Assignment 1

CSCI131

Name: James Marino

Username: jm617

Student Number: 4720994

Table Of Contents

1. Executive Summary

This report aims to provide demonstrations with bit manipulation and binary, octal and hexadecimal representations. The demonstrations were written in the C programming language.

The structure of the program is a test based quiz involving a series of exercises to complete by the user, such as the negation of a bit pattern.

Part 1
The following questions regarding bit manipulations may be asked with random values for a and b either in binary (base 2), octal (base 8) or hexadecimal (base 16):

- ~a
- a & b
- a & ~b
- a | b
- a ^ b
- ~(~a & ~b)

Part 2
Questions are asked regarding bit shifts where the value a is a value in either octal, binary or hexadecimal and c is a constant integer.

- a << c
- a >> c

There are two tests for both Logical and Arithmetic shifts.

2. Introduction

The report will show the following code and appropriate comments made for the test in regards to bit manipulations and shifts. It will also show the correct output and variations of results.

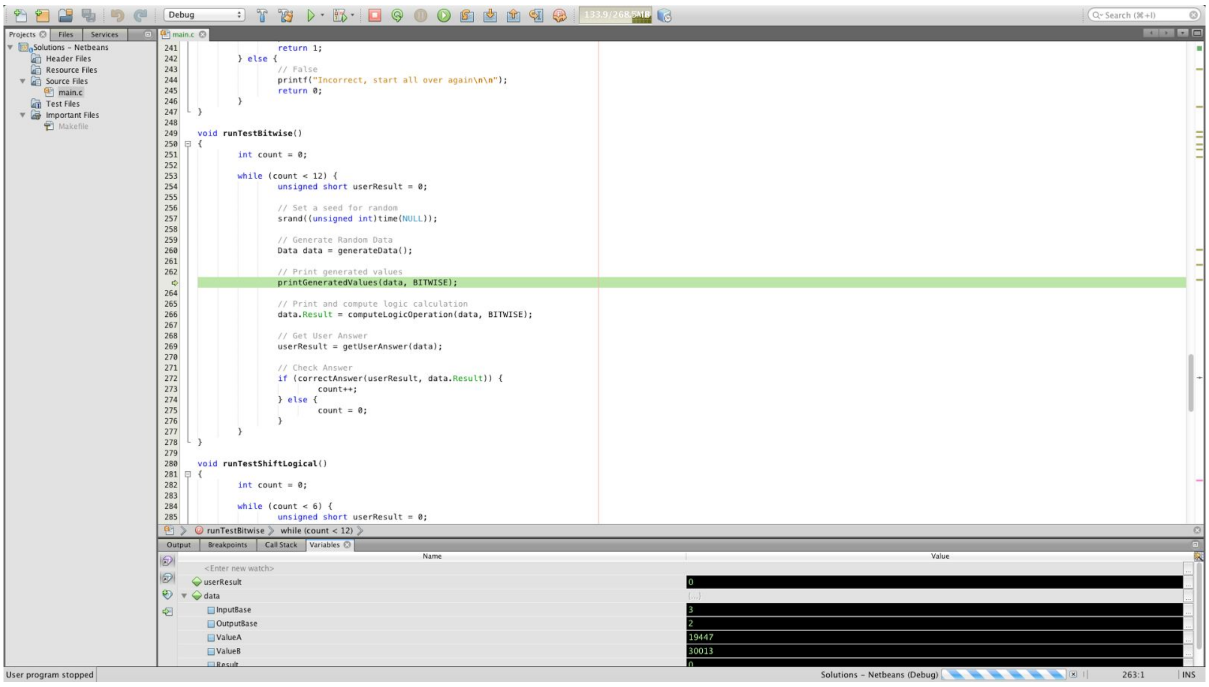
3. Body

3.1 Netbeans IDE

The Netbeans IDE was used to create a C based Application project.

Features of Netbeans used in Assignment:

- Breakpoints and stepping through code
- Syntax highlighting and code completion



```c
                return 1;
        } else {
                // False
                printf("Incorrect, start all over again\n\n");
                return 0;
        }
}

void runTestBitwise()
{
        int count = 0;

        while (count < 12) {
                unsigned short userResult = 0;

                // Set a seed for random
                srand((unsigned int)time(NULL));

                // Generate Random Data
                Data data = generateData();

                // Print generated values
                printGeneratedValues(data, BITWISE);

                // Print and compute logic calculation
                data.Result = computeLogicOperation(data, BITWISE);

                // Get User Answer
                userResult = getUserAnswer(data);

                // Check Answer
                if (correctAnswer(userResult, data.Result)) {
                        count++;
                } else {
                        count = 0;
                }
        }
}

void runTestShiftLogical()
{
        int count = 0;

        while (count < 6) {
                unsigned short userResult = 0;
```

Name	Value
<Enter new watch>	
userResult	0
data	{...}
InputBase	3
OutputBase	2
ValueA	19447
ValueB	30013
Result	0

As we can see in the screenshot above many useful functions and data of the IDE are displayed

- Variables Panel
  - We are able to see variables and their values at any point of execution
- Call stack
  - Stack of functions being used in the layers of their calls
- Step Over, Step Into and Step Out
  - All useful functions in navigating in and around the code

3.2 Code for Project

Code that Runs quiz for conversion and bit operations between values:

File: main.c

```
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <limits.h>
#include <string.h>
#include <ctype.h>

struct Data {
	int InputBase;
	int OutputBase;
	unsigned short ValueA;
	unsigned short ValueB;
	unsigned short Result;
};

typedef enum {BITWISE, SHIFT_LOGICAL, SHIFT_ARITHMETIC} QuestionType;
typedef struct Data Data;

int getRandomNumberBetween(int min, int max)
{
	int random = min;

	// Gaurd
	while (1) {
		random = (rand() % (max + 1)) + min;

		if ((random >= min) && (random <= max)) {
			break;
		}
	}

	return random;
}

Data generateData()
{
	Data temp;

	temp.InputBase = getRandomNumberBetween(1, 3);
	temp.OutputBase = getRandomNumberBetween(1, 3);
	temp.ValueA = getRandomNumberBetween(0, SHRT_MAX);
	temp.ValueB = getRandomNumberBetween(0, SHRT_MAX);
	temp.Result = 0;

	return temp;
}

unsigned short computeLogicOperation(Data data, QuestionType type)
{
```

```
unsigned short result = 0;

if (type == BITWISE) {
      int random = getRandomNumberBetween(0, 5);

      switch (random) {
            case 0:
                  printf("~a");
                  result = ~data.ValueA;
                  break;

            case 1:
                  printf("a & b");
                  result = data.ValueA & data.ValueB;
                  break;

            case 2:
                  printf("a & ~b");
                  result = data.ValueA & ~data.ValueB;
                  break;

            case 3:
                  printf("a | b");
                  result = data.ValueA | data.ValueB;
                  break;

            case 4:
                  printf("a ^ b");
                  result = data.ValueA ^ data.ValueB;
                  break;

            case 5:
                  printf("~(~a & ~b)");
                  result = ~(~data.ValueA & ~data.ValueB);
                  break;

            default:
                  result = 0;
                  break;
      }

} else if ((type == SHIFT_LOGICAL) || (type == SHIFT_ARITHMETIC)) {

      int random = getRandomNumberBetween(0, 1);
      int randomValue = getRandomNumberBetween(1, 3);

      switch (random) {
            case 0:
                  printf("a << %d", randomValue);
                  result = data.ValueA << randomValue;
                   break;

             case 1:
                   printf("a >> %d", randomValue);
                   result = data.ValueA >> randomValue;
```

```
                break;

            default:
                result = 0;
                break;
        }

    }

    printf("\n");
    return result;
}

void printBinary(unsigned short decimal)
{
    int mask = SHRT_MIN, count = 0;

    for (count = 0; count < 16; count++) {
        char ch = (decimal & mask) ? '1' : '0';
        decimal = decimal << 1;
        printf("%c", ch);
    }
}

unsigned short scanBinary() {
    char buf[256];
    fgets(buf, sizeof(buf), stdin);

    unsigned short val = 0;
    int len = (int)strlen(buf);
    int i = 0;

    while ((i < len) && (isspace(buf[i]))) {
        i++;
    }

    while ((i < len) && ((buf[i] == '0') || (buf[i] == '1'))) {
        val = val << 1;

        if (buf[i] == '1') {
            val = val | 01;
        }

        i++;
    }

    return val;
}

void printGeneratedValues(Data data, QuestionType type)
{
    switch (data.InputBase) {
        case 1:
            if (type == BITWISE) {
                printf("a = 0x%hx\nb = 0x%hx\n", data.ValueA, data.ValueB);
            } else {
                printf("a = 0x%hx\n", data.ValueA);
            }
            break;

        case 2:
            if (type == BITWISE) {
                printf("a = ");
                printBinary(data.ValueA);
                printf("\n");

                printf("b = ");
                printBinary(data.ValueB);
                printf("\n");
            } else {
                printf("a = ");
                printBinary(data.ValueA);
                printf("\n");
            }
            break;

        case 3:
            if (type == BITWISE) {
                printf("a = 0%ho\nb = 0%ho\n", data.ValueA, data.ValueB);
            } else {
                printf("a = 0%ho\n", data.ValueA);
            }
            break;

        default:
            break;
    }
}

unsigned short getUserAnswer(Data data)
{
    unsigned short temp = 0;

    printf("Enter Answer (Base 10: %d) in ", data.Result);

    switch (data.OutputBase) {
        case 1:
            printf("Hexadecimal: ");
            scanf("%hx", &temp);
            break;

        case 2:
            printf("Binary: ");
            temp = scanBinary();
            break;

        case 3:
            printf("Octal: ");
            scanf("%ho", &temp);
            break;

            default:
                break;
    }

    return temp;
}

int correctAnswer(unsigned short user, unsigned short machine)
{
    if (user == machine) {
        // True
        printf("Correct\n\n");
        return 1;
    } else {
        // False
        printf("Incorrect, start all over again\n\n");
        return 0;
    }
}

int correctAnswerArithmetic(signed short user, signed short machine)
{
    if (user == machine) {
        // True
        printf("Correct\n\n");
        return 1;
    } else {
        // False
        printf("Incorrect, start all over again\n\n");
        return 0;
    }
}

void runTestBitwise()
{
    int count = 0;

    while (count < 12) {
        unsigned short userResult = 0;

        // Set a seed for random
        srand((unsigned int)time(NULL));

        // Generate Random Data
        Data data = generateData();

        // Print generated values
        printGeneratedValues(data, BITWISE);

        // Print and compute logic calculation
        data.Result = computeLogicOperation(data, BITWISE);

        // Get User Answer
        userResult = getUserAnswer(data);

        // Check Answer
        if (correctAnswer(userResult, data.Result)) {
            count++;
        } else {
            count = 0;
        }
    }
}

void runTestShiftLogical()
{
    int count = 0;

    while (count < 6) {
        unsigned short userResult = 0;

        // Set a seed for random
        srand((unsigned int)time(NULL));

        // Generate Random Data
        Data data = generateData();

        // Print generated values
        printGeneratedValues(data, SHIFT_LOGICAL);

        // Print and compute logic calculation
        data.Result = computeLogicOperation(data, SHIFT_LOGICAL);

        // Get User Answer
        userResult = getUserAnswer(data);

        // Check Answer
        if (correctAnswer(userResult, data.Result)) {
            count++;
        } else {
            count = 0;
        }
    }
}

void runTestShiftArithmetic()
{
    int count = 0;

    while (count < 6) {
        unsigned short userResult = 0;

        // Set a seed for random
        srand((unsigned int)time(NULL));

        // Generate Random Data
        Data data = generateData();

        // Print generated values
```

```
            printGeneratedValues(data, SHIFT_ARITHMETIC);

            // Print and compute logic calculation
            data.Result = computeLogicOperation(data, SHIFT_ARITHMETIC);

            // Get User Answer
            userResult = getUserAnswer(data);

            // Cast to signed, Arithmetic Shifting
            // Check Answer
            if (correctAnswer((signed short)userResult, (signed short)data.Result)) {
                count++;
            } else {
                count = 0;
            }
        }
}

int main(int argc, const char * argv[]) {

        printf("Bitwise Operations\n");
        runTestBitwise();

        printf("Logical Shift Operations\n");
        runTestShiftLogical();

        printf("Arithmetic Shift Operations\n");
        runTestShiftArithmetic();

        return 0;
}
```

3.3 Results

Note: Extra debug output was added to the test including results of the the bit operations in Base 10 for easier debugging and testing

Bitwise Operations
Test run once through

- As we can see in the output, when a user types in an incorrect input, the counter of correct questions is reset

```
Bitwise Operations
a = 0x2fce
b = 0x1037
~a
Enter Answer (Base 10: 53297) in Binary: 1101000000110001
Correct

a = 0011000101111111
b = 0011101100101001
~(~a & ~b)
Enter Answer (Base 10: 15231) in Octal: 35577
Correct

a = 0x2bf5
b = 0x9fb
a ^ b
Enter Answer (Base 10: 8718) in Binary: 420
Incorrect, start all over again

a = 0x28ec
b = 0x3007
~a
Enter Answer (Base 10: 55059) in Binary: 1101011100010011
Correct

a = 0110000000101101
b = 0011110000000110
~a
Enter Answer (Base 10: 40914) in Hexadecimal: 9FD2
Correct

a = 0x2bf5
b = 0x9fb
a ^ b
Enter Answer (Base 10: 8718) in Binary: 10001000001110
Correct

a = 0000001000100101
b = 0101011111000100
a ^ b
Enter Answer (Base 10: 21985) in Hexadecimal: 55E1
Correct
```

```
a = 036434
b = 020076
a & ~b
Enter Answer (Base 10: 7424) in Octal: 16400
Correct

a = 0x3208
b = 0x3de2
~a
Enter Answer (Base 10: 52727) in Binary: 1100110111110111
Correct

a = 020730
b = 067540
a & b
Enter Answer (Base 10: 8512) in Binary: 10000101000000
Correct

a = 011051
b = 044147
~a
Enter Answer (Base 10: 60886) in Hexadecimal: EDD6
Correct

a = 0001011101010001
b = 0110100001011011
a & b
Enter Answer (Base 10: 81) in Octal: 121
Correct

a = 020556
b = 054634
a ^ b
Enter Answer (Base 10: 30962) in Octal: 74362
Correct

a = 0010010001110011
b = 0000010001001011
a & b
Enter Answer (Base 10: 1091) in Octal: 2103
Correct

a = 0011111010101000
b = 0001011001001000
~a
Enter Answer (Base 10: 49495) in Hexadecimal: C157
Correct
```

Shift Operations
Note: Is in same executable

```
Logical Shift Operations
a = 0x439f
a << 2
Enter Answer (Base 10: 3708) in Binary: 111001111100
Correct

a = 0101100011111001
a >> 3
Enter Answer (Base 10: 2847) in Hexadecimal: 0B1F
Correct

a = 0x76d9
a >> 3
Enter Answer (Base 10: 3803) in Binary: 111011011011
Correct

a = 0100010100100111
a << 1
Enter Answer (Base 10: 35406) in Hexadecimal: 8A4E
Correct

a = 012766
a << 2
Enter Answer (Base 10: 22488) in Hexadecimal: 57D8
Correct

a = 030053
a >> 1
Enter Answer (Base 10: 6165) in Octal: 14025
Correct

Arithmetic Shift Operations
a = 0010111010101110
a << 2
Enter Answer (Base 10: 47800) in Hexadecimal: BAB8
Correct

a = 047560
a << 1
Enter Answer (Base 10: 40672) in Binary: 1001111011100000
Correct

a = 0110010000011011
a << 1
Enter Answer (Base 10: 51254) in Binary: 1100100000110110
Correct

a = 02335
a << 3
Enter Answer (Base 10: 9960) in Octal: 23350
Correct
```

```
a = 0x527d
a << 2
Enter Answer (Base 10: 18932) in Binary: 100100111110100
Correct

a = 0x2d30
a << 3
Enter Answer (Base 10: 27008) in Binary: 110100110000000
Correct
```

4. Conclusion

As we can see the program runs as expected and provides the appropriate output to calculations and answers as seen in the results above.