CPE 325: Intro to Embedded Computer System

Lab02

Common Data Types and Array Averages

Submitted by: Gianna Foti

Date of Experiment:______2 September 2024_______________

Report Deadline:_______3 September 2024________________

Lab Section: ______CPE353-02______

Demonstration Deadline: _____ 3 September 2024___________

Introduction

In this lab, we had two programs to write. The first program required us to write a C program that would print the sizes and ranges of common data types. We were to do this for 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. We were to output in a specified way that showed the date type, size in bytes, as well as the minimum and maximum. In between programs, we were to Compute the maximum and minimum values of a data type whose size is 4 bytes by hand. For the second program, we were to write a C program that declares and initializes two integer arrays x and y of at least 5 elements. Then we had to compute the float array z such that each element of z was the average of the corresponding elements in x and y. For the bonus, we were to write a C program that performs matrix multiplication on arrays A and B and prints the resulting array. Array A holds all one value, and array B holds all two values.

Theory Topics

1. Different Data Types
    a. There are 10 main data types examined in this lab: char, short int, int, long int, unsigned int, unsigned long int, long long int, unsigned long long int, float, and double.
    b. They differ in the typical number of bytes they occupy in memory as well as their possible range of values. For these reasons, different data types have varying use cases.
    c. The data type char is often used to represent ASCII characters whereas float is used to represent decimal numbers with limited precision.
    d. Unsigned Types represent nonnegative numbers.
    e. Signed Types represent negative and positive numbers.
2. Size Limit of Data Types
    a. The size limit of data types is defined by the amount of memory allocated to store a specific data type depending on the compiler, programming language, and system architecture.
    b. Eight bits equal one byte which means a 32-bit system will be 4 bytes
    c. Let n be the number of bits. A byte is 8 bits.
        i. For signed integers, the range goes from -2^(n-1) to 2^(n-1) - 1.

        ii. For unsigned data types, the range goes from 0 to 2^n - 1.
        iii. Long data types generally double the number of bytes

3. Endianness
    a. Two types of endianness
        i. Big Endian and Little Endian
            1. Big Endian: stores the most significant bit
            2. Little Endian: stores least significant bit
        ii. The MSP430 uses Little Endian, meaning the least significant byte (LSB) is stored at a lower address or first in memory.
        iii. Big Endian, the most significant byte (LSB) would be stored at a lower address or first in memory.

Results & Observation

Part 1:

Program Description:

This program prints the sizes and ranges of common data types. The program does this for the 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. The program's output is shown in a specified way that shows the date type, size in bytes, as well as the minimum and maximum.

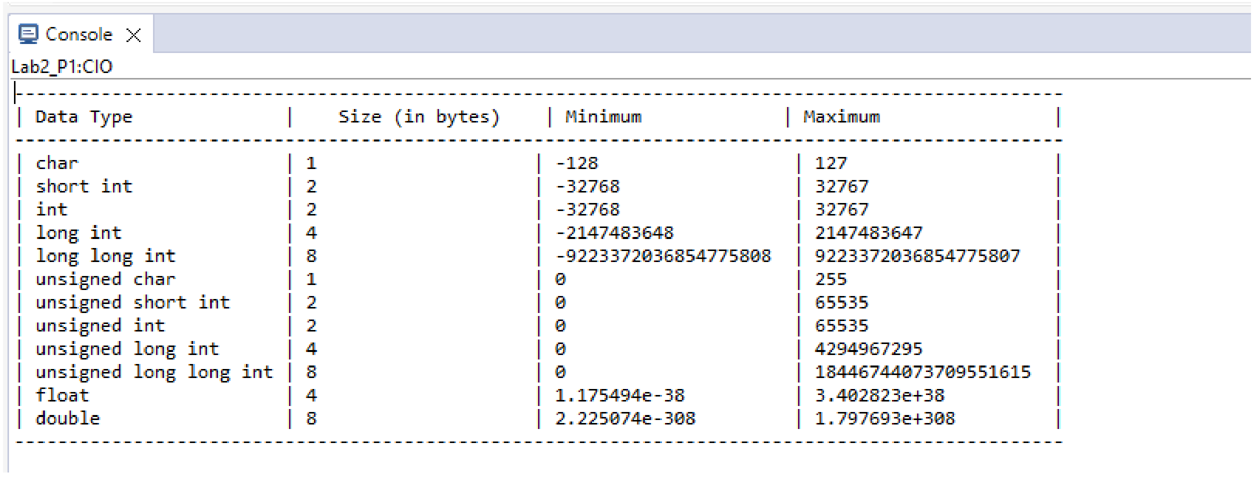
Process:

1. I made a printf statement that set up the table of my output with the specified categories.
2. For each date type, I found the maximum and minimum ranges using the header files, "limits.h" and "float.h".
    a. This was done for 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.
3. I used the sizeof operator to get the size of each data type.

4. I used format specifiers like %d, %zu, %ld, and others to control how the output is formatted when printing different types of data.

5. End

Program Output:



Console ✕

Lab2_P1:CIO

```
|-------------------------------------------------------------------------------------------------
| Data Type               |    Size (in bytes)    | Minimum               | Maximum               |
-------------------------------------------------------------------------------------------------
| char                    | 1                     | -128                  | 127                   |
| short int               | 2                     | -32768                | 32767                 |
| int                     | 2                     | -32768                | 32767                 |
| long int                | 4                     | -2147483648           | 2147483647            |
| long long int           | 8                     | -9223372036854775808  | 9223372036854775807   |
| unsigned char           | 1                     | 0                     | 255                   |
| unsigned short int      | 2                     | 0                     | 65535                 |
| unsigned int            | 2                     | 0                     | 65535                 |
| unsigned long int       | 4                     | 0                     | 4294967295            |
| unsigned long long int  | 8                     | 0                     | 18446744073709551615  |
| float                   | 4                     | 1.175494e-38          | 3.402823e+38          |
| double                  | 8                     | 2.225074e-308         | 1.797693e+308         |
-------------------------------------------------------------------------------------------------
```

Figure 1: Program 1 Output

Report Questions:

No lab questions for Program 1.

Part 2:

Calculation Description:

For this part, we were to compute the maximum and minimum values of a date-type whose size is 4 bytes by hand or in a doc. I copied my work onto a doc and is listed below. We were to perform this computation considering the date type to be unsigned and signed.

Unsigned:

4 bytes = 32 bits

Maximum value:

- All bits set to 1: 11111111 11111111 11111111 11111111
- In decimal: 2^32 - 1 = 4,294,967,295

Minimum value:

- All bits set to 0: 00000000 00000000 00000000 00000000

- In decimal: 0

Signed:

4 bytes = 32 bits

Available bits for magnitude: 31

Maximum value:

- In decimal: 2^31 - 1 = 2,147,483,647

Minimum negative value:

- Sign bit is 1, all other bits are 0: 10000000 00000000 00000000 00000000
- In decimal: -2^31 = -2,147,483,648

Report Questions:

1. **In the list of data types that you printed in Q1, which data types are 4-bytes?**
    a. Long int, unsigned long int, and float are 4 bytes.
2. **Do your minimum and maximum values match with your outputs in Q1?**
    a. Yes, it matches the values of long int.

Part 3:

Program Description:

This program computes a float array, z, from the average of the data from two integer arrays x and y. This C program declares and initializes two integer arrays x and y of at least 5 elements.

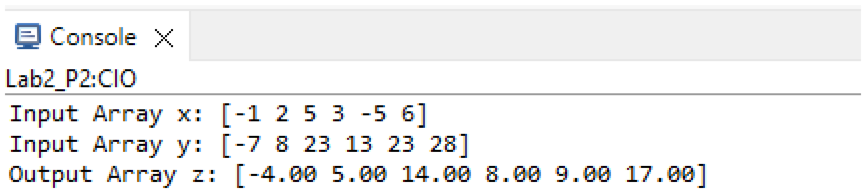
Process:

1. I initialized arrays x and y and found the sizes of both arrays.
2. I checked for the minimum array size to check if the arrays had at least 5 elements.
    a. If not, an error message printed and the program exited.
3. Declared the z_result array
    a. I hardcoded the size to 6 because it was only necessary to have at least 5 elements.
    b. Each element will store the average of the corresponding elements from ‘x’ and ‘y’
4. I then computed the averages and stored them in z_result.
    a. I did this by looping through each element in arrays x and y and computing the average.
5. I printed the input arrays and then printed the output array.

a. This involved looping through each array and printing the elements in a formatted manner.

b. The averages were formatted to two decimal places like shown in the example.

Program Output:

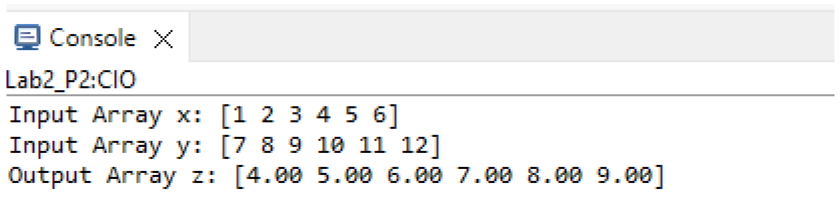


```
Console ×
Lab2_P2:CIO
Input Array x: [-1 2 5 3 -5 6]
Input Array y: [-7 8 23 13 23 28]
Output Array z: [-4.00 5.00 14.00 8.00 9.00 17.00]
```

Figure 2: Program 2 Output

Report Questions:

1. **What were the key considerations while solving the question?**
   a. A key consideration was using correct data types. The average of two integers was computed as a float to preserve decimal precision. This involved casting the integer elements to float before performing the division.
2. **Does your program work with other data values?**
   a. Yes, the program works for other data values. Below is an example of other data values with the inputs and outputs shown.



```
Console ×
Lab2_P2:CIO
Input Array x: [1 2 3 4 5 6]
Input Array y: [7 8 9 10 11 12]
Output Array z: [4.00 5.00 6.00 7.00 8.00 9.00]
```

Program Flowchart:

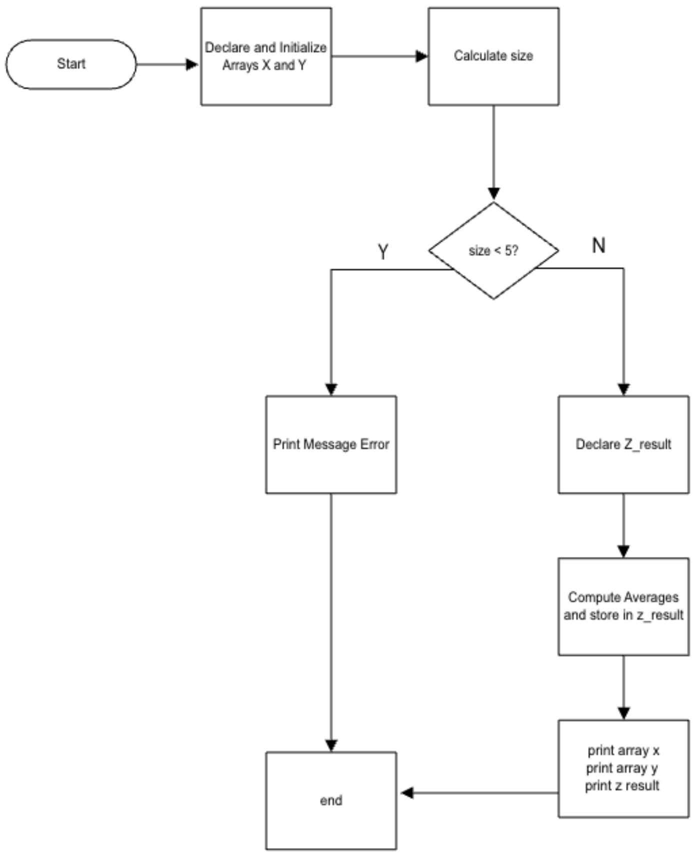


Figure 3: Program 2 Flowchart

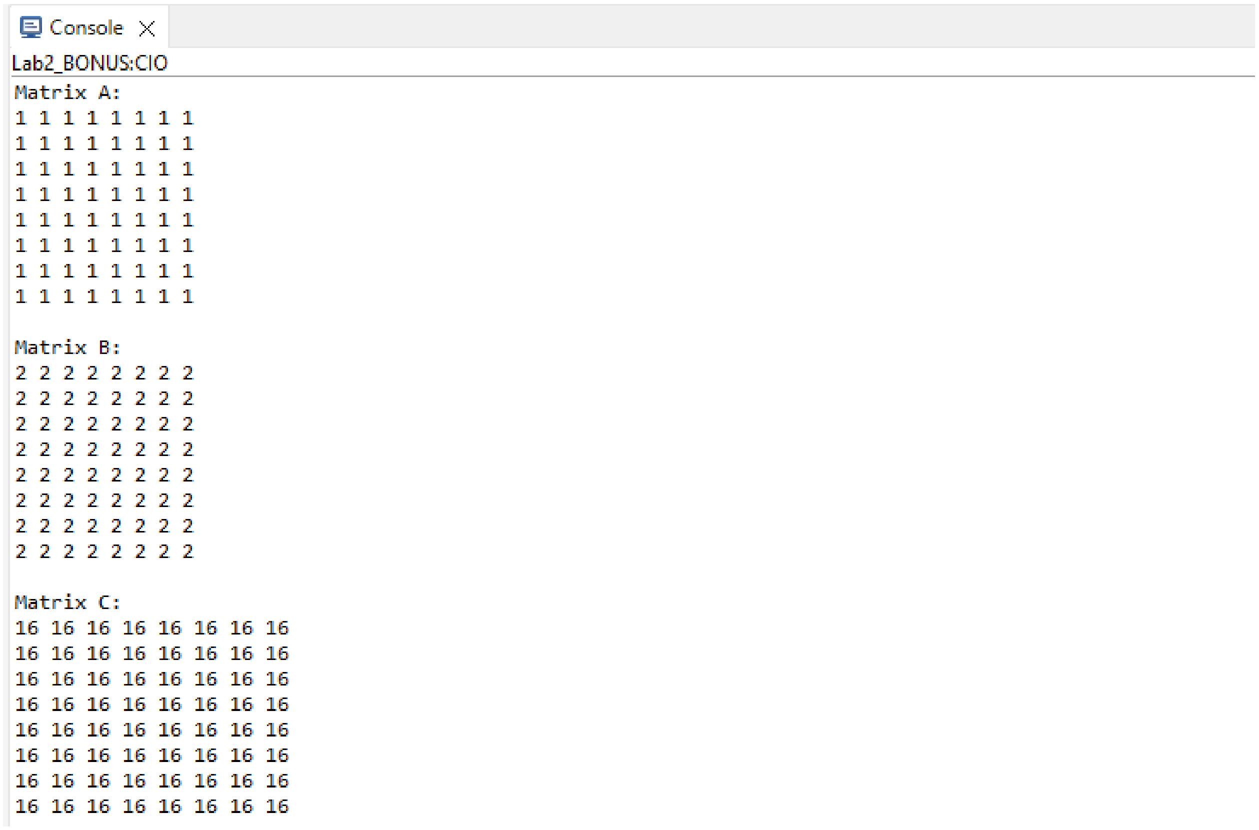
Bonus:

Program Description:

For the bonus, we were to write a C program that performs matrix multiplication on arrays A and B and prints the resulting array. Array A holds all one value, and array B holds all two values.

Process:

1. I started by initializing an array, A, and an array, B. A was supposed to be filled with all 1's while B was to be filled with all 2's.
2. I then found the size of the rows and columns of both A and B. I then printed those matrices.
3. I computed the dot product of A and B.
4. Next, I created a zresult which holds the results of A and B's multiplication.
5. I then printed this result.



Console ×

Lab2_BONUS:CIO

```
Matrix A:
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1

Matrix B:
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2

Matrix C:
16 16 16 16 16 16 16 16
16 16 16 16 16 16 16 16
16 16 16 16 16 16 16 16
16 16 16 16 16 16 16 16
16 16 16 16 16 16 16 16
16 16 16 16 16 16 16 16
16 16 16 16 16 16 16 16
16 16 16 16 16 16 16 16
```

Figure 4: Bonus Program Output

Conclusion

In conclusion, this lab taught me about data types, size limits, endianness, and how they interact with computer architecture. This laboratory assignment provided valuable insights into working with arrays, type casting, and arithmetic operations in the C programming language. In Part 1, I developed a program that focused on understanding and utilizing format specifiers while determining and printing the sizes and limits of various data types such as `char`, `int`, `long int`, `float`, and `double`. This exercise reinforced my knowledge of how data types interact with memory and how to effectively use format specifiers for clear and precise output. In Part 2, I applied my understanding of type casting by computing the average of corresponding elements from two integer arrays and storing the results in a float array. This part of the assignment emphasized the importance of using the correct data types to ensure accurate calculations and prevent data loss. Explicit type casting from integer to float was necessary to maintain precision in the computed averages. Overall, this laboratory assignment enhanced my understanding of data types, their limits, and the significance of type casting in C. Additionally, it provided practical experience in using format specifiers and handling arrays, which are fundamental skills in C programming. The biggest issue I faced was in the second program I had to write. I could not figure out how to store my results in the z_result. I then figured it out by initializing it as float z_result[6]. This solved my problem and my program worked as it should. In all, this was a good learning experience and I did not struggle too much.

Appendix

Table 1: Program 1 source code

```
/*------------------------------------------------------------------------------
 * File:        Lab02_P1.c
 * Description: This program prints 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
 * Board:       5529
 * Input:       none
 * Output:      shows the date type, size in bytes, as well as the minimum and maximum
 * Author:      Gianna Foti, ghf0004@uah.edu
 * Date:        September 3, 2024
 *--------------------------------------------------------------------------*/
#include <msp430.h>
#include <stdio.h>
#include <limits.h>
#include <float.h>

int main()
{

// Stop WatchDogTimer
   WDTCTL = WDTPW + WDTHOLD;
//
   int zero = 0;
   printf("----------------------------------------------------------------------------------------------------\n");
   printf("| Data Type              |    Size (in bytes)    | Minimum              | Maximum              |\n");
   printf("----------------------------------------------------------------------------------------------------\n");
   printf("| char                   | %-20zu | %-21d | %-21d |\n", sizeof(char), SCHAR_MIN, SCHAR_MAX);
   printf("| short int              | %-20zu | %-21d | %-21d |\n", sizeof(short int), SHRT_MIN, SHRT_MAX);
   printf("| int                    | %-20zu | %-21d | %-21d |\n", sizeof(int), INT_MIN, INT_MAX);
   printf("| long int               | %-20zu | %-21ld | %-21ld |\n", sizeof(long int), LONG_MIN, LONG_MAX);
   printf("| long long int          | %-20zu | %-21lld | %-21lld |\n", sizeof(long long int), LLONG_MIN, LLONG_MAX);
   printf("| unsigned char          | %-20zu | %-21u | %-21u |\n", sizeof(unsigned char), zero, UCHAR_MAX);
   printf("| unsigned short int     | %-20zu | %-21u | %-21u |\n", sizeof(unsigned short int), zero, USHRT_MAX);
   printf("| unsigned int           | %-20zu | %-21u | %-21u |\n", sizeof(unsigned int), zero, UINT_MAX);
   printf("| unsigned long int      | %-20zu | %-21u | %-21lu |\n", sizeof(unsigned long int), zero, ULONG_MAX);
   printf("| unsigned long long int | %-20zu | %-21u | %-21llu |\n", sizeof(unsigned long long int), zero, ULLONG_MAX);
   printf("| float                  | %-20zu | %-21e | %-21e |\n", sizeof(float), FLT_MIN, FLT_MAX);
   printf("| double                 | %-20zu | %-21e | %-21e |\n", sizeof(double), DBL_MIN, DBL_MAX);
   printf("----------------------------------------------------------------------------------------------------\n");

   return 0;
}
```

Table 2: Program 2 source code

```
/*------------------------------------------------------------------------------
 * File:        Lab02_P2.c
 * Description:  A C program that declares and initializes two integer arrays x and y
 *              of at least 5 elements. It computes a float array z where each element
 *              of z is the average of corresponding elements in x and y.
 * Board:       5529
 * Input:       Two integer arrays x and y of at least 5 elements.
 * Output:      The float array z, with each element being the average of corresponding
 *              elements in x and y.
 * Author:      Gianna Foti, ghf0004@uah.edu
 * Date:        September 3, 2024
 /*--------------------------------------------------------------------------------
 * File:        Lab02_P2.c
 * Description:  A C program that declares and initializes two integer arrays x and y
 *              of at least 5 elements. It computes a float array z where each element
 *              of z is the average of corresponding elements in x and y.
 * Board:       5529
 * Input:       Two integer arrays x and y of at least 5 elements.
 * Output:      The float array z, with each element being the average of corresponding
 *              elements in x and y.
 * Author:      Gianna Foti, ghf0004@uah.edu
 * Date:        September 3, 2024
 *---------------------------------------------------------------------------*/

#include <stdio.h>
#include <msp430.h>

int main()
{
   WDTCTL = WDTPW + WDTHOLD;
   // Declare and initialize arrays x and y
   int x[] = {-1, 2, 5, 3, -5, 6};
   int y[] = {-7, 8, 23, 13, 23, 28};


   // Calculate the number of elements in the arrays
   int size = sizeof(x) / sizeof(x[0]);

   // Ensure we have at least 5 elements
   if (size < 5)
   {
      printf("Error: Arrays must have at least 5 elements.\n");
      return 1;
   }

   // Declare array z to store the averages, hardcoded it to 6
   float z_result[6];

   // Compute averages and store in z
   int i;
   for (i = 0; i < size; i++)
   {
      z_result[i] = ((float)x[i] + (float)y[i]) / 2.0f;
   }

   // Print input arrays
   printf("Input Array x: [");
   int j;
   for ( j = 0; j < size; j++)
   {
      printf("%d", x[j]);
      if (j < size - 1)
         printf(" ");
   }
   printf("]\n");

   printf("Input Array y: [");
   int k;
   for (k = 0; k < size; k++)
   {
      printf("%d", y[k]);
      if (k < size - 1)
         printf(" ");
   }
   printf("]\n");

   // Print output array
   printf("Output Array z: [");
   int l;
   for (l = 0; l < size; l++)
   {
      printf("%.2f", z_result[l]);
      if (l < size - 1)
         printf(" ");
   }
   printf("]\n");

   return 0;
}
```

Table 3: Bonus source code

```
/*------------------------------------------------------------------------------
 * File:        Lab02_BONUS.c
 * Description: This program performs the matrix multiplication on two 8x8
 * matrices with all values 1 for matrix A and 2 for matrix B
 * Board:       5529
 * Input:       matrix a and matrix b
 * Output:      the result of the multiplication of a and b
 * Author:      Gianna Foti, ghf0004@uah.edu
 * Date:        September 3, 2024
 *---------------------------------------------------------------------------*/

#include <stdio.h>
#include <msp430.h>
#include <string.h>
#include <ctype.h>
#include <limits.h>
#include <float.h>


int main(void)
{
   // stop watchdog timer
   WDTCTL = WDTPW | WDTHOLD;
   int i;
   int k;
   int p;
```

```
    // declared a with all values set to 1 as prompted
    int A[8][8] =
    {
        {1, 1, 1, 1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1, 1, 1, 1},
        {1, 1, 1, 1, 1, 1, 1, 1}
    };

// Calculating size of A
    //row a
    const int sizeRowA = sizeof(A) / sizeof(A[0]);
    //column a
    const int sizeColumnA = sizeof(A[0]) / sizeof(A[0][0]);

    //Printing matrix A
    printf("Matrix A:\n");
    for (i = 0; i < sizeRowA; i++)
    {
        for (k= 0; k< sizeColumnA; k++)
            {
                printf("%d ", A[i][k]);
            }
            printf("\n");
    }
    printf("\n");

    //PRINT_ARRAY(sizeRowA, sizeColumnA, A); // FUNCTION CALL FOR PRINT_ARRAY

    int B[8][8] = // DECLARE ARRAY B
    {
        {2, 2, 2, 2, 2, 2, 2, 2},
        {2, 2, 2, 2, 2, 2, 2, 2},
        {2, 2, 2, 2, 2, 2, 2, 2},
        {2, 2, 2, 2, 2, 2, 2, 2},
        {2, 2, 2, 2, 2, 2, 2, 2},
        {2, 2, 2, 2, 2, 2, 2, 2},
        {2, 2, 2, 2, 2, 2, 2, 2},
        {2, 2, 2, 2, 2, 2, 2, 2}
    };

// Calculating size of B
    //row b
    const int sizeRowB = sizeof(B) / sizeof(B[0]);
    const int sizeColumnB = sizeof(B[0]) / sizeof(B[0][0]);

    printf("Matrix B:\n");

    //Printing matrix B
    for (i = 0; i < sizeRowB; i++)
    {
        for (k= 0; k< sizeColumnB; k++)
```

```
          {
             printf("%d ", B[i][k]);
          }
          printf("\n");
    }
    printf("\n");

    // declaring the z result matrix
    int zresult[8][8] = {0};

    // matrix multiplication and stored in z
    for(i = 0; i < sizeRowA; i++)
    {
       for (k= 0; k< sizeColumnB; k++)
       {
          zresult[i][k] = 0;
          for(p = 0; p < sizeColumnA; p++)
          {
             zresult[i][k] += A[i][p] * B[p][k];
          }
       }
    }


// Size calculation for z
    //rows of z
    const int sizeRowZ = sizeof(zresult) / sizeof(zresult[0]);
    //columns of z
    const int sizeColumnZ = sizeof(zresult[0]) / sizeof(zresult[0][0]);
    printf("Matrix C:\n");

    //printing C array
    for (i = 0; i < sizeRowZ; i++)
    {
       for (k= 0; k < sizeColumnZ; k++)
          {
             printf("%d ", zresult[i][k]);
          }
          printf("\n");
    }
    printf("\n");

    return 0;
}
```