CPE 325: Intro to Embedded Computer System

Lab02

Data Types in C

Submitted by: Hannah Kelley

Date of Experiment: September 3, 2025

Report Deadline: September 10, 2025

Demonstration Deadline: September 10, 2025

Theory

Topic 1: Different Data Types

a) In C programming date types determine the kind of value a variable can hold and how much memory is allocated for it. Integer values of various sizes can be stored in char, short, int, long, and long long variables. Decimal values can be stored in float and double variables. Each data type has two variations—signed and unsigned—which determine whether or not it can represent negative values. Declaring the correct data type is important for optimizing a program's efficiency and accuracy.

Topic 2: Size Limit of Data Types

a) In computer systems, the size of a data type is defined by the number of bytes it occupies in memory, which in turn determines the range of values it can store. For signed integer types, one bit is dedicated to representing the sign (positive or negative), reducing the maximum magnitude that can be expressed. In contrast, unsigned integer types use all bits to represent the magnitude, allowing for a greater range of positive values but excluding negative numbers. Floating-point types, such as float and double, are represented according to the IEEE 754 standard, which divides the available bits into a sign, an exponent, and a mantissa, with the specific allocation depending on the precision of the data type.

Topic 3: Endianness

a) Endianness refers to the way in which a computer stores and interprets data in memory. In little-endian systems, such as the one in this lab, the least significant byte is stored at the lowest memory address. In big-endian systems, the most significant is placed at the lowest memory address.

Results & Observation

Program 1:

Program Description:

This program prints the sizes and ranges of common C data types by using macros contained in the limits.h and float.h libraries. The results are displayed in a formatted table.

Program Output:

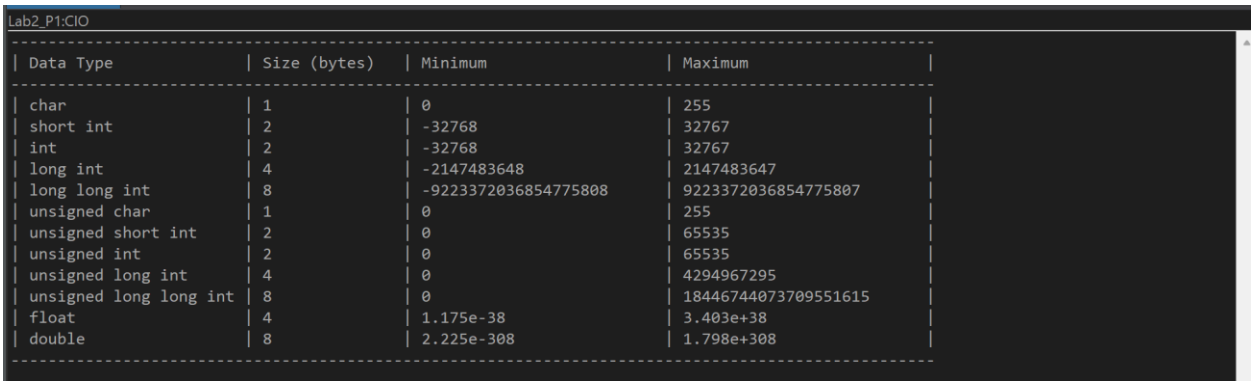

Figure 01: Program 1 Output

Report Questions:

1. **How are format specifiers used in your Q1 program?**
   In this program, format specifiers are used with the printf function to correctly display values of different data types. Each data type requires its own specifier so that the stored value is interpreted and printed properly. By carefully selecting the correct format specifier, the program ensures that all sizes, minimums, and maximums for different data types are displayed in a clear and accurate way.

Problem 2:

Problem Description:

Manually calculate the maximum and minimum values for all 4-byte data types found in program 1.

Problem Calculations:

The 4-byte data types identified in the output table of program 1 are signed and unsigned long ints and float. The table below summarizes the findings of this section.

Table 01: Data Type Maximum and Minimum Values

Data Type	Maximum Value	Minimum Value
Signed long int	$2{,}147{,}483{,}647$	$-2{,}147{,}483{,}648$
Unsigned long int	$4{,}294{,}967{,}295$	0
Float (32 bits)	3.403×10^{38}	1.175×10^{-38}

We know that the maximum value of a signed data type is $2^{n-1} - 1$ and the minimum value is -2^{n-1} where n is the number of bits each data type has allocated. We subtract one from the exponent because the sign bit takes up the first bit of a number and cannot be used to represent the magnitude of the number. We subtract one from the maximum value because zero is represented as a positive number. Using these formulas we know the maximum value of a signed long int is $2^{32-1} - 1 = 2{,}147{,}483{,}647$. The minimum value is $-2^{32-1} = -2{,}147{,}483{,}648$.

For unsigned data types the maximum value is 2^{n-1}. In this case we do not need to subtract one from the exponent since all the bits allocated for the value are used to stored the value itself and not the sign. The maximum value of an unsigned long int is $2^{32-1} = 4{,}294{,}967{,}295$. The minimum value is zero.

Floats are also known as single precision floating point numbers. From the IEEE-754 standard we know that 32 bit float use 1 bit to represent the sign, 8 bits to represent the exponent, and the remaining 23 bits to represent the mantissa. To get the highest possible number the sign bit will be 0 and the exponent bits will be 1111_1110. The exponent value 1111_1111 is reserved for special values like infinity and NaN. The largest possible mantissa will have 23 bits of all ones. Additionally, since all IEEE-754 numbers have a implicit leading 1 we have 24 bits of all ones. This will give us the largest 32 bit float number of $(-1)^0(2^{127})(2-2^{-23}) = 3.408x10^{38}$. To get the lowest possible positive value the sign bit will be 1, the exponent will be 1, and the mantissa will be 1. This will give us the smallest positive 32 bit float number of $(-1)^0(2^{-126})(1.0) = 1.175x10^{-38}$.

Program 3:

Program Description:

This program declares and initializes two integer arrays, x and y, each containing at least five elements. It computes a third array where each element is the result of subtracting the corresponding element of y from x. The program then displays the input arrays and the resulting output array in a clear, formatted manner.

Program Output:

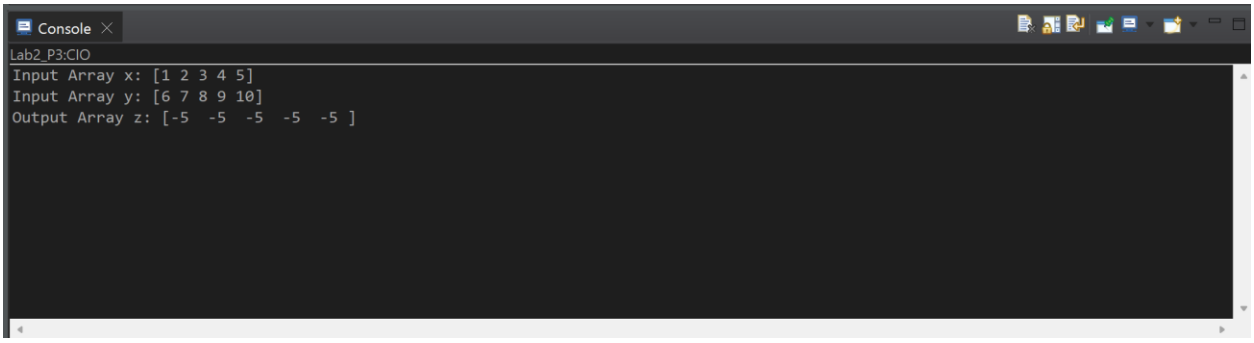

Figure 02: Program 3 Output

Report Questions:

1. **How are you calculating the output matrix in Q3?**
   In this program the output matrix is calculated by looping through all three matrices at the same time. The subtraction is calculated index-by-index and then stored in the output matrix before moving on to the next loop iteration.

Program Flowchart:

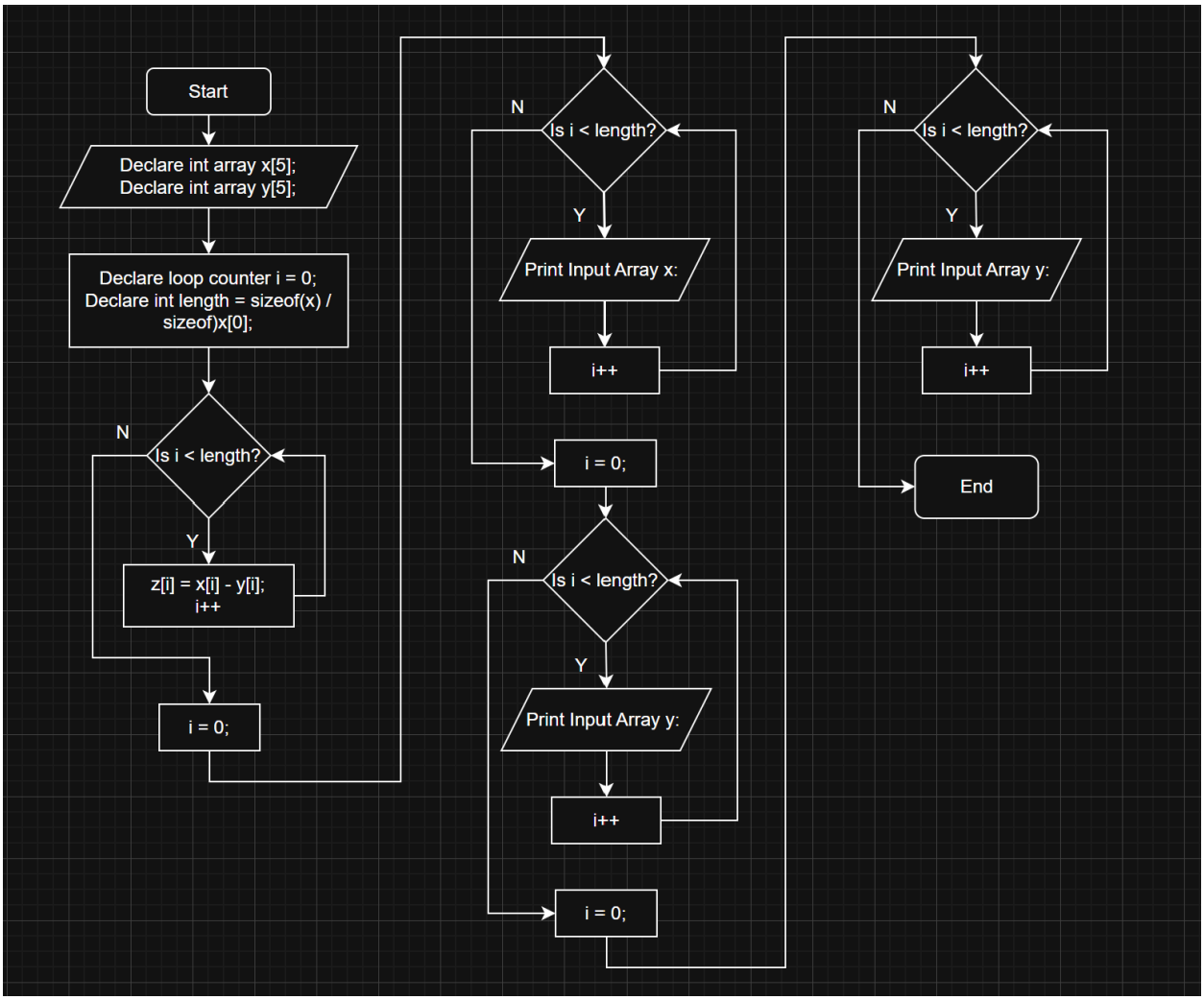

Figure 03: Program 3 Flowchart

Bonus Program:

Program Description:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Program Output:

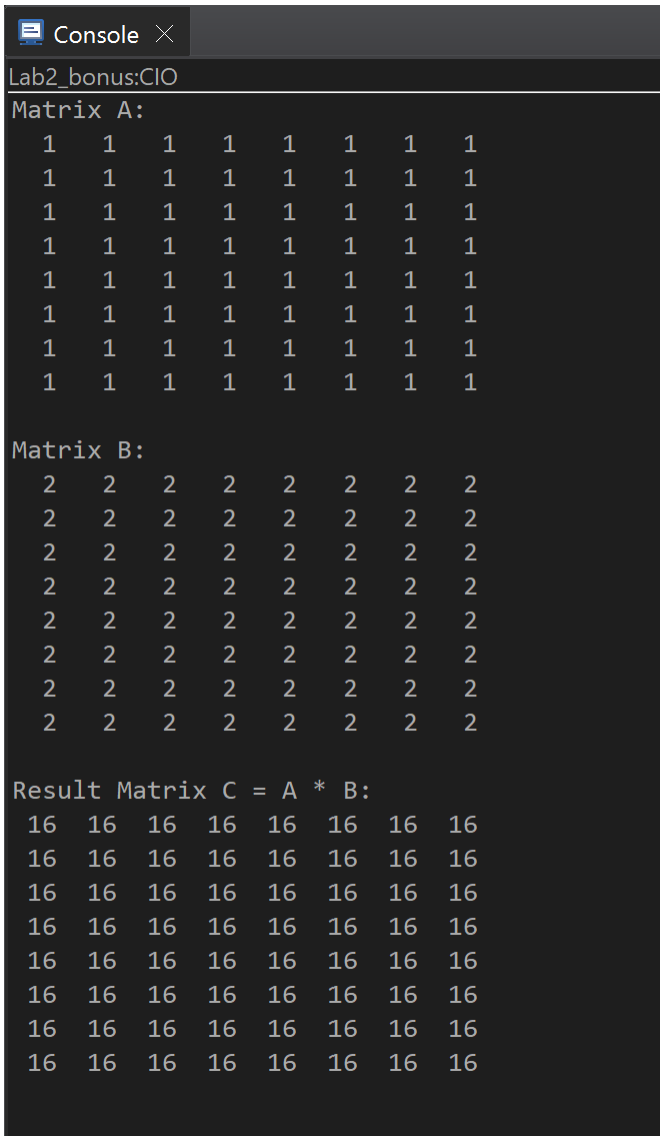

Figure 04: Bonus Program Output

Appendix

Table 02: Program 1 Source Code

```
/*-------------------------------------------------------------------------
 * File:        Lab02_bonus.c
 * Description: calculates and prints a table of min and max data type
values
 *
 * Board:       5529
 * Input:
 * Output:      a table of min and max values for every data type
 * Author:      Hannah Kelley
 * Date:        September 3, 2025
 *-------------------------------------------------------------------------*/
#include <msp430.h>
#include <stdio.h>
#include <limits.h>
#include <float.h>

void printTable() {
    printf("-----------------------------------------------------------n");
    printf("| %-22s | %-14s | %-25s | %-25s |\n",
           "Data Type", "Size (bytes)", "Minimum", "Maximum");
    printf("-----------------------------------------------------------\n");

    // Signed integer types, using macros
    printf("| %-22s | %-14zu | %-25d | %-25d |\n",
           "char", sizeof(char), CHAR_MIN, CHAR_MAX);
    printf("| %-22s | %-14zu | %-25d | %-25d |\n",
           "short int", sizeof(short), SHRT_MIN, SHRT_MAX);
    printf("| %-22s | %-14zu | %-25d | %-25d |\n",
           "int", sizeof(int), INT_MIN, INT_MAX);
    printf("| %-22s | %-14zu | %-25ld | %-25ld |\n",
           "long int", sizeof(long), LONG_MIN, LONG_MAX);
    printf("| %-22s | %-14zu | %-25lld | %-25lld |\n",
           "long long int", sizeof(long long), LLONG_MIN, LLONG_MAX);

    // Unsigned integer types, using macros
    printf("| %-22s | %-14zu | %-25u | %-25u |\n",
           "unsigned char", sizeof(unsigned char), 0, UCHAR_MAX);
    printf("| %-22s | %-14zu | %-25u | %-25u |\n",
           "unsigned short int", sizeof(unsigned short), 0, USHRT_MAX);
    printf("| %-22s | %-14zu | %-25u | %-25u |\n",
           "unsigned int", sizeof(unsigned int), 0, UINT_MAX);
    printf("| %-22s | %-14zu | %-25lu | %-25lu |\n",
           "unsigned long int", sizeof(unsigned long), 0UL, ULONG_MAX);
    printf("| %-22s | %-14zu | %-25llu | %-25llu |\n",
           "unsigned long long int", sizeof(unsigned long long), 0ULL,
ULLONG_MAX);
    // Floating-point types, using macros
    printf("| %-22s | %-14zu | %-25.3e | %-25.3e |\n",
           "float", sizeof(float), FLT_MIN, FLT_MAX);
    printf("| %-22s | %-14zu | %-25.3e | %-25.3e |\n",
           "double", sizeof(double), DBL_MIN, DBL_MAX);
    printf("-----------------------------------------------------------\n");
    return; // end of table
```

```
}

int main(void) {
    // stop watchdog timer
    WDTCTL = WDTPW | WDTHOLD;
    printTable(); // call function table
    return 0; // end of program
}
```

Table 03: Program 3 Source Code

```
/*-------------------------------------------------------------------------
 * File:        Lab02_P3.c
 * Description: given two input arrays subtract them, store in output array,
 *              and print the output
 *
 * Board:       5529
 * Input:       two int arrays of min size 5
 * Output:      int array z = x - y
 * Author:      Hannah Kelley
 * Date:        September 3, 2025
 *-------------------------------------------------------------------------*/
#include <msp430.h>
#include <stdio.h>

int main(void) {
    // stop watchdog timer
    WDTCTL = WDTPW | WDTHOLD;
    // hard-coded input arrays
    int x[] = {1, 2, 3, 4, 5};
    int y[] = {6, 7, 8, 9, 10};
    // output array
    int z[5];
    int i;
    int length = sizeof(x) / sizeof(x[0]);
    // loop through the input arrays and subtract
    for (i = 0; i < length; i++) {
        z[i] = x[i] - y[i];
    }
    // print input array x
    printf("Input Array x: [");
    for (i = 0; i < length; i++) {
        printf("%d", x[i]);
        if (i < length - 1) {
            printf(" ");
        }
    }
    printf("]\n");
    // print input array y
    length = sizeof(y) / sizeof(y[0]);
    printf("Input Array y: [");
    for (i = 0; i < length; i++) {
        printf("%d", y[i]);
        if (i < length - 1) {
            printf(" ");
        }
    }
```

```
    printf("]\n");
    // print output array z
    length = sizeof(z) / sizeof(z[0]);
    printf("Output Array z: [");
    for (i = 0; i < length; i++) {
        printf("%d ", z[i]);
        if (i < length - 1) {
            printf(" ");
        }
    }
    printf("]\n");
    return 0; // end of program
}}
```

Table 04: Bonus Program Source Code

```
/*-------------------------------------------------------------------------
 * File:        Lab02_bonus.c
 * Description: perform matrix multiplication on 2 8x8 matrices
 *
 * Board:       5529
 * Input:       two 8x8 arrays A and B
 * Output:      one 8x8 array C
 * Author:      Hannah Kelley
 * Date:        September 3, 2025
 *-------------------------------------------------------------------------*/
#include <msp430.h>
#include <stdio.h>

#define SIZE 8

int main(void) {
        // stop watchdog timer
        WDTCTL = WDTPW | WDTHOLD;
        int MatA[SIZE][SIZE], MatB[SIZE][SIZE], MatC[SIZE][SIZE];
        int i, j, k;

        // Initialize MatA with 1's and MatB with 2's
        i = 0;
        j = 0;
        for (i = 0; i < SIZE; i++) {
            for (j = 0; j < SIZE; j++) {
                MatA[i][j] = 1;
                MatB[i][j] = 2;
            }
        }
        // Perform matrix multiplication: MatC = MatA * MatB
        i = 0; j = 0;
        for (i = 0; i < SIZE; i++) {
            for (j = 0; j < SIZE; j++) {
                MatC[i][j] = 0;
                for (k = 0; k < SIZE; k++) {
                    MatC[i][j] += MatA[i][k] * MatB[k][j];
                }
            }
        }
        // Print MatA
```

```
        i = 0; j = 0;
        printf("Matrix A:\n");
        for (i = 0; i < SIZE; i++) {
            for (j = 0; j < SIZE; j++) {
                printf("%3d ", MatA[i][j]);
            }
            printf("\n");
        }
        // Print MatB
        i = 0; j = 0;
        printf("\nMatrix B:\n");
        for (i = 0; i < SIZE; i++) {
            for (j = 0; j < SIZE; j++) {
                printf("%3d ", MatB[i][j]);
            }
            printf("\n");
        }
        // Print Result MatC
        i = 0; j = 0;
        printf("\nResult Matrix C = A * B:\n");
        for (i = 0; i < SIZE; i++) {
            for (j = 0; j < SIZE; j++) {
                printf("%3d ", MatC[i][j]);
            }
            printf("\n");
        }
       return 0;
}
```