Since unsigned data types start at 0 and are only positive, we simply include the range up to and including 65536.
    e. Therefore, an unsigned data type contains values in the range
    f. [0, 655365]

2. Signed data type
    a. 1 bytes = 8 bits
    b. 2 bytes = 16 bits

c. $2^{16} = 65536$
d. Because this is a signed data type, we must include an equal number of negative and positive values.
e. 65536 / 2 = 32768
f. For negative numbers, this is easy. -1, -2, -3, … -32768.
g. For positive numbers, we must include 0 although it is neither explicitly positive nor negative.
h. 32768 -1 = 32767
i. Therefore, a signed data type contains values in the range
j. [-32768, 32767]

The data types that are 2 bytes are short, int, unsigned short, and unsigned int. The values found here align with the values found in question 1 for the above data types.

Appendix 3: Lab2_Q3.c

```
/*-------------------------------------------------------
* File: Lab2_Q3.c
* Description: This programs computes the dot product
* Input: Two random int arrays of size ARRAYSIZE
* Output: Dot product
* Author: David Thornton
* Lab Section: 2
* Date: September 7, 2020
* *-------------------------------------------------------*/
#include <time.h> // For srand()
#include <stdio.h> // For printf
#include <msp430.h>
#include <stdlib.h> // For srand()

#define ARRAYSIZE 13 // With a size of 13 and range of [-50, 50],
                     // this program will never reach INT_MAX or INT_MIN.
                     // 50 x 50 (max result) = 2500. 2500 x 13 (array size) = 32500. 32500 <
INT_MAX (32767).
                     // The same calculation can be done with negative numbers for INT_MIN.
                     // ARRAYSIZE and maxRange can be adjusted based on design requirements.

int dotProduct(int arrayA[], int arrayB[], int sizeA, int sizeB); // Function prototype

int main()
{
   WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer to prevent time out reset
   srand(time(NULL)); // Random number seed based on system time
   int x[ARRAYSIZE]; int y[ARRAYSIZE]; unsigned int i;
   const unsigned int maxRange = 101; // Sets the range to [0, 100]
   const unsigned int toNegative = maxRange/2;
```

```
    // Shifts the range down to include negative numbers. This causes some uneven
distribution.

    for(i = 0; i <= ARRAYSIZE-1; i++) // This loop populates the two arrays with random
numbers in the range [-50, 50]
    {
        int randX = rand() % maxRange - toNegative;
        int randY = rand() % maxRange - toNegative;
        x[i] = randX;
        y[i] = randY;
    }
    printf("X Array: \n");
    for(i = 0; i <= ARRAYSIZE-1; i++)
    {
        printf("Value %02d: %3d", i+1, x[i]);
        printf("\n");
    }

    printf("\nY Array: \n");
    for(i = 0; i <= ARRAYSIZE-1; i++)
    {
        printf("Value %02d: %3d", i+1, y[i]);
        printf("\n");
    }

    unsigned int sizeX = (sizeof(x)/sizeof((x)[0]));
    unsigned int sizeY = (sizeof(y)/sizeof((y)[0]));

    int result = dotProduct(x, y, sizeX, sizeY);
    printf("\nThe dot product is: %d", result);
    return 0;
}

int dotProduct(int arrayA[], int arrayB[], int sizeA, int sizeB)
{
    int product = 0; int i = 0;
    if (sizeA == sizeB) // This size check is not needed in this case, as array size is a defined
constant ARRAYSIZE.
                        // I am still including it as a best practice and a failsafe.
    {
        for (i = 0; i <= sizeA-1; i++) // Loop to calculate dot product
        {
            product += (arrayA[i] * arrayB[i]);
        }
        return product;
    }
    else if (sizeA != sizeB)
```

```
    {
        printf("Array sizes must match. Exiting program.");
        exit(0);
    }
    return 0;
}
```

Appendix 4: Lab2_Q4.c

```
/*-------------------------------------------------------
* File: Lab2_Q4.c
* Description: This programs performs matrix multiplication
*              on two 8x8 matrices.
* Input: Two random 8x8 matrices
* Output:
* Author: David Thornton
* Lab Section: 2
* Date: September 7, 2020
* *-------------------------------------------------------*/
#include <time.h> // For srand()
#include <stdio.h> // For printf
#include <msp430.h>
#include <stdlib.h> // For srand()

#define ARRAYSIZE 8 // This program only works on square matrices.

void printMatrix(int anyMatrix[][]); // Function prototype

int main()
{
    WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer to prevent time out reset
    srand(time(NULL)); // Random number seed based on system time
    int x[ARRAYSIZE] [ARRAYSIZE]; int y[ARRAYSIZE] [ARRAYSIZE]; int result[ARRAYSIZE]
[ARRAYSIZE];
    unsigned int i,j,k; // Loop counters
    const unsigned int maxRange = 101; // Sets the range to [0, 100]
    const unsigned int toNegative = maxRange/2; // Shifts the range down to include negative
numbers. This causes some uneven distribution.

    // This loop populates the two arrays with random numbers in the range [-50, 50]
    for(i = 0; i <= ARRAYSIZE-1; i++)
        {
            for(j = 0; j <= ARRAYSIZE-1; j++)
            {
                int randX = rand() % maxRange - toNegative;
                int randY = rand() % maxRange - toNegative;
```

```
            x[i][j] = randX;
            y[i][j] = randY;
         }
      }
   // This loop calculates the value for each cell of the result matrix
   for(i = 0; i <= ARRAYSIZE-1; i++)
   {
      for(j = 0; j <= ARRAYSIZE-1; j++)
      {
         result[i][j] = 0; // Set everything to 0 as a failsafe
         for(k = 0; k <= ARRAYSIZE-1; k++)
         {
            result[i][j]+=x[i][k]*y[k][j];
         }
      }
   }
   printf("A:\n");
   printMatrix(x);
   printf("\nB:\n");
   printMatrix(y);
   printf("\nAB:\n");
   printMatrix(result);
}

// This function prints a 2 dimensional array
void printMatrix(int anyMatrix[ARRAYSIZE][ARRAYSIZE])
{
   unsigned int i,j; // Loop counters
   for(i = 0; i <= ARRAYSIZE-1; i++)
      {
         for(j = 0; j <= ARRAYSIZE-1; j++)
         {
            printf("%d\t", anyMatrix[i][j]);
         }
         printf("\n");
      }
}
```