Anders Dahl Homework 0 1/27/15

1.

a.
53_{10}
53-32 => 00100000, 18 left
21-16 => 00110000, 5 left
5-4 => 00110100, 1 left
1-1 => 00110101, 0 left
= 00110101

b.
FA_{16}
F = 1111, A = 1010
= 11111010

2.

a.
$19_{10} \rightarrow (?)_2$
19-16 => 00010000, 3 left
3-2 => 00010010, 1 left
1-1 => 00010011, 0 left
= 00010011

b.
$-13_{10} \rightarrow (?)_2$
13-8 => 00001000, 5 left
5-4 => 00001100, 1 left
1-1 => 00001101, 0 left
Flip it to negative: 11110010
Add 1: 11110011
= 11110011

c.
$-23_{10} \rightarrow (?)_{16}$
23-16 => 00010000, 7 left
7-4 => 00010100, 3 left
3-2 => 00010110, 1 left
1-1 => 00010111
Flip it to negative: 11101000
Add 1: 11101001
Make it hex: E9
= 0xFFFFFFFFFFFFFFE9

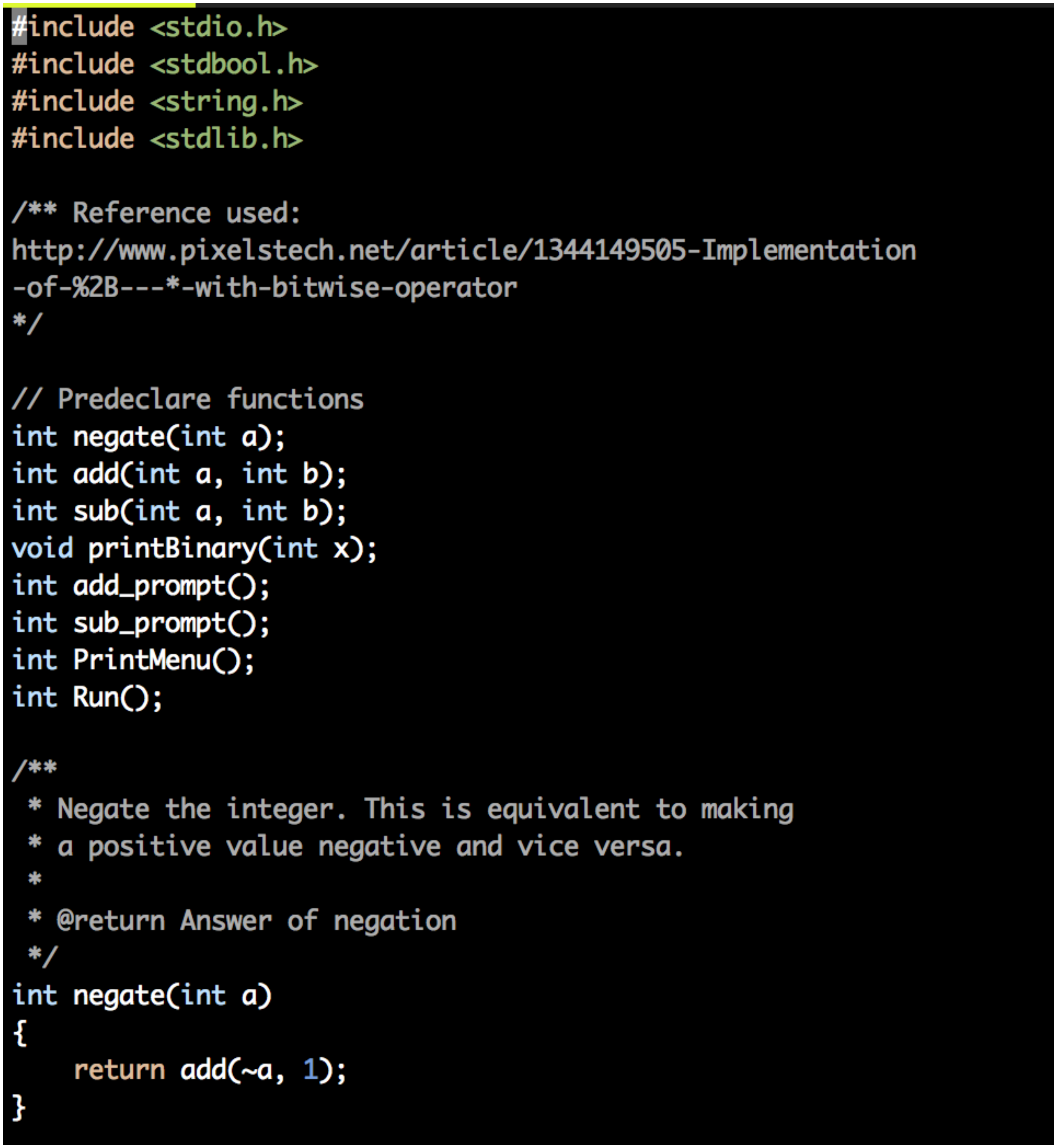
d.
$ED_{16} \rightarrow (?)_{10}$
Make it binary: 11101101

11101101 => 1 + 4 + 8 + 32 + 64 +128 = 237
= 237

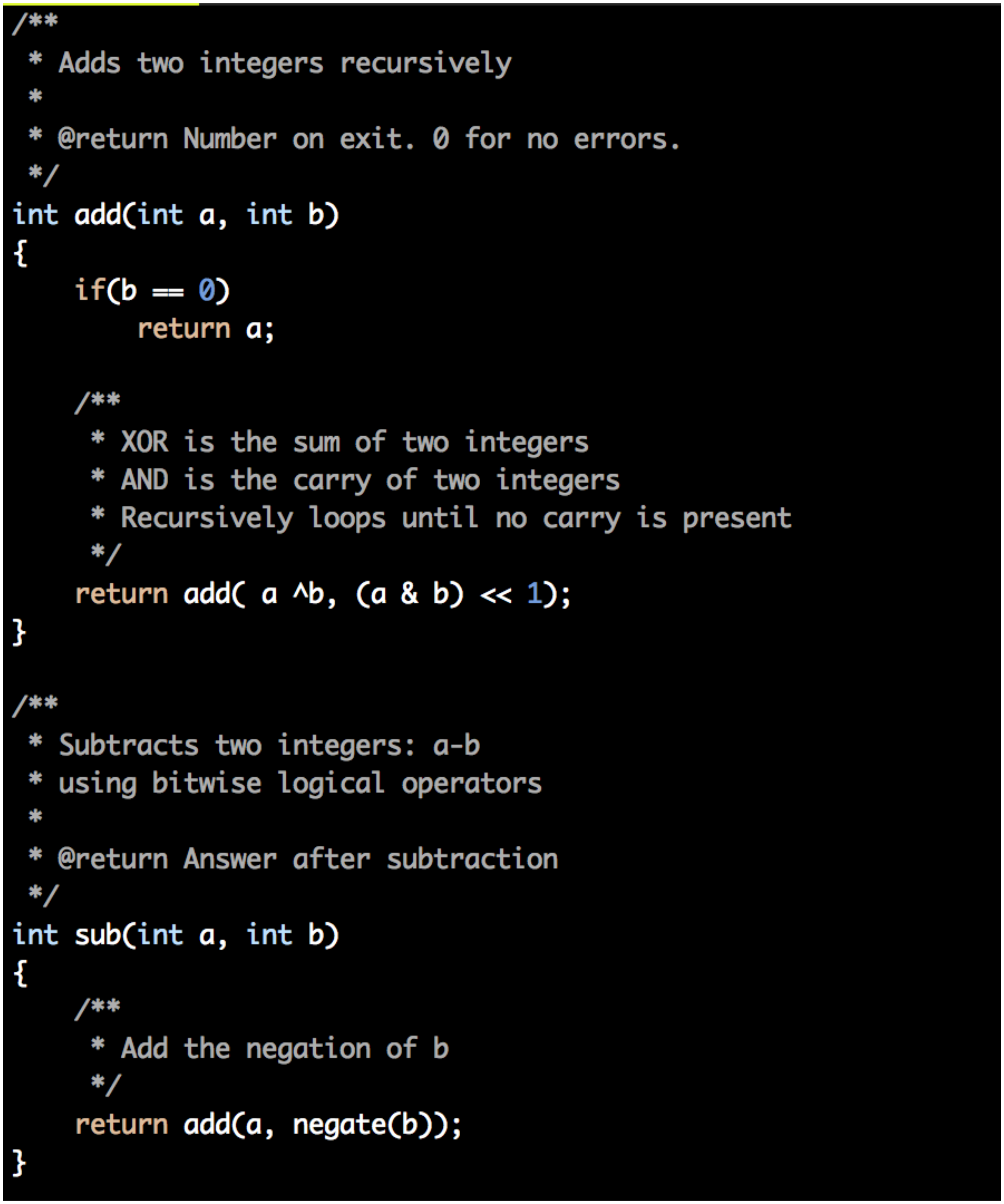
3.
a.
0xABCD or 0x9876
0xABCD => 1010 1011 1100 1101
0x9876 => 1001 1000 0111 0110
1010 1011 1100 1101 or
1001 1000 0111 0110
= 1011 1011 1111 1111
= 0xBBFF

b.
0xFEED and (not(0xBEEF))
0xFEED => 1111 1110 1110 1101
0xBEEF => 1011 1110 1110 1111
not (0xBEEF) => 0100 0001 0001 0000
1111 1110 1110 1101 and
0100 0001 0001 0000
= 0100 0000 0000 0000
Convert to hex: 0x8000
= 0x8000

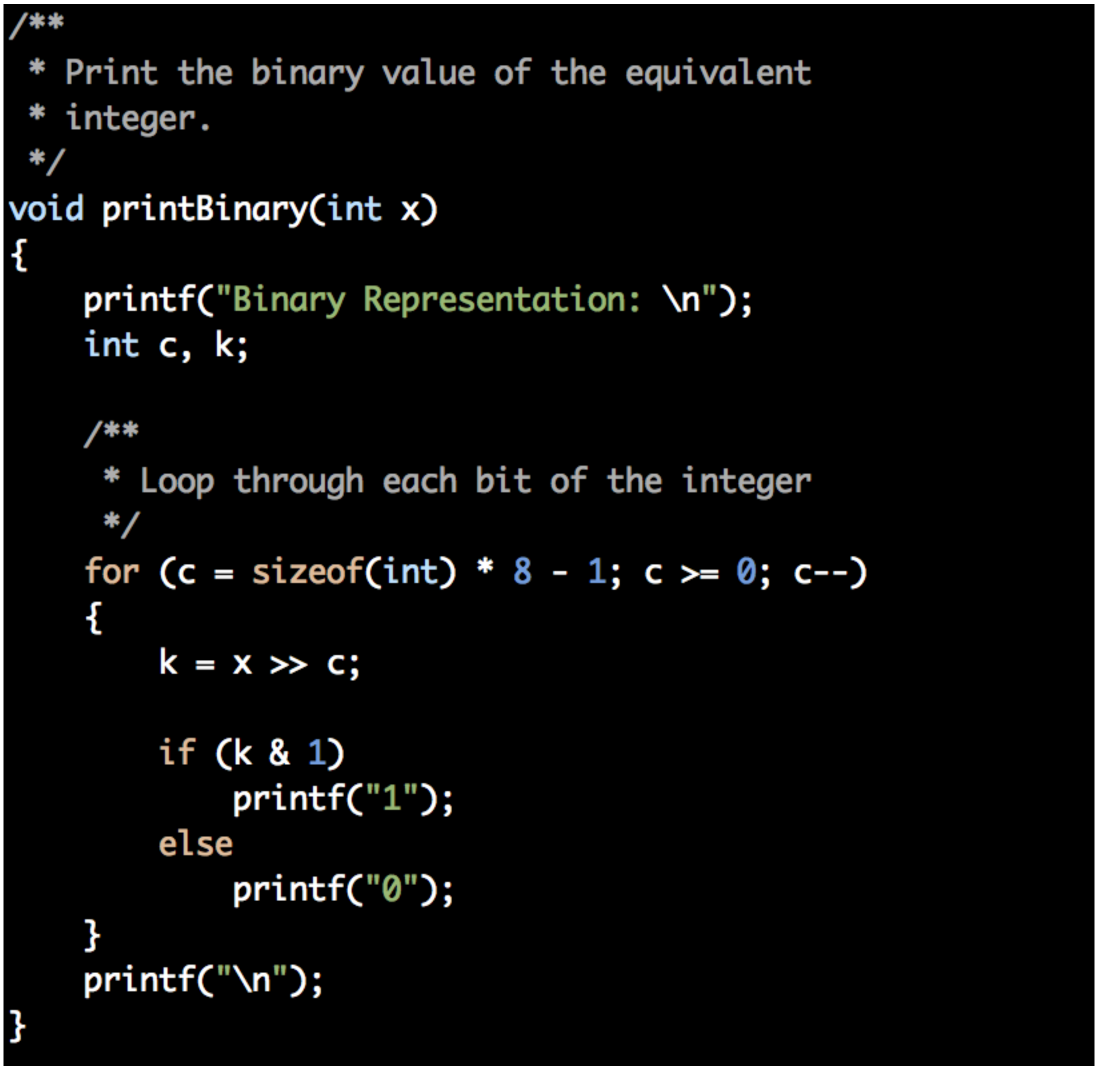
4. Code:



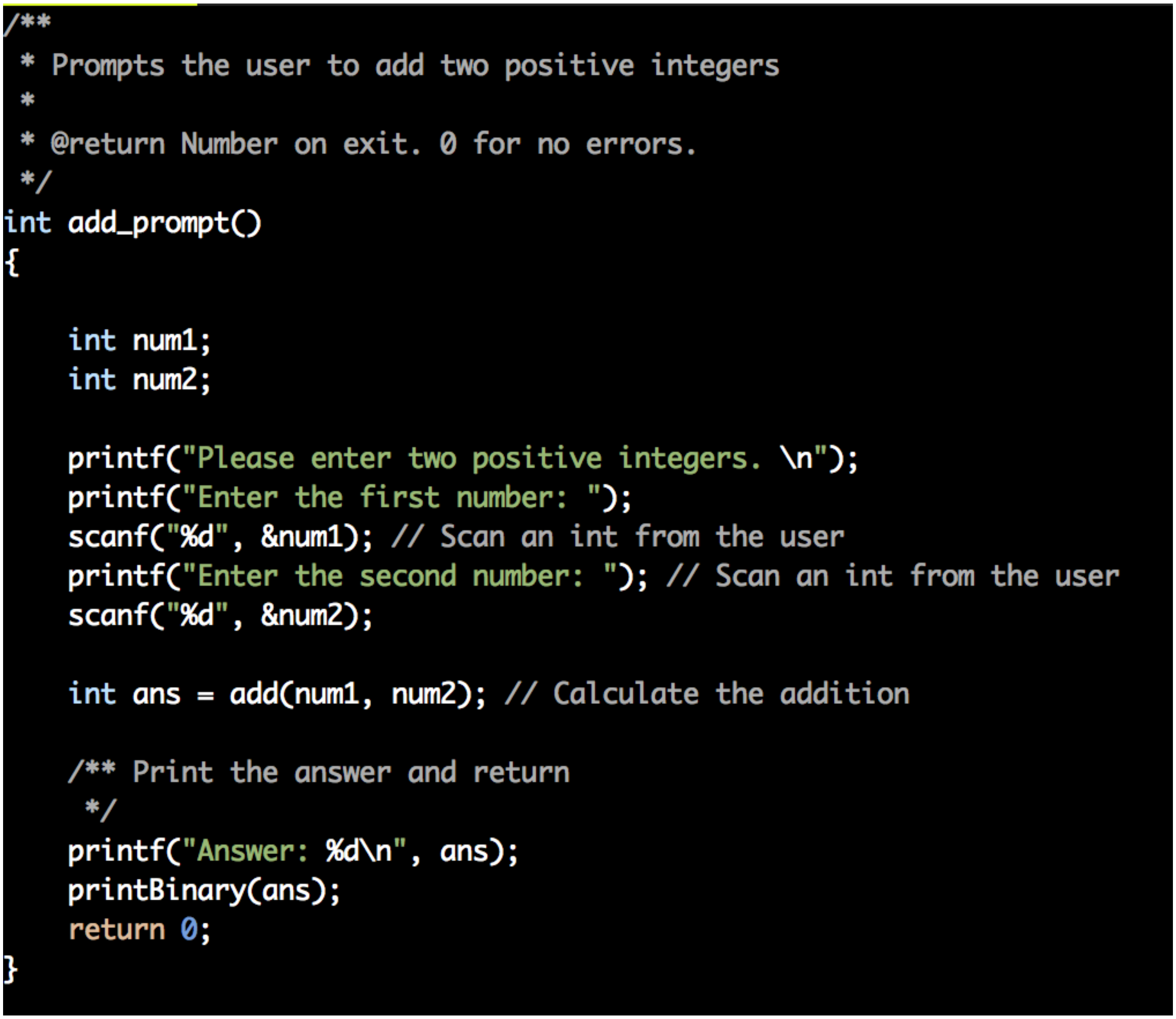
```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>

/** Reference used:
http://www.pixelstech.net/article/1344149505-Implementation
-of-%2B---*-with-bitwise-operator
*/

// Predeclare functions
int negate(int a);
int add(int a, int b);
int sub(int a, int b);
void printBinary(int x);
int add_prompt();
int sub_prompt();
int PrintMenu();
int Run();

/**
 * Negate the integer. This is equivalent to making
 * a positive value negative and vice versa.
 *
 * @return Answer of negation
 */
int negate(int a)
{
    return add(~a, 1);
}
```



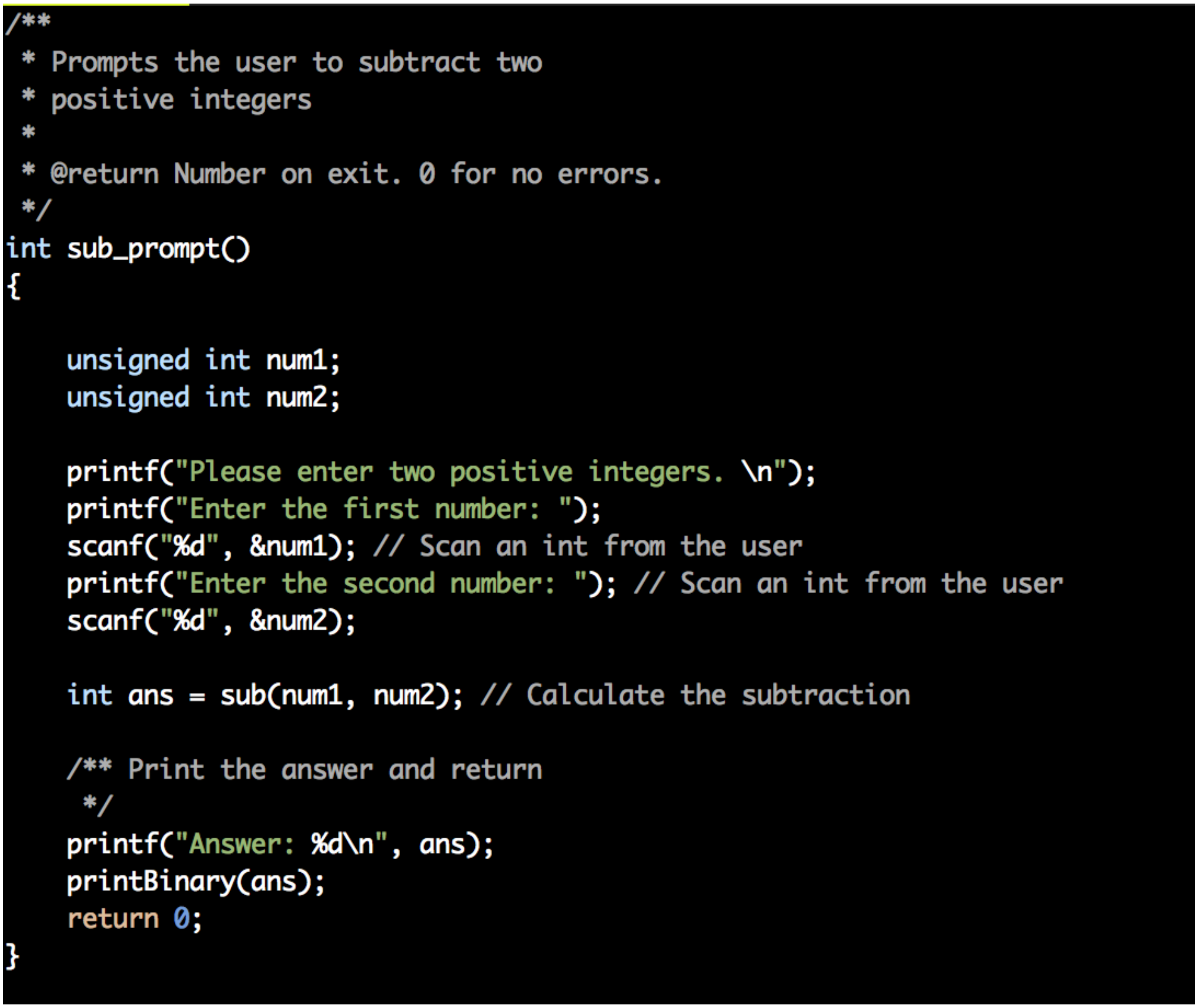
```
/**
 * Adds two integers recursively
 *
 * @return Number on exit. 0 for no errors.
 */
int add(int a, int b)
{
    if(b == 0)
        return a;

    /**
     * XOR is the sum of two integers
     * AND is the carry of two integers
     * Recursively loops until no carry is present
     */
    return add( a ^b, (a & b) << 1);
}

/**
 * Subtracts two integers: a-b
 * using bitwise logical operators
 *
 * @return Answer after subtraction
 */
int sub(int a, int b)
{
    /**
     * Add the negation of b
     */
    return add(a, negate(b));
}
```



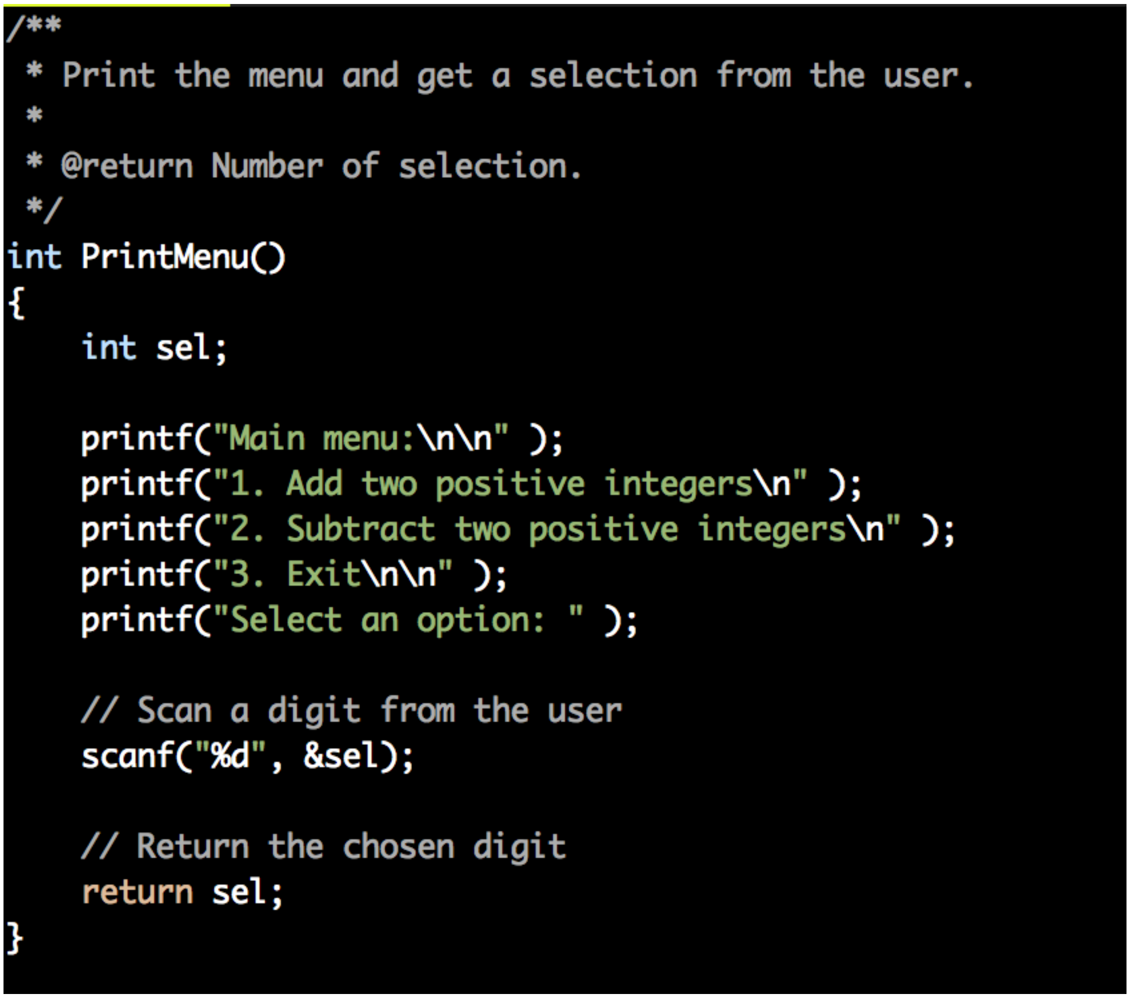
```
/**
 * Print the binary value of the equivalent
 * integer.
 */
void printBinary(int x)
{
    printf("Binary Representation: \n");
    int c, k;

    /**
     * Loop through each bit of the integer
     */
    for (c = sizeof(int) * 8 - 1; c >= 0; c--)
    {
        k = x >> c;

        if (k & 1)
            printf("1");
        else
            printf("0");
    }
    printf("\n");
}
```



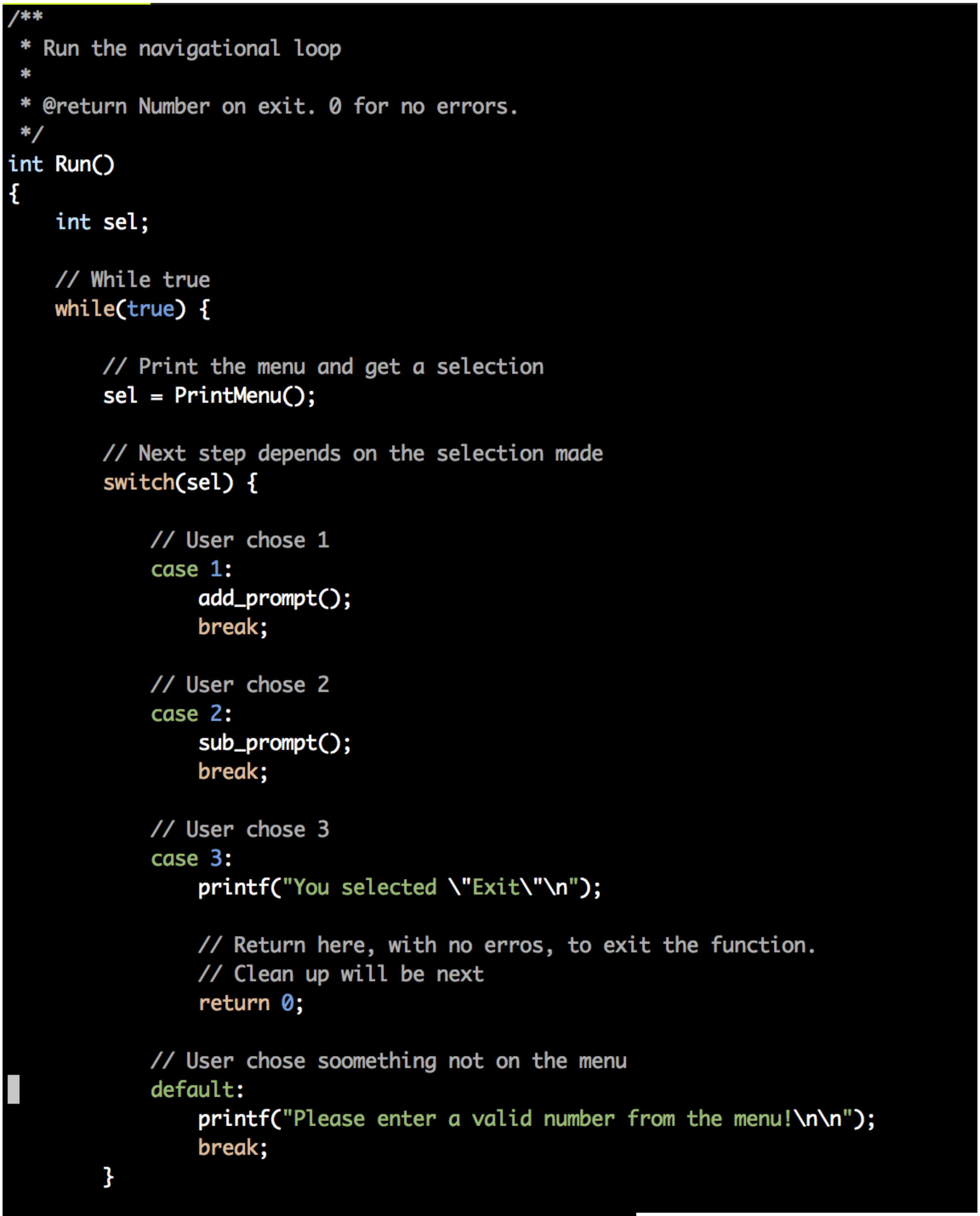
```c
/**
 * Prompts the user to add two positive integers
 *
 * @return Number on exit. 0 for no errors.
 */
int add_prompt()
{

    int num1;
    int num2;

    printf("Please enter two positive integers. \n");
    printf("Enter the first number: ");
    scanf("%d", &num1); // Scan an int from the user
    printf("Enter the second number: "); // Scan an int from the user
    scanf("%d", &num2);

    int ans = add(num1, num2); // Calculate the addition

    /** Print the answer and return
     */
    printf("Answer: %d\n", ans);
    printBinary(ans);
    return 0;
}
```



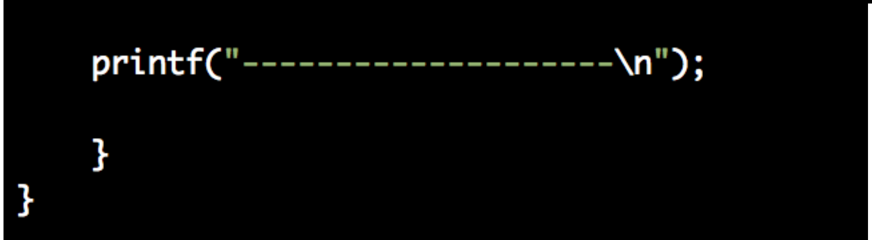
```
/**
 * Prompts the user to subtract two
 * positive integers
 *
 * @return Number on exit. 0 for no errors.
 */
int sub_prompt()
{

    unsigned int num1;
    unsigned int num2;

    printf("Please enter two positive integers. \n");
    printf("Enter the first number: ");
    scanf("%d", &num1); // Scan an int from the user
    printf("Enter the second number: "); // Scan an int from the user
    scanf("%d", &num2);

    int ans = sub(num1, num2); // Calculate the subtraction

    /** Print the answer and return
     */
    printf("Answer: %d\n", ans);
    printBinary(ans);
    return 0;
}
```



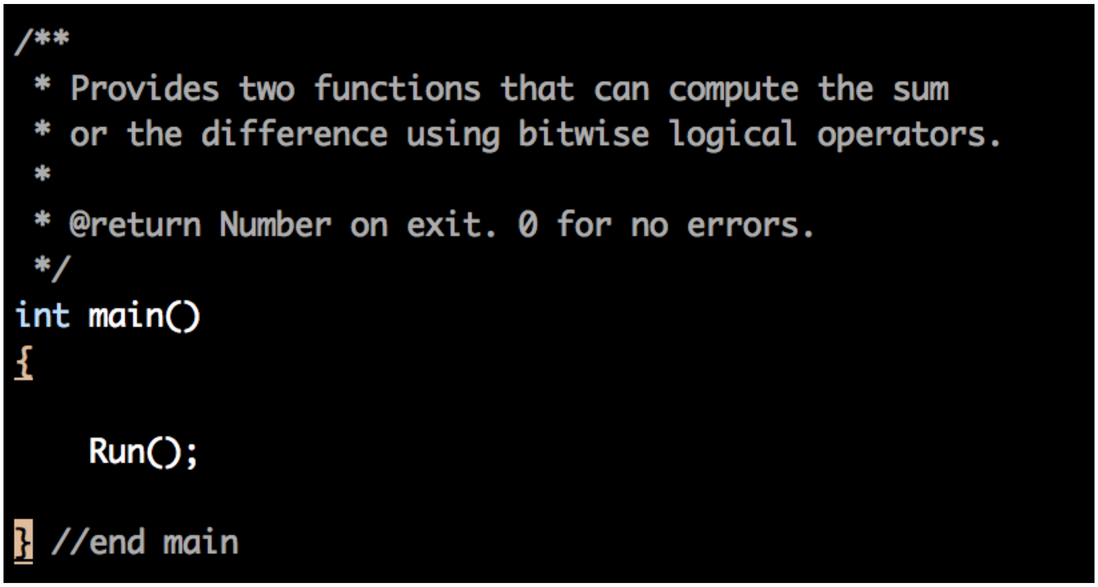
```c
/**
 * Print the menu and get a selection from the user.
 *
 * @return Number of selection.
 */
int PrintMenu()
{
    int sel;

    printf("Main menu:\n\n" );
    printf("1. Add two positive integers\n" );
    printf("2. Subtract two positive integers\n" );
    printf("3. Exit\n\n" );
    printf("Select an option: " );

    // Scan a digit from the user
    scanf("%d", &sel);

    // Return the chosen digit
    return sel;
}
```



```
/**
 * Run the navigational loop
 *
 * @return Number on exit. 0 for no errors.
 */
int Run()
{
    int sel;

    // While true
    while(true) {

        // Print the menu and get a selection
        sel = PrintMenu();

        // Next step depends on the selection made
        switch(sel) {

            // User chose 1
            case 1:
                add_prompt();
                break;

            // User chose 2
            case 2:
                sub_prompt();
                break;

            // User chose 3
            case 3:
                printf("You selected \"Exit\"\n");

                // Return here, with no erros, to exit the function.
                // Clean up will be next
                return 0;

            // User chose soomething not on the menu
            default:
                printf("Please enter a valid number from the menu!\n\n");
                break;
        }
```

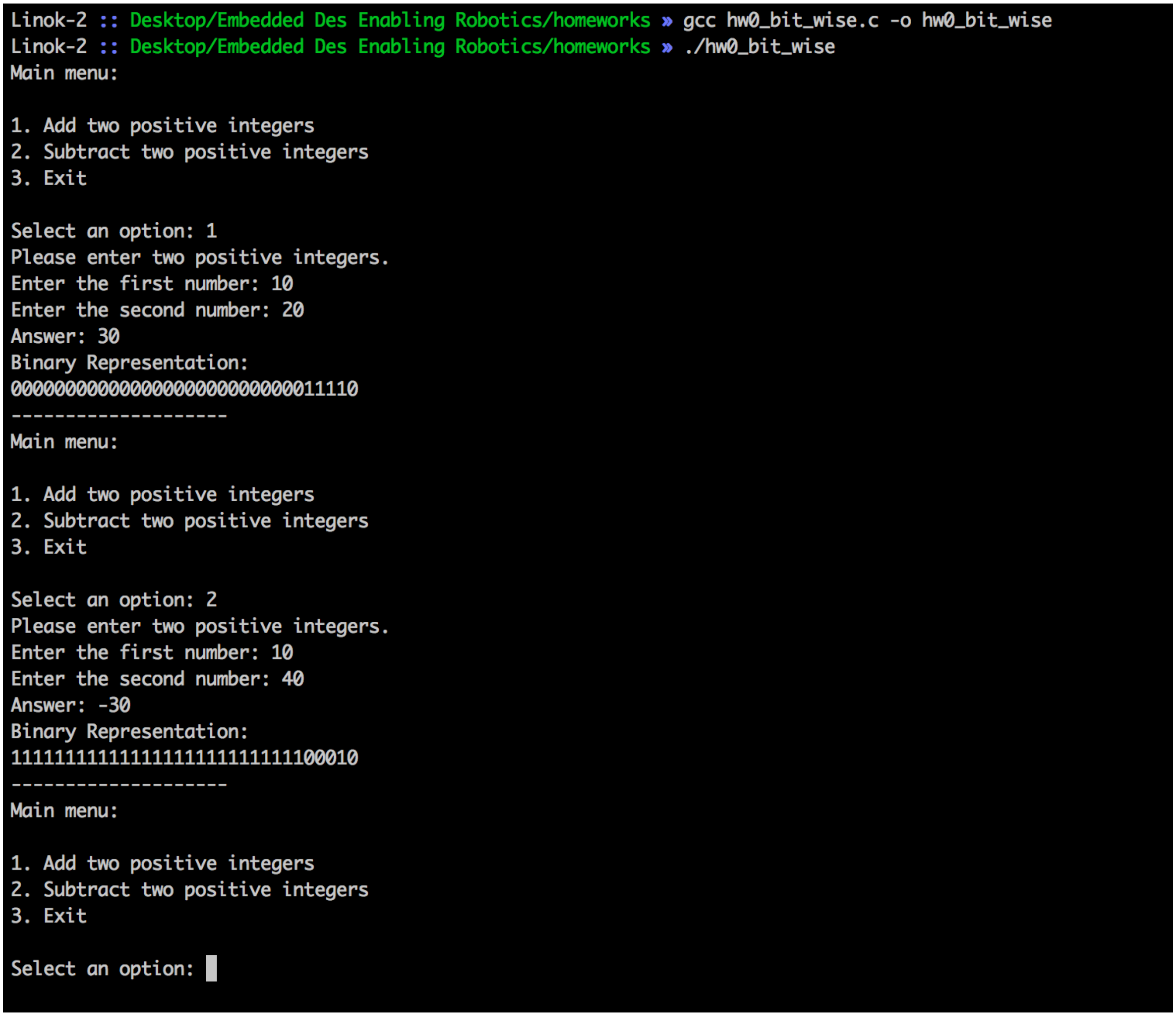


```
    printf("--------------------\n");

    }
}
```



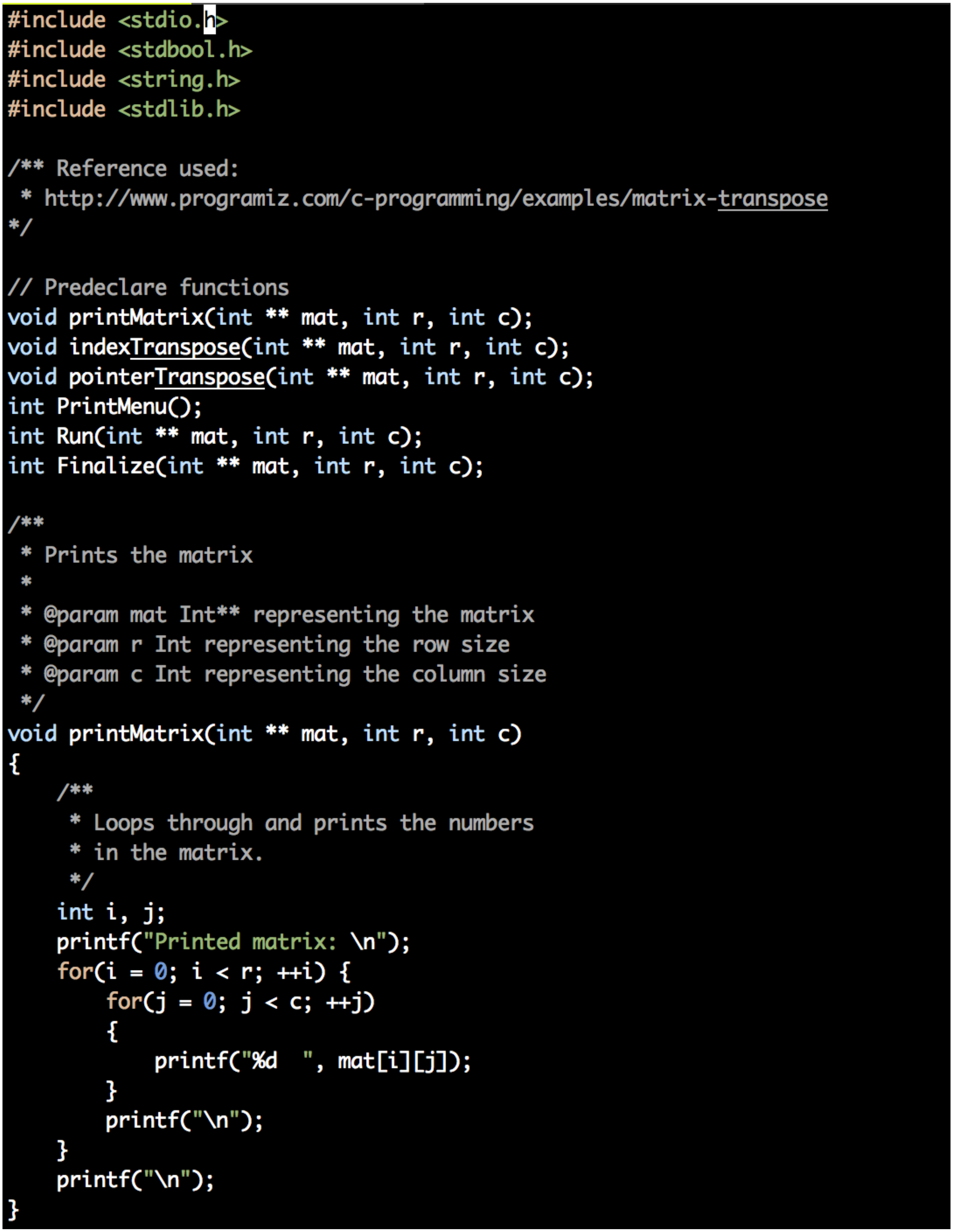
```
/**
 * Provides two functions that can compute the sum
 * or the difference using bitwise logical operators.
 *
 * @return Number on exit. 0 for no errors.
 */
int main()
{

    Run();

} //end main
```

4. Running the Code:

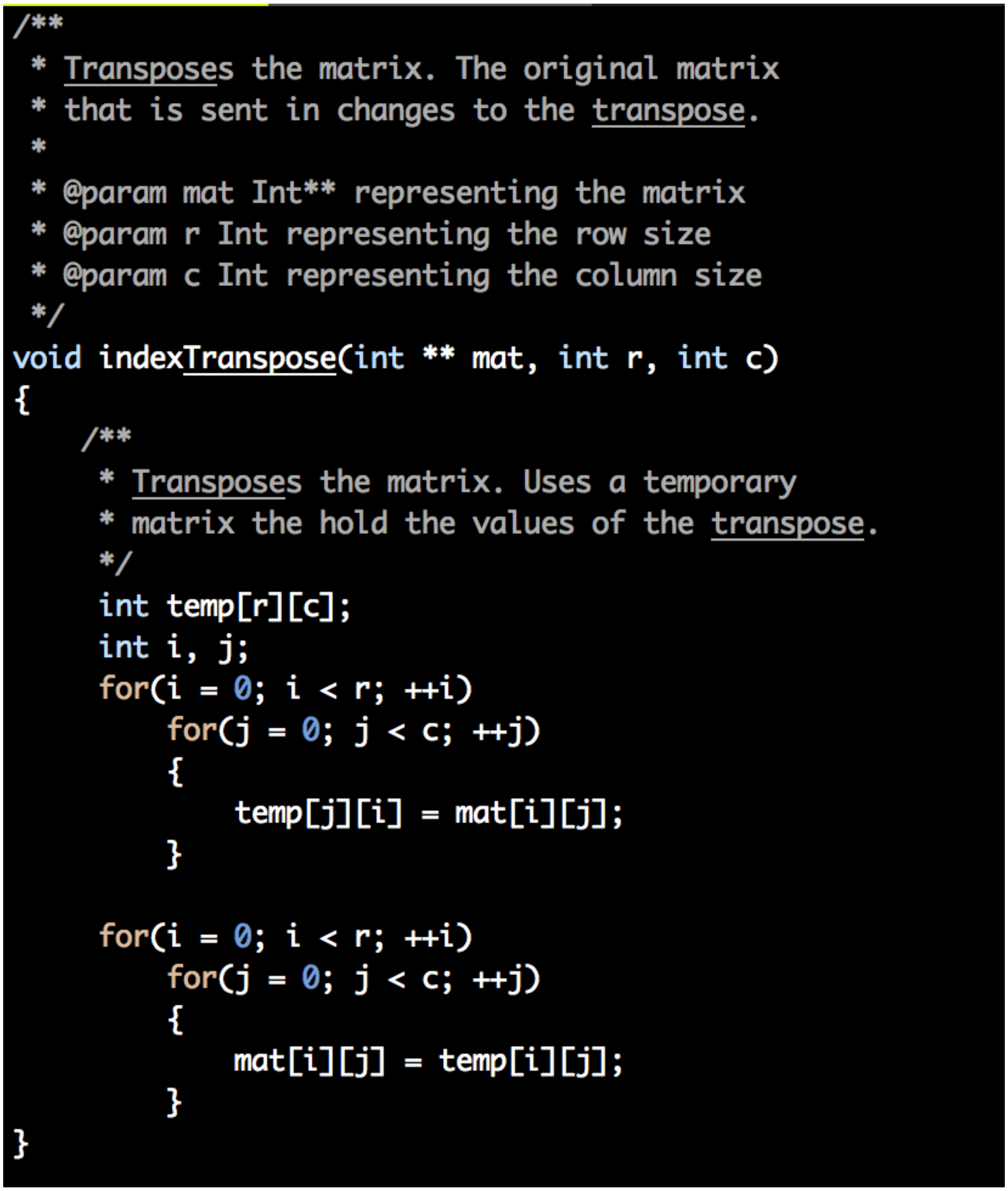


```
Linok-2 :: Desktop/Embedded Des Enabling Robotics/homeworks » gcc hw0_bit_wise.c -o hw0_bit_wise
Linok-2 :: Desktop/Embedded Des Enabling Robotics/homeworks » ./hw0_bit_wise
Main menu:

1. Add two positive integers
2. Subtract two positive integers
3. Exit

Select an option: 1
Please enter two positive integers.
Enter the first number: 10
Enter the second number: 20
Answer: 30
Binary Representation:
00000000000000000000000000011110
--------------------
Main menu:

1. Add two positive integers
2. Subtract two positive integers
3. Exit

Select an option: 2
Please enter two positive integers.
Enter the first number: 10
Enter the second number: 40
Answer: -30
Binary Representation:
11111111111111111111111111100010
--------------------
Main menu:

1. Add two positive integers
2. Subtract two positive integers
3. Exit

Select an option: 
```

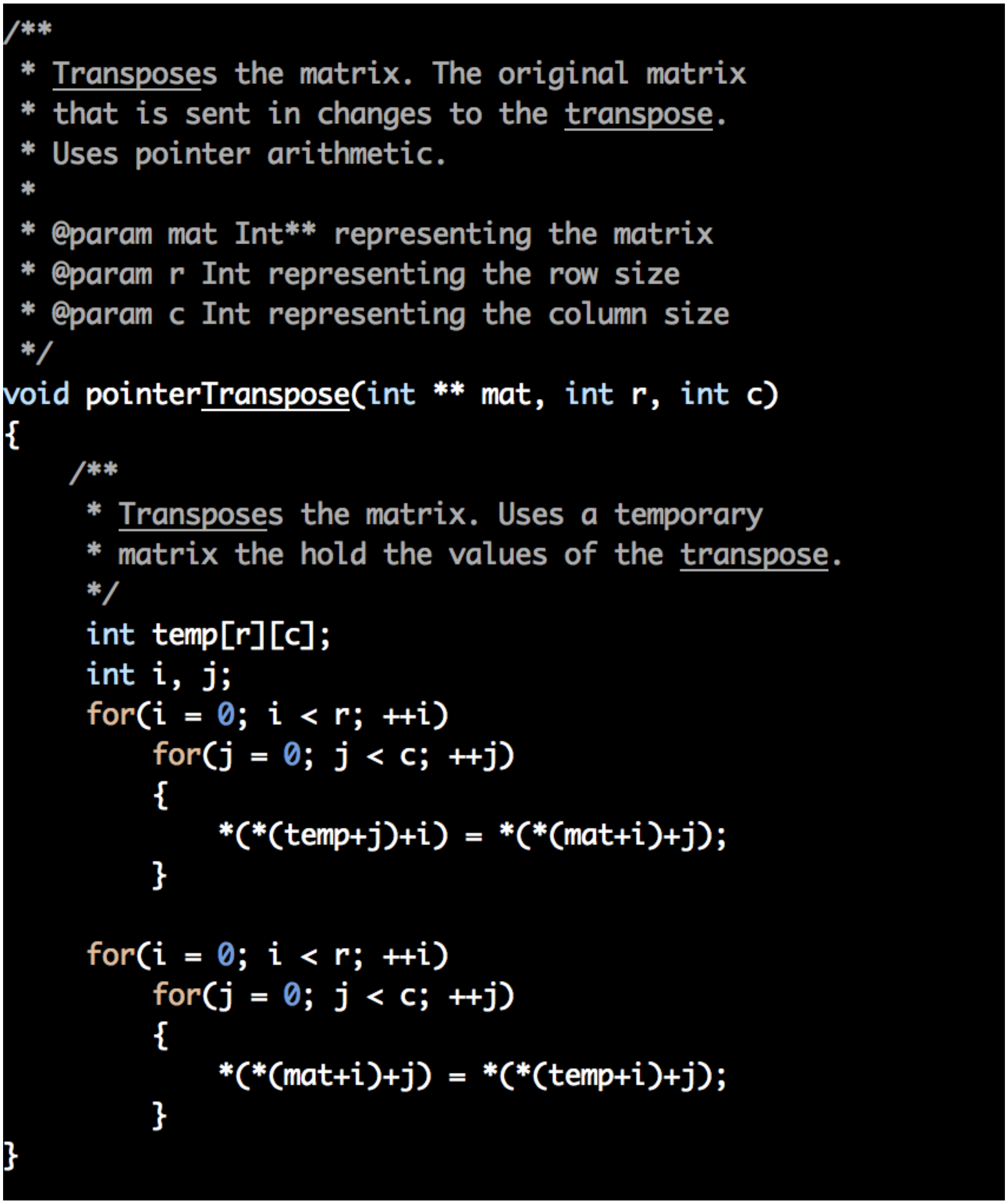
5. Code:



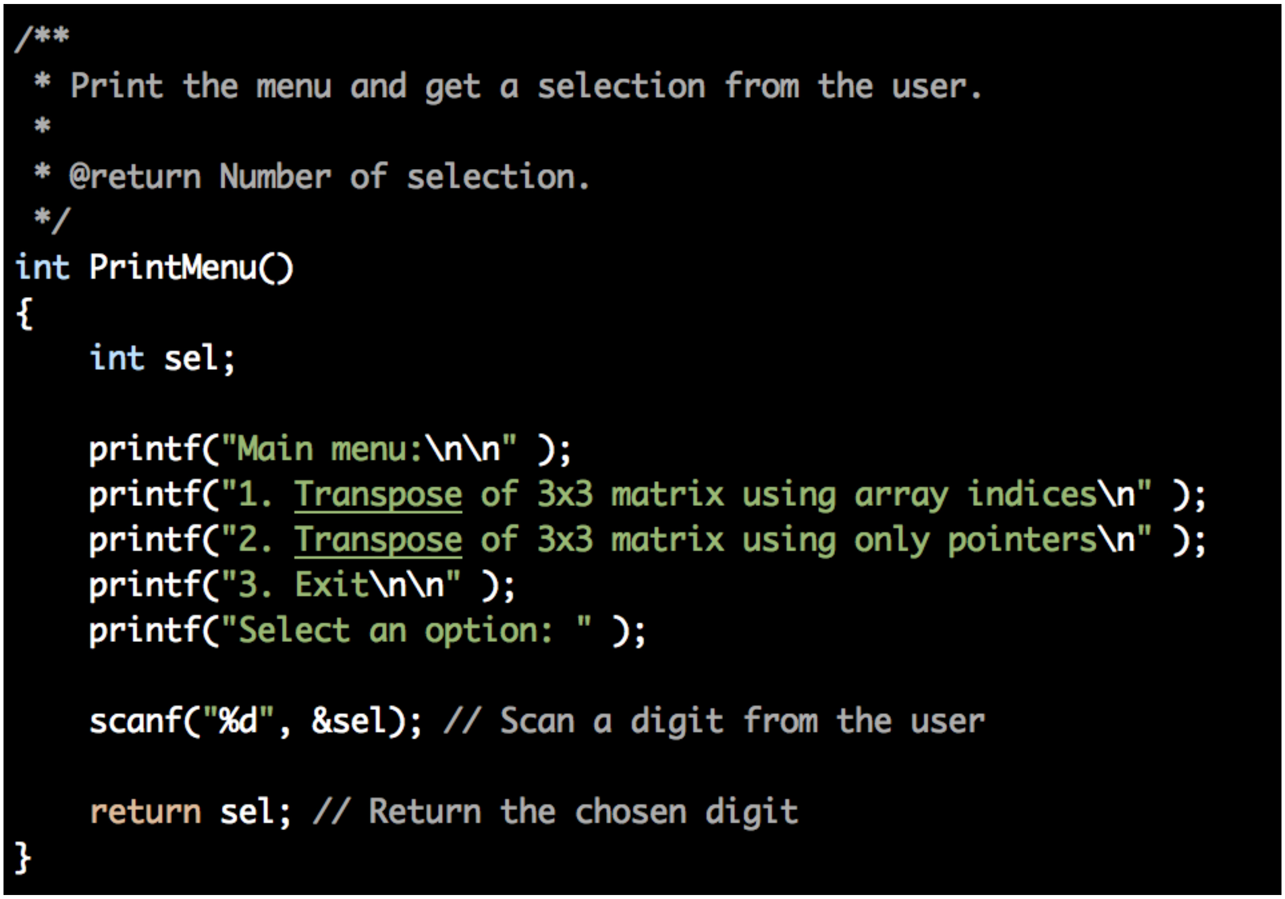
```c
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>

/** Reference used:
 * http://www.programiz.com/c-programming/examples/matrix-transpose
*/

// Predeclare functions
void printMatrix(int ** mat, int r, int c);
void indexTranspose(int ** mat, int r, int c);
void pointerTranspose(int ** mat, int r, int c);
int PrintMenu();
int Run(int ** mat, int r, int c);
int Finalize(int ** mat, int r, int c);

/**
 * Prints the matrix
 *
 * @param mat Int** representing the matrix
 * @param r Int representing the row size
 * @param c Int representing the column size
 */
void printMatrix(int ** mat, int r, int c)
{
    /**
     * Loops through and prints the numbers
     * in the matrix.
     */
    int i, j;
    printf("Printed matrix: \n");
    for(i = 0; i < r; ++i) {
        for(j = 0; j < c; ++j)
        {
            printf("%d  ", mat[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}
```



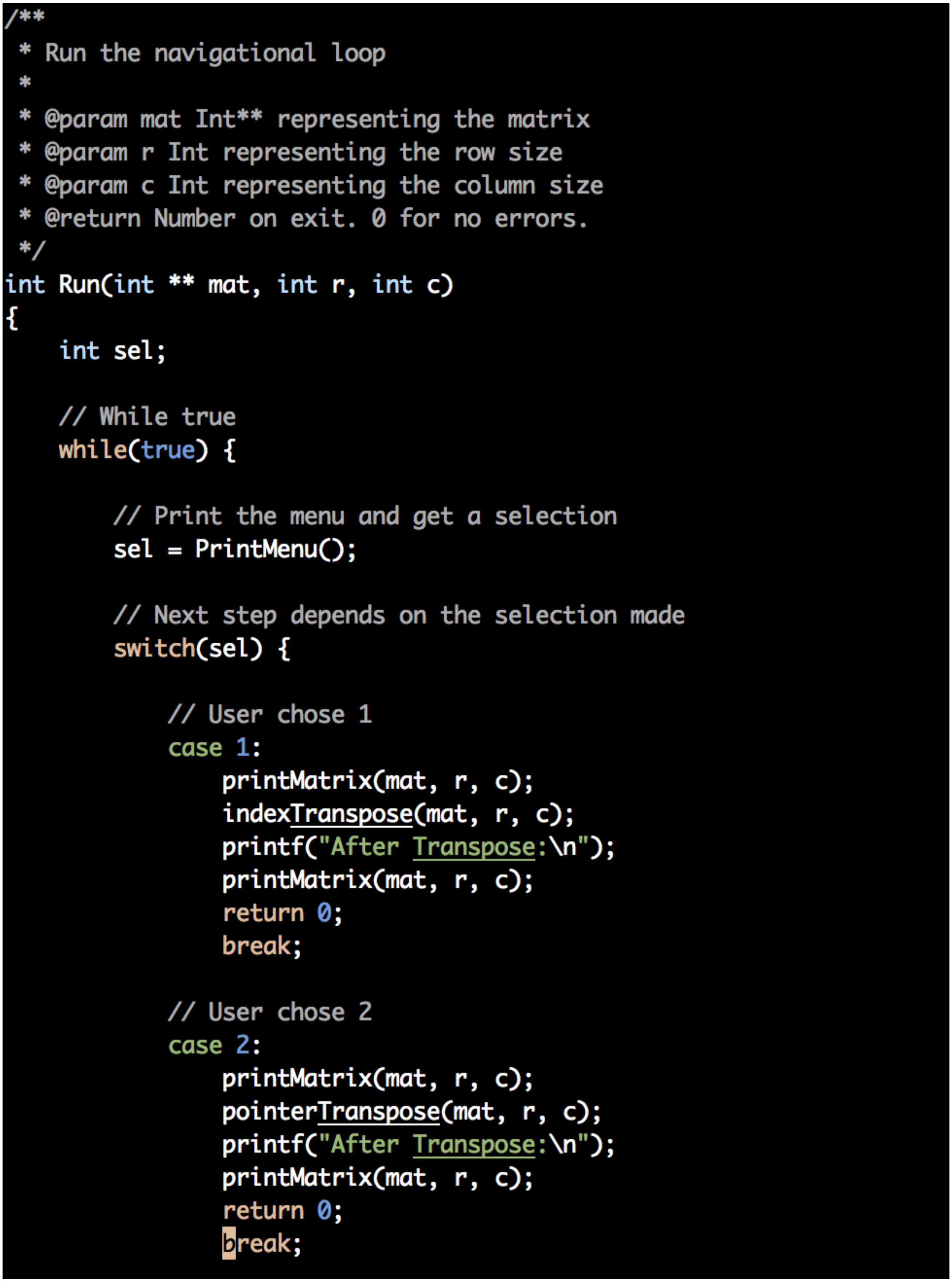
```
/**
 * Transposes the matrix. The original matrix
 * that is sent in changes to the transpose.
 *
 * @param mat Int** representing the matrix
 * @param r Int representing the row size
 * @param c Int representing the column size
 */
void indexTranspose(int ** mat, int r, int c)
{
    /**
     * Transposes the matrix. Uses a temporary
     * matrix the hold the values of the transpose.
     */
     int temp[r][c];
     int i, j;
     for(i = 0; i < r; ++i)
         for(j = 0; j < c; ++j)
         {
             temp[j][i] = mat[i][j];
         }

     for(i = 0; i < r; ++i)
         for(j = 0; j < c; ++j)
         {
             mat[i][j] = temp[i][j];
         }
}
```



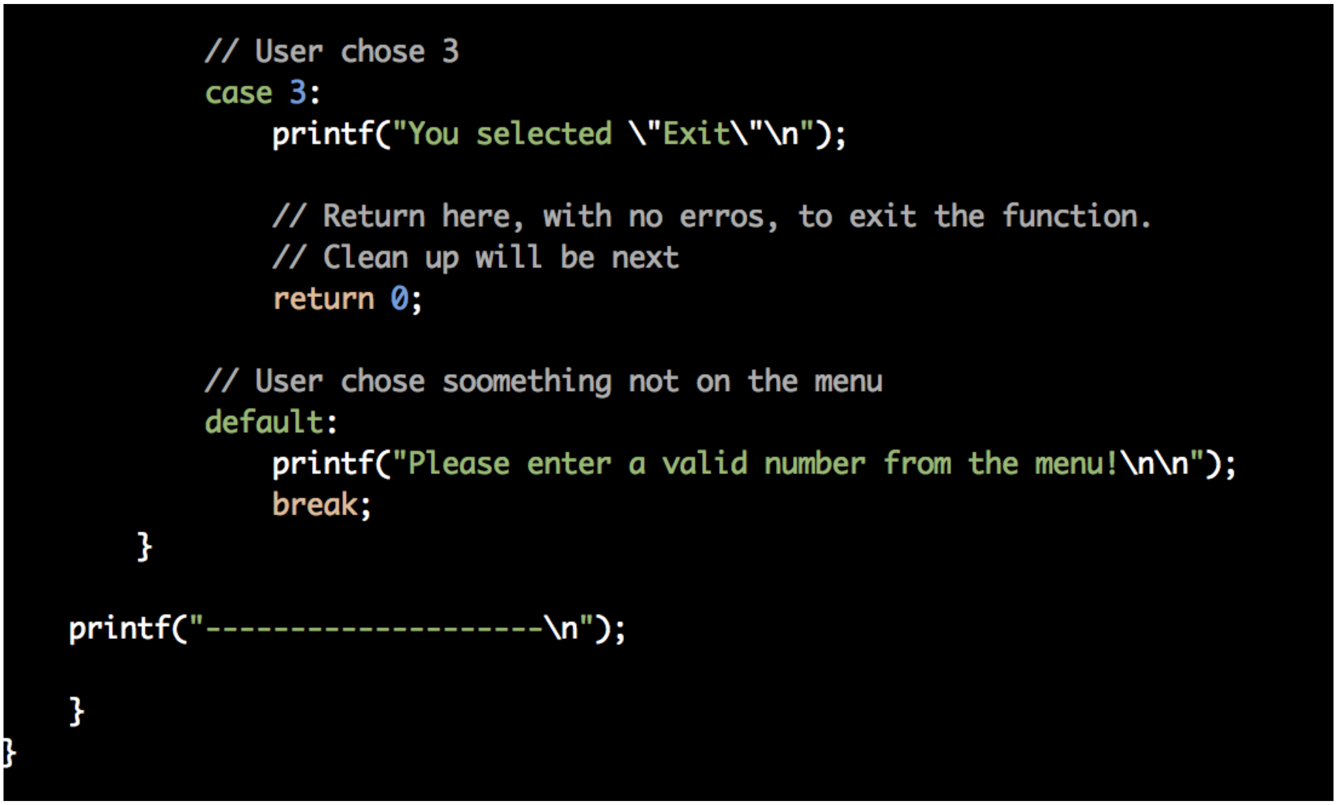
```
/**
 * Transposes the matrix. The original matrix
 * that is sent in changes to the transpose.
 * Uses pointer arithmetic.
 *
 * @param mat Int** representing the matrix
 * @param r Int representing the row size
 * @param c Int representing the column size
 */
void pointerTranspose(int ** mat, int r, int c)
{
    /**
     * Transposes the matrix. Uses a temporary
     * matrix the hold the values of the transpose.
     */
    int temp[r][c];
    int i, j;
    for(i = 0; i < r; ++i)
        for(j = 0; j < c; ++j)
        {
            *(*(temp+j)+i) = *(*(mat+i)+j);
        }

    for(i = 0; i < r; ++i)
        for(j = 0; j < c; ++j)
        {
            *(*(mat+i)+j) = *(*(temp+i)+j);
        }
}
```



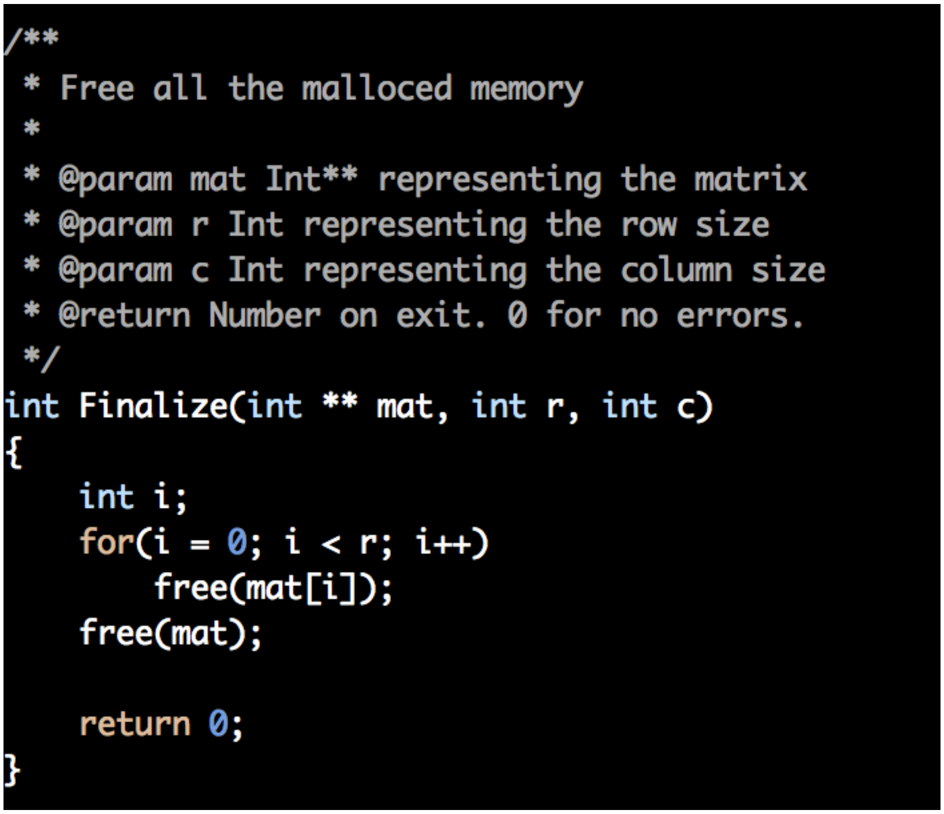
```c
/**
 * Print the menu and get a selection from the user.
 *
 * @return Number of selection.
 */
int PrintMenu()
{
    int sel;

    printf("Main menu:\n\n" );
    printf("1. Transpose of 3x3 matrix using array indices\n" );
    printf("2. Transpose of 3x3 matrix using only pointers\n" );
    printf("3. Exit\n\n" );
    printf("Select an option: " );

    scanf("%d", &sel); // Scan a digit from the user

    return sel; // Return the chosen digit
}
```



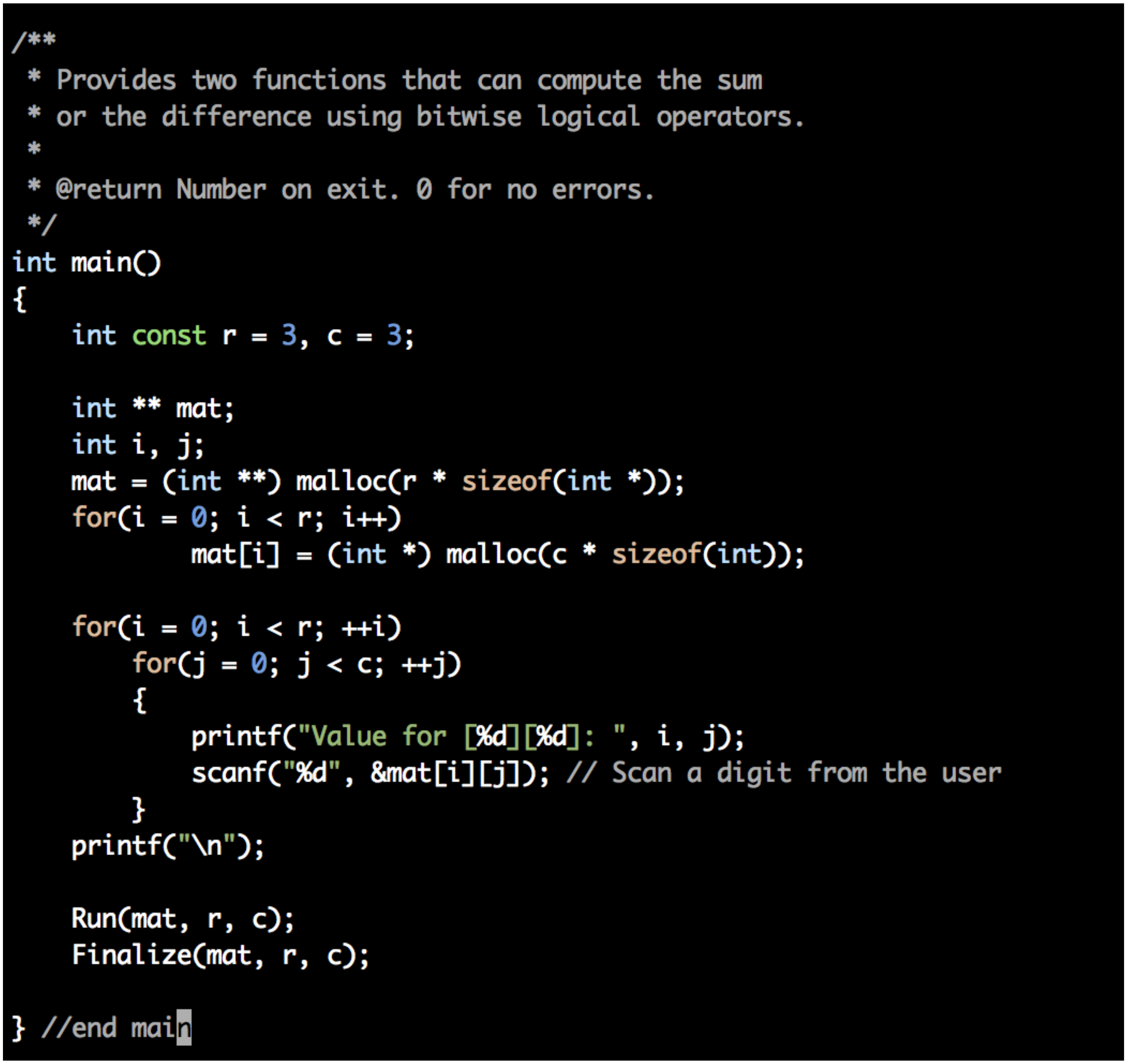
```c
/**
 * Run the navigational loop
 *
 * @param mat Int** representing the matrix
 * @param r Int representing the row size
 * @param c Int representing the column size
 * @return Number on exit. 0 for no errors.
 */
int Run(int ** mat, int r, int c)
{
    int sel;

    // While true
    while(true) {

        // Print the menu and get a selection
        sel = PrintMenu();

        // Next step depends on the selection made
        switch(sel) {

            // User chose 1
            case 1:
                printMatrix(mat, r, c);
                indexTranspose(mat, r, c);
                printf("After Transpose:\n");
                printMatrix(mat, r, c);
                return 0;
                break;

            // User chose 2
            case 2:
                printMatrix(mat, r, c);
                pointerTranspose(mat, r, c);
                printf("After Transpose:\n");
                printMatrix(mat, r, c);
                return 0;
                break;
```



```
            // User chose 3
            case 3:
                printf("You selected \"Exit\"\n");

                // Return here, with no erros, to exit the function.
                // Clean up will be next
                return 0;

            // User chose soomething not on the menu
            default:
                printf("Please enter a valid number from the menu!\n\n");
                break;
        }

    printf("--------------------\n");

    }
}
```

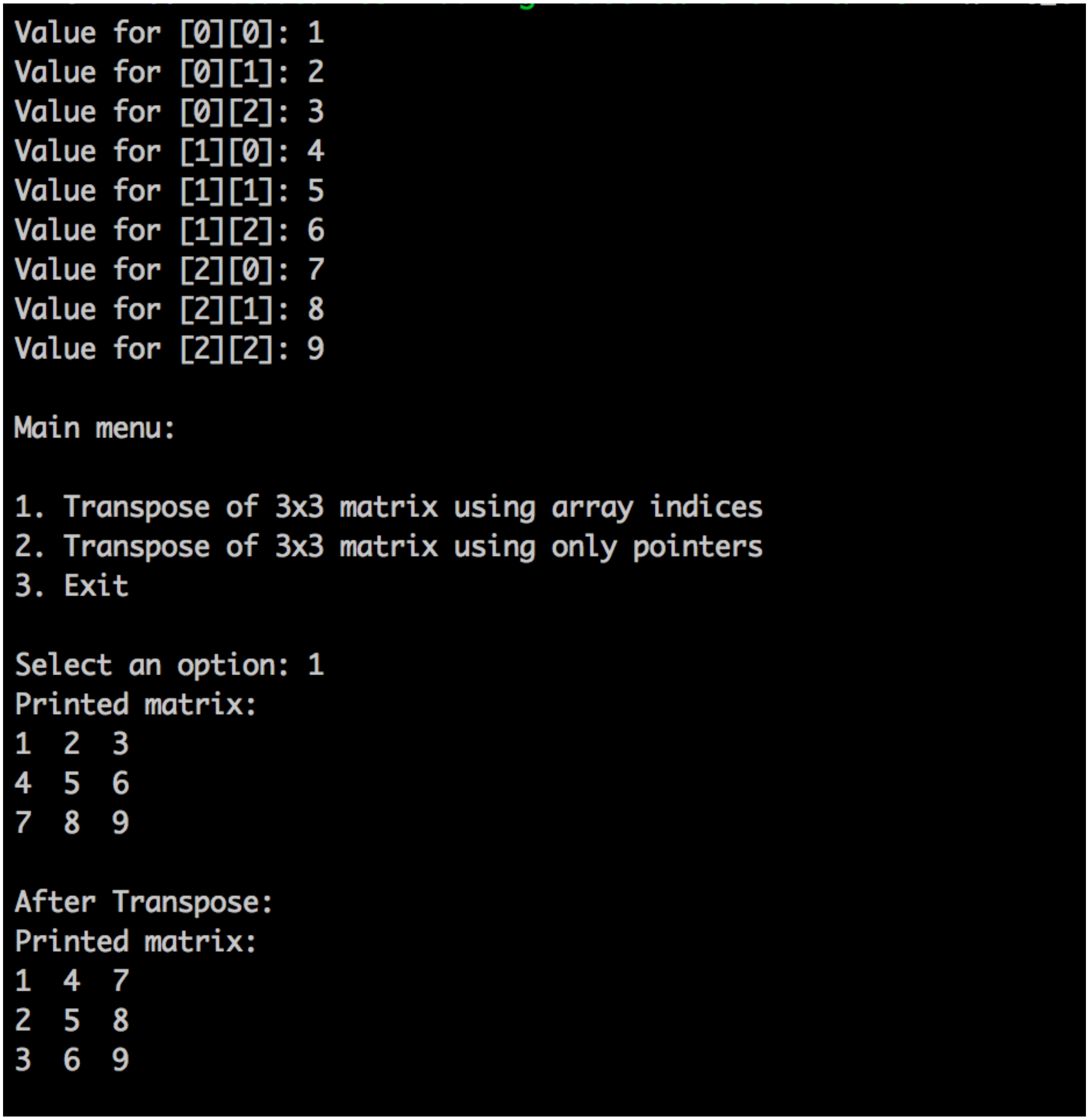


```
/**
 * Free all the malloced memory
 *
 * @param mat Int** representing the matrix
 * @param r Int representing the row size
 * @param c Int representing the column size
 * @return Number on exit. 0 for no errors.
 */
int Finalize(int ** mat, int r, int c)
{
    int i;
    for(i = 0; i < r; i++)
        free(mat[i]);
    free(mat);

    return 0;
}
```

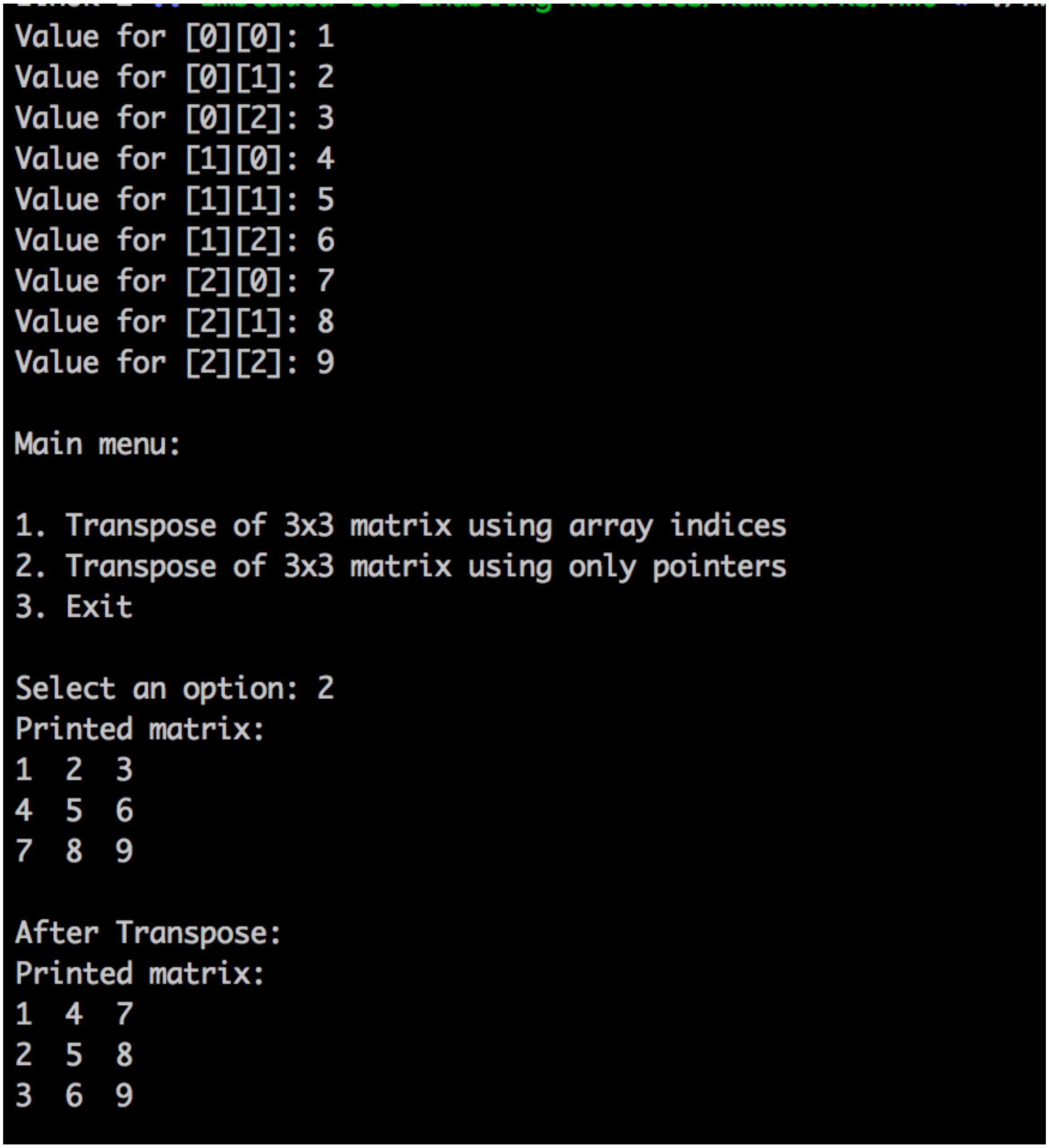


```c
/**
 * Provides two functions that can compute the sum
 * or the difference using bitwise logical operators.
 *
 * @return Number on exit. 0 for no errors.
 */
int main()
{
    int const r = 3, c = 3;

    int ** mat;
    int i, j;
    mat = (int **) malloc(r * sizeof(int *));
    for(i = 0; i < r; i++)
            mat[i] = (int *) malloc(c * sizeof(int));

    for(i = 0; i < r; ++i)
        for(j = 0; j < c; ++j)
        {
            printf("Value for [%d][%d]: ", i, j);
            scanf("%d", &mat[i][j]); // Scan a digit from the user
        }
    printf("\n");

    Run(mat, r, c);
    Finalize(mat, r, c);

} //end main
```

5. Run Program:



```
Value for [0][0]: 1
Value for [0][1]: 2
Value for [0][2]: 3
Value for [1][0]: 4
Value for [1][1]: 5
Value for [1][2]: 6
Value for [2][0]: 7
Value for [2][1]: 8
Value for [2][2]: 9

Main menu:

1. Transpose of 3x3 matrix using array indices
2. Transpose of 3x3 matrix using only pointers
3. Exit

Select an option: 1
Printed matrix:
1  2  3
4  5  6
7  8  9

After Transpose:
Printed matrix:
1  4  7
2  5  8
3  6  9
```



```
Value for [0][0]: 1
Value for [0][1]: 2
Value for [0][2]: 3
Value for [1][0]: 4
Value for [1][1]: 5
Value for [1][2]: 6
Value for [2][0]: 7
Value for [2][1]: 8
Value for [2][2]: 9

Main menu:

1. Transpose of 3x3 matrix using array indices
2. Transpose of 3x3 matrix using only pointers
3. Exit

Select an option: 2
Printed matrix:
1  2  3
4  5  6
7  8  9

After Transpose:
Printed matrix:
1  4  7
2  5  8
3  6  9
```