LAB GOALS

- Understanding datatypes and bitwise operators

Step 1: Start with a simple C++ console project called I308_Lab2.

Step 2: Add a new (main.cpp) item (file) under the source files.
Step 3: The first thing we want to do is to understand our basic data types and how they are constructed in our compiler. Create the following program and run it. (Make sure to include the necessary header files, and namespaces)

```cpp
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

void main()
{
    unsigned char clearChar;
    cout << "====Understanding your Compiler and Datatypes====" << endl;
    cout << "Size of char:             " << sizeof(char) << endl;
    cout << "Size of unsigned char:    " << sizeof(unsigned char) << endl;
    cout << "Size of unicode widechar: " << sizeof(wchar_t) << endl;
    cout << "Size of int:              " << sizeof(int) << endl;
    cout << "Size of float:            " << sizeof(float) << endl;
    cout << "Size of double:           " << sizeof(double) << endl;
    getchar();
}
```

Step 4: The next thing we should learn is to display information in different formats. This will come in quite handy as we get into lower-level programming. Sure you have a debugger which may be able to show you this data. But you should really know how to do this yourself, and without a debugger! Add the following code to your main program.

```cpp
cout << "==============Displaying Information in Different Formats====================" << endl;
unsigned int aNum;
aNum = 65;
cout << "-------------- aNum (int) --------------" << endl;
cout << "char format    = " << char(aNum) << endl;
cout << "decimal format = " << std::dec << (aNum) << endl;
cout << "Octal format   = " << std::oct << (aNum) << endl;
cout << "HEX format     = " << std::hex << (aNum) << endl;
```

Run the above to see the results. Now add the following code as well and re-run the code. This should have the same results (however using the printf() language which is inherited from the "C" language.

```cpp
cout << "================================-" << endl;
printf("Oct format(C)  = %c \n",aNum); // C style printf printing Char number.
printf("Hex format(C)  = %d \n",aNum); // C style printf printing Decimal number.
printf("Oct format(C)  = %o \n",aNum); // C style printf printing OCTal number.
printf("Hex format(C)  = %x \n",aNum); // C style printf printing HEXadecimal number.
cout << "================================-" << endl;
```
Step 5: Now let’s create a function to display the actual bit pattern of our data. We do this in 3 steps.

1) Add a forward declaration for a function `int_to_binary()` before the main function. This function will take an unsigned integer value and converts it in to any array of characters (string) and returns it. string 
int_to_binary(unsigned int value);

```c++
// Convert Unsigned integer into a string of bits..
string int_to_binary(unsigned int value) {
    char theResult[128];
    unsigned int mask = 0x80000000;
    int i;
    for (i = 0; i < sizeof(value) * 8; i++) {
        if ((value & mask) == 0)
            theResult[i] = '0';
        else
            theResult[i] = '1';
        mask >>= 1;
    }
    theResult[i] = '\0';
    string result = theResult;
    return(result);
}
```

2) Add the actual `int_to_binary()` after your main() function.

3) In the main() function, call the `int_to_binary()` function with the integer value of aNum.
```c++
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;
// Forward declarations:
string int_to_binary(unsigned int value);
void main() {
    //...add the new code above the getchar() function...
    cout << "Binary format  = " << int_to_binary(aNum) << endl;
```
Now run the program again to see the results:

```
---Understanding your Compiler and Datatypes---
Size of char: 1
Size of unsigned char: 1
Size of unicode wchar: 2
Size of int: 4
Size of Float: 4
Size of double: 8

---Displaying Information in Different Formats---
Char format (C++) = Α
Dec format (C++) = 55
Oct format (C++) = 101
HEX format (C++) = 35

Char format (C) = 0
Dec format (C) = 65
Oct format (C) = 101
HEX format (C) = 41

Binary format = 000000000000000000000001000001
```

Now that you see how one can look at data in different representations, why don’t you try to do the same thing for “Char” data type. See if you can also print the bit representation for it. You may have to make a new copy of the int_to_binary() and modify it a little to accommodate char_to_binary(); See below for a little clue!

```cpp
string char_to_binary(unsigned char value) {
    char theResult[128];
    unsigned char mask = 0x80;
    int i;

    for (i = 0; i < sizeof(value) * 8; i++) {
        if ((value & mask) == 0)
            theResult[i] = '0';
        else
            theResult[i] = '1';
        mask >>= 1;
    }
    theResult[i] = '\0';
    string result = theResult;
    return(result);
}
```

Go ahead and add the above function to your program. Then make sure that you also include a forward declaration for it about the main() function. We’ll use this function later on.
Step 6: The next thing we want to do is to understand the bitwise operators. These include the\n\n\( \text{shift-LEFT } \ll \), \( \text{shift-Right } \gg \), \( \text{bitwise-OR } | \), \( \text{bitwise-AND } & \), \( \text{bitwise-XOR } ^\vee \), \( \text{bitwise-NOT } ! \) \) and \( \text{bitwise-COMPLEMENT } ~ \).

To test the **SHIFT operators**, add the following code above the getchar():

```cpp
Step 6: The next thing we want to do is to understand the bitwise operators. These include the (shift-LEFT \ll), (shift-Right \gg), (bitwise-OR |), (bitwise-AND &), (bitwise-XOR ^), (bitwise-NOT !) and (bitwise-COMPLEMENT ~).

To test the **SHIFT operators**, add the following code above the getchar():

```cpp
Step 6: The next thing we want to do is to understand the bitwise operators. These include the (shift-LEFT \ll), (shift-Right \gg), (bitwise-OR |), (bitwise-AND &), (bitwise-XOR ^), (bitwise-NOT !) and (bitwise-COMPLEMENT ~).

To test the **SHIFT operators**, add the following code above the getchar():

```cpp
Step 6: The next thing we want to do is to understand the bitwise operators. These include the (shift-LEFT \ll), (shift-Right \gg), (bitwise-OR |), (bitwise-AND &), (bitwise-XOR ^), (bitwise-NOT !) and (bitwise-COMPLEMENT ~).

To test the **SHIFT operators**, add the following code above the getchar():

```cpp
```
Step 9: Now to test the XOR operator add the following code (above the getchar() line) and run your program.

```
cout << "=====================XOR Operation=====================
" << endl;
clearChar = 0x09;
cout << "0000 1001 ^ 0000 1100 = " << char_to_binary(clearChar ^ 0x0C) << endl;
cout << "=================Double XOR Operation=====================
Back to the original char: " << endl;
cout << char_to_binary(clearChar) << " ^ 0000 1100 ^ 0000 1100 = " << char_to_binary(clearChar ^ 0x0C ^ 0x0C) << endl;
```
Step 10: Now to test the **NOT operator** add the following code (above the getchar() line) and run your program.

```cpp
    cout << " ===================NOT Operation==================" << endl;
    clearChar = 0x05; // a non-zero number
    cout << "~0000 1001 = " << char_to_binary(!clearChar) << endl;
    clearChar = 0x00; // a zero
    cout << "~0000 0000 = " << char_to_binary(!clearChar) << endl;
```

Step 11: Now to test the **COMPLEMENT operator** add the following code (above the getchar() line) and run your program.

```cpp
    cout << " ===================Complement Operation==================" << endl;
    clearChar = 0x09;
    cout << "~0000 1001 = " << char_to_binary(~clearChar) << endl;
```

Optional:

If you are still interested to learn more, perhaps about 32bit floating point numbers in VC++ and how their bit patterns are organized, see below:
// INFO-I 308 Information Representation
// Proof of concept exercise for assignment 4
// Understanding and use of bitwise operators.
// Hossein Hakimzadeh
// 3/3/2014
//
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

// Forward declarations:

string char_to_binary(unsigned char value);
string int_to_binary(unsigned int value);
string f32_to_binary(float value);

unsigned int ChangeEndian32(unsigned int value); // Change byte order

void main()
{
    cout << "====Understanding your Compiler and Datatypes====" << endl;
    cout << "Size of char:             " << sizeof(char) << endl;
    cout << "Size of unsigned char:    " << sizeof(unsigned char) << endl;
    cout << "Size of unicode widechar: " << sizeof(wchar_t) << endl;
    cout << "Size of int:              " << sizeof(int) << endl;
    cout << "Size of float:            " << sizeof(float) << endl;
    cout << "Size of double:           " << sizeof(double) << endl;
    cout << "==============Displaying Information in Different Formats====================" << endl;
    unsigned int aNum;
aNum = 65;
    cout << "------------ aNum (int) --------------" << endl;
    cout << "Char format(C++)    = " << char(aNum) << endl;
    cout << "Dec format(C++)     = " << std::dec << (aNum) << endl;
    cout << "Oct format(C++)     = " << std::oct << (aNum) << endl;
    cout << "HEX format(C++)     = " << std::hex << (aNum) << endl;
    cout << "----------------------------------" << endl;
    printf("Char format(C) = %c 
",aNum); // C style printf printing Char number.
    printf("Dec format(C) = %d 
",aNum); // C style printf printing Decimal number.
    printf("Oct format(C) = %o 
",aNum); // C style printf printing OCTal number.
    printf("Hex format(C) = %x 
",aNum); // C style printf printing HEXadecimal number.
    cout << "----------------------------------" << endl;
    cout << "Binary format  = " << int_to_binary(aNum) << endl;
    cout << "----------------------------------" << endl;

    // Now we try the same thing with a char
    unsigned char aChar;
aChar = 'A';
aChar = 'A';
    cout << "------------- aChar --------------" << endl;
    cout << "Char format(C++)    = " << char(aChar) << endl;
    cout << "Dec format(C++) = " << int(aChar) << endl;
    cout << "Octal format(C++)   = " << std::oct << int(aChar) << endl;
    cout << "HEX format(C++)     = " << std::hex << int(aChar) << endl;
    cout << "----------------------------------" << endl;
    printf("char format(C) = %c \n",aChar); // C style printf printing Char number.
    printf("Dec format(C) = %d \n",aChar); // C style printf printing Decimal number.
    printf("Oct format(C) = %o \n",aChar); // C style printf printing OCTal number.
    printf("Hex format(C) = %x \n",aChar); // C style printf printing HEXadecimal number.
    cout << "----------------------------------" << endl;

    // Now we try the same thing with a float
    float afloat;
    afloat = 2.0;
    cout << "------------- Float -------------" << endl;
    cout << "Binary format  = " << f32_to_binary(afloat) << endl;
    cout << "----------------------------------" << endl;
}
afloat = -2.0;
cout << "----------- Float ---------------" << endl;
cout << "Binary format = " << f32_to_binary(afloat) << endl;
cout << "-------------------------------" << endl;

afloat = 4.0;
cout << "----------- Float ---------------" << endl;
cout << "Binary format = " << f32_to_binary(afloat) << endl;
cout << "-------------------------------" << endl;

afloat = 6.0;
cout << "----------- Float ---------------" << endl;
cout << "Binary format = " << f32_to_binary(afloat) << endl;
cout << "-------------------------------" << endl;

afloat = 1;
cout << "----------- Float ---------------" << endl;
cout << "Binary format = " << f32_to_binary(afloat) << endl;
cout << "-------------------------------" << endl;

afloat = 0.75;
cout << "----------- Float ---------------" << endl;
cout << "Binary format = " << f32_to_binary(afloat) << endl;
cout << "-------------------------------" << endl;

afloat = 2.5;
cout << "----------- Float ---------------" << endl;
cout << "Binary format = " << f32_to_binary(afloat) << endl;
cout << "-------------------------------" << endl;

afloat = 0.1;
cout << "----------- Float ---------------" << endl;
cout << "Binary format = " << f32_to_binary(afloat) << endl;
cout << "-------------------------------" << endl;

afloat = 0.75;
cout << "----------- Float ---------------" << endl;
cout << "Binary format = " << f32_to_binary(afloat) << endl;
cout << "-------------------------------" << endl;

cout << "==============Understanding Bitwise Shift Operations=====================
unsigned char clearChar;
clearChar = 0x01;
cout << "0x01 in Binary format   = " << char_to_binary(clearChar) << endl;
clearChar = (clearChar << 2);
cout << "Shift left 2 bits       = " << char_to_binary(clearChar) << endl;
clearChar = (clearChar >> 1);
cout << "Shift right 1 bit       = " << char_to_binary(clearChar) << endl;
clearChar = (clearChar << 6);
cout << "Shift left 6 bits       = " << char_to_binary(clearChar) << endl;

cout << "====================Shifting left..=========================
for(int i = 0; i< sizeof(unsigned char) * 8; i++)
cout << std::setw(4) << int(clearChar << i) << " [Binary format = " << char_to_binary(clearChar << i) << "]" << endl;


cout << "====================Shifting right.=========================
for(int i = 0; i< sizeof(unsigned char) * 8; i++)
cout << std::setw(4) << int(clearChar >> i) << " [Binary format = " << char_to_binary(clearChar >> i) << "]" << endl;


cout << "==================== OR Operation=========================
clearChar = 0x01;
cout << "0000 0001 | 00001100 = " << (clearChar | 0x0C ) << endl;
cout << "Hex:                   " << std::hex << (clearChar | 0x0C ) << endl;
cout << "Binary Result:         " << char_to_binary(clearChar | 0x0C ) << endl;


cout << "=====================AND Operation=========================
clearChar = 0x01;
cout << "0000 0001 & 00001100 = " << (clearChar & 0x0C ) << endl;
cout << "Hex:                   " << std::hex << (clearChar & 0x0C ) << endl;
cout << "Binary Result:         " << char_to_binary(clearChar & 0x0C ) << endl;

cout << "================================================================================

11
clearChar = 0xff;
cout << "1111 1111 & 00001100 = " << (clearChar & 0x0C) << endl;
cout << "hex: " << std::hex << (clearChar & 0x0C) << endl;
cout << "Binary Result: " << char_to_binary(clearChar & 0x0C) << endl;
cout << "============================XOR Operation======================================" << endl;
clearChar = 0x09;
cout << "0000 1001 ^ 0000 1100 = " << char_to_binary(clearChar ^ 0x0C) << endl;
cout << "Hex:                   " << std::hex << (clearChar & 0x0C) << endl;
cout << "Binary Result:         " << char_to_binary(clearChar & 0x0C) << endl;
cout << "=================Double XOR Operation========================================" << endl;
clearChar = 0x09; // a non-zero number
cout << "-0000 1001 = " << char_to_binary(!clearChar) << endl;
clearChar = 0x00; // a zero
cout << "-0000 0000 = " << char_to_binary(!clearChar) << endl;
cout << "=================Complement Operation========================================" << endl;
clearChar = 0x09;
cout << "-0000 1001 = " << char_to_binary(~clearChar) << endl;
/**/
getchar();
}

string char_to_binary(unsigned char value) {
    char theResult[128];
    unsigned char mask = 0x80;
    int i;

    for (i = 0; i < sizeof(value) * 8; i++) {
        if ((value & mask) == 0)
            theResult[i] = '0';
        else
            theResult[i] = '1';
        mask >>= 1;
    }
    theResult[i] = '\0';
    string result = theResult;
    return(result);
}

string int_to_binary(unsigned int value) {
    char theResult[128];
    unsigned int mask = 0x80000000;
    int i;

    for (i = 0; i < sizeof(value) * 8; i++) {
        if ((value & mask) == 0)
            theResult[i] = '0';
        else
            theResult[i] = '1';
        mask >>= 1;
    }
    theResult[i] = '\0';
    string result = theResult;
    return(result);
}

// The bit pattern for float (4byte) was verified using the following
// Microsoft web site:
//
// IEEE Floating-Point Representation (Visual Studio 2015)
//
// Note this function will not work for DOUBLE (64 bit) double
// precision floating point numbers.

string f32_to_binary(float value)
{
    cout << "Float value    = " << value << endl;

    // Changing the byte order for float appears to be necessary.
    // Otherwise printing the bits produces incorrect byte order.
    // Also we need to cast the float into a unsigned int before
    // sending it to the ChangeEndian32() function, because
    // bitwise operators can not be applied to float data type.
    unsigned int *intPtr;
    intPtr = (unsigned int *) &value;
    *intPtr = ChangeEndian32(*intPtr);     // change byte order

    unsigned char *startingByte;
    startingByte = (unsigned char *) &value;
    char theResult[128];
    int j = 0;
    for (int numBytes = 3; numBytes >= 0; numBytes--) {
        for (int numBits = 7; numBits >= 0; numBits--) {
            if (((startingByte >> numBits) & 1) == 1)
                theResult[j] = '1';
            else
                theResult[j] = '0';
            j++;
            //cout << ((startingByte >> numBits) & 1);
            startingByte++;
            //cout << endl;
        }
    }
    theResult[j] = '\0';
    string result = theResult;
    return(result);
}

unsigned int ChangeEndian32(unsigned int value)
{
    return ((value & 0xFF000000ul) >> 24)  |
            ((value & 0x00FF0000ul) >> 8)  |
            ((value & 0x0000FF00ul) << 8)  |
            ((value & 0x000000FFul) << 24);