

# Assignment 1

## CSCI131

**Name: James Marino**

**Username: jm617**

**Student Number: 4720994**

## Table Of Contents

1. Executive Summary
2. Introduction
3. Body
  - 3.1. Netbeans IDE
  - 3.2. Code for Project
  - 3.3. Results
4. Conclusion

### 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.

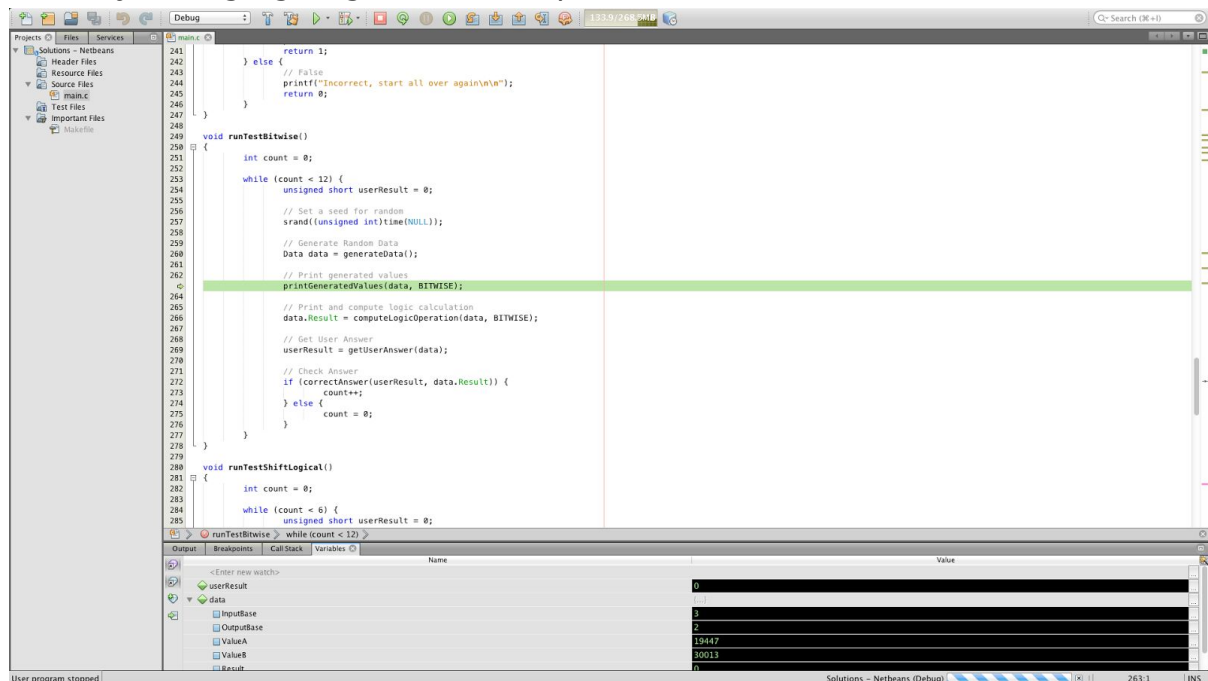
### 3. Body

#### 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



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

```
1 #include <time.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <limits.h>
5 #include <string.h>
6 #include <ctype.h>
7
8 struct Data {
9     int InputBase;
10    int OutputBase;
11    unsigned short ValueA;
12    unsigned short ValueB;
13    unsigned short Result;
14 };
15
16 typedef enum {BITWISE, SHIFT_LOGICAL, SHIFT_ARITHMETIC} QuestionType;
17 typedef struct Data Data;
18
19 int getRandomNumberBetween(int min, int max)
20 {
21     int random = min;
22
23     // Gaurd
24     while (1) {
25         random = (rand() % (max + 1)) + min;
26
27         if ((random >= min) && (random <= max)) {
28             break;
29         }
30     }
31
32     return random;
33 }
34
35 Data generateData()
36 {
37     Data temp;
38
39     temp.InputBase = getRandomNumberBetween(1, 3);
40     temp.OutputBase = getRandomNumberBetween(1, 3);
41     temp.ValueA = getRandomNumberBetween(0, SHRT_MAX);
42     temp.ValueB = getRandomNumberBetween(0, SHRT_MAX);
43     temp.Result = 0;
44
45     return temp;
46 }
47
48 unsigned short computeLogicOperation(Data data, QuestionType type)
49 {
```

```
50     unsigned short result = 0;
51
52     if (type == BITWISE) {
53         int random = getRandomNumberBetween(0, 5);
54
55         switch (random) {
56             case 0:
57                 printf("~a");
58                 result = ~data.ValueA;
59                 break;
60
61             case 1:
62                 printf("a & b");
63                 result = data.ValueA & data.ValueB;
64                 break;
65
66             case 2:
67                 printf("a & ~b");
68                 result = data.ValueA & ~data.ValueB;
69                 break;
70
71             case 3:
72                 printf("a | b");
73                 result = data.ValueA | data.ValueB;
74                 break;
75
76             case 4:
77                 printf("a ^ b");
78                 result = data.ValueA ^ data.ValueB;
79                 break;
80
81             case 5:
82                 printf("~(~a & ~b)");
83                 result = ~(~data.ValueA & ~data.ValueB);
84                 break;
85
86             default:
87                 result = 0;
88                 break;
89         }
90
91     } else if ((type == SHIFT_LOGICAL) || (type == SHIFT_ARITHMETIC)) {
92
93         int random = getRandomNumberBetween(0, 1);
94         int randomValue = getRandomNumberBetween(1, 3);
95
96         switch (random) {
97             case 0:
98                 printf("a << %d", randomValue);
99                 result = data.ValueA << randomValue;
100                break;
101
102             case 1:
103                 printf("a >> %d", randomValue);
104                 result = data.ValueA >> randomValue;
```

```
105         break;
106
107     default:
108         result = 0;
109         break;
110     }
111 }
112 }
113
114 printf("\n");
115 return result;
116 }
117
118 void printBinary(unsigned short decimal)
119 {
120     int mask = SHRT_MIN, count = 0;
121
122     for (count = 0; count < 16; count++) {
123         char ch = (decimal & mask) ? '1' : '0';
124         decimal = decimal << 1;
125         printf("%c", ch);
126     }
127 }
128
129 unsigned short scanBinary() {
130     char buf[256];
131     fgets(buf, sizeof(buf), stdin);
132
133     unsigned short val = 0;
134     int len = (int)strlen(buf);
135     int i = 0;
136
137     while ((i < len) && (isspace(buf[i]))) {
138         i++;
139     }
140
141     while ((i < len) && ((buf[i] == '0') || (buf[i] == '1'))) {
142         val = val << 1;
143
144         if (buf[i] == '1') {
145             val = val | 01;
146         }
147
148         i++;
149     }
150
151     return val;
152 }
153
154 void printGeneratedValues(Data data, QuestionType type)
155 {
156     switch (data.InputBase) {
157     case 1:
158         if (type == BITWISE) {
159             printf("a = 0x%hx\nb = 0x%hx\n", data.ValueA, data.ValueB);
```

```
160         } else {
161             printf("a = 0x%hx\n", data.ValueA);
162         }
163         break;
164
165     case 2:
166         if (type == BITWISE) {
167             printf("a = ");
168             printBinary(data.ValueA);
169             printf("\n");
170
171             printf("b = ");
172             printBinary(data.ValueB);
173             printf("\n");
174         } else {
175             printf("a = ");
176             printBinary(data.ValueA);
177             printf("\n");
178         }
179         break;
180
181     case 3:
182         if (type == BITWISE) {
183             printf("a = 0%ho\nb = 0%ho\n", data.ValueA, data.ValueB);
184         } else {
185             printf("a = 0%ho\n", data.ValueA);
186         }
187         break;
188
189     default:
190         break;
191 }
192 }
193
194 unsigned short getUserAnswer(Data data)
195 {
196     unsigned short temp = 0;
197
198     printf("Enter Answer (Base 10: %d) in ", data.Result);
199
200     switch (data.OutputBase) {
201     case 1:
202         printf("Hexadecimal: ");
203         scanf("%hx", &temp);
204         break;
205
206     case 2:
207         printf("Binary: ");
208         temp = scanBinary();
209         break;
210
211     case 3:
212         printf("Octal: ");
213         scanf("%ho", &temp);
214         break;
```

```
215
216     default:
217         break;
218 }
219
220 return temp;
221 }
222
223 int correctAnswer(unsigned short user, unsigned short machine)
224 {
225     if (user == machine) {
226         // True
227         printf("Correct\n\n");
228         return 1;
229     } else {
230         // False
231         printf("Incorrect, start all over again\n\n");
232         return 0;
233     }
234 }
235
236 int correctAnswerArithmetic(signed short user, signed short machine)
237 {
238     if (user == machine) {
239         // True
240         printf("Correct\n\n");
241         return 1;
242     } else {
243         // False
244         printf("Incorrect, start all over again\n\n");
245         return 0;
246     }
247 }
248
249 void runTestBitwise()
250 {
251     int count = 0;
252
253     while (count < 12) {
254         unsigned short userResult = 0;
255
256         // Set a seed for random
257         srand((unsigned int)time(NULL));
258
259         // Generate Random Data
260         Data data = generateData();
261
262         // Print generated values
263         printGeneratedValues(data, BITWISE);
264
265         // Print and compute logic calculation
266         data.Result = computeLogicOperation(data, BITWISE);
267
268         // Get User Answer
269         userResult = getUserAnswer(data);
```



```
270
271     // Check Answer
272     if (correctAnswer(userResult, data.Result)) {
273         count++;
274     } else {
275         count = 0;
276     }
277 }
278 }
279
280 void runTestShiftLogical()
281 {
282     int count = 0;
283
284     while (count < 6) {
285         unsigned short userResult = 0;
286
287         // Set a seed for random
288         srand((unsigned int)time(NULL));
289
290         // Generate Random Data
291         Data data = generateData();
292
293         // Print generated values
294         printGeneratedValues(data, SHIFT_LOGICAL);
295
296         // Print and compute logic calculation
297         data.Result = computeLogicOperation(data, SHIFT_LOGICAL);
298
299         // Get User Answer
300         userResult = getUserAnswer(data);
301
302         // Check Answer
303         if (correctAnswer(userResult, data.Result)) {
304             count++;
305         } else {
306             count = 0;
307         }
308     }
309 }
310
311 void runTestShiftArithmetic()
312 {
313     int count = 0;
314
315     while (count < 6) {
316         unsigned short userResult = 0;
317
318         // Set a seed for random
319         srand((unsigned int)time(NULL));
320
321         // Generate Random Data
322         Data data = generateData();
323
324         // Print generated values
```

```
325     printGeneratedValues(data, SHIFT_ARITHMETIC);
326
327     // Print and compute logic calculation
328     data.Result = computeLogicOperation(data, SHIFT_ARITHMETIC);
329
330     // Get User Answer
331     userResult = getUserAnswer(data);
332
333     // Cast to signed, Arithmetic Shifting
334     // Check Answer
335     if (correctAnswer((signed short)userResult, (signed short)data.Result)) {
336         count++;
337     } else {
338         count = 0;
339     }
340 }
341 }
342
343 int main(int argc, const char * argv[]) {
344
345     printf("Bitwise Operations\n");
346     runTestBitwise();
347
348     printf("Logical Shift Operations\n");
349     runTestShiftLogical();
350
351     printf("Arithmetic Shift Operations\n");
352     runTestShiftArithmetic();
353
354     return 0;
355 }
356
```

### 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.