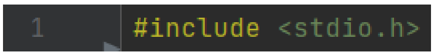
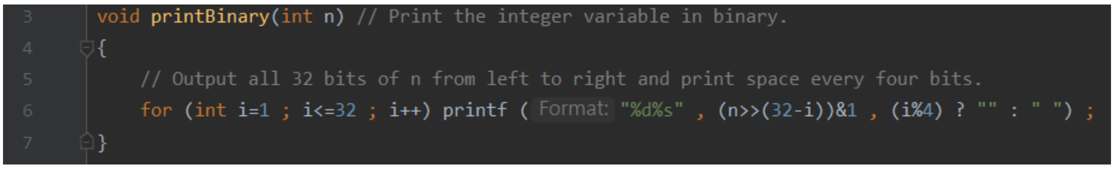
Name: Jack
Student ID: D1166506

In order to fulfill the required function, I separated and coded the program into 3 parts. Since there aren't any extra required operations to deal with, I only used the stdio.h library.



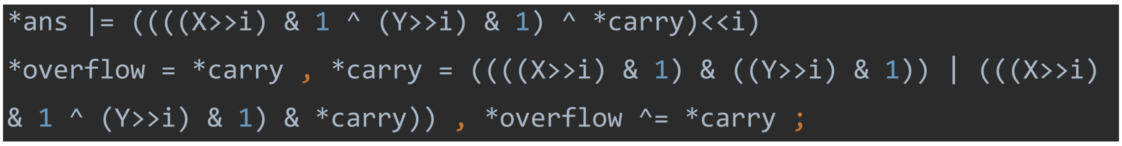
```c
#include <stdio.h>
```

The first part of this program is to print all 32 bits of the variables in binary. To do this, I use the loop and right shift operator (>>) to get and print each bit of the integer variable n and print a space every 4 bits.

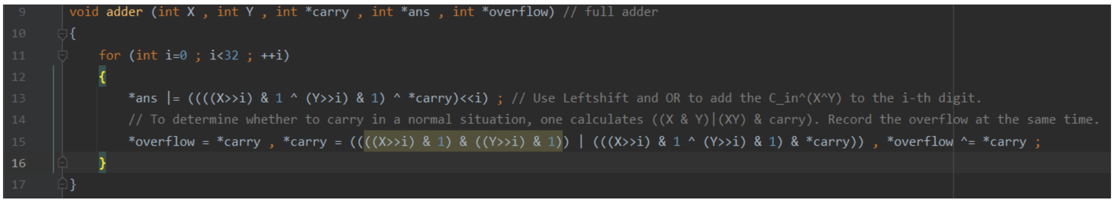


```c
void printBinary(int n) // Print the integer variable in binary.
{
    // Output all 32 bits of n from left to right and print space every four bits.
    for (int i=1 ; i<=32 ; i++) printf ( Format: "%d%s" , (n>>(32-i))&1 , (i%4) ? "" : " ") ;
}
```

Next, and the most important part, is to make a full adder. Because calling a half adder into a full adder will increase the work of the program, I made a full adder only instead. In the function adder, it has to input X and Y variables in integer type and the addresses of carry, ans, and overflow. In order to get the answer to the equation and determine whether the equation is overflowing or not, I use the following equation to assign the value of the answer. Especially for overflow, I record the carry in value first, then do an XOR operation with the carry out value. After these operations, I can get the answer and determine the status of overflow easily.

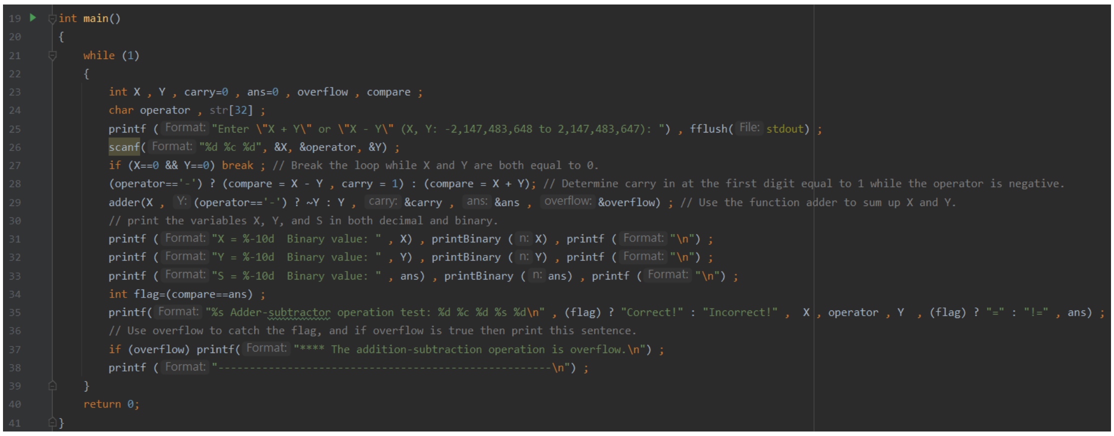


```c
*ans |= ((((X>>i) & 1 ^ (Y>>i) & 1) ^ *carry)<<i)
*overflow = *carry , *carry = ((((X>>i) & 1) & ((Y>>i) & 1)) | (((X>>i)
& 1 ^ (Y>>i) & 1) & *carry)) , *overflow ^= *carry ;
```



```c
void adder (int X , int Y , int *carry , int *ans , int *overflow) // full adder
{
    for (int i=0 ; i<32 ; ++i)
    {
        *ans |= ((((X>>i) & 1 ^ (Y>>i) & 1) ^ *carry)<<i) ; // Use Leftshift and OR to add the C_in^(X^Y) to the i-th digit.
        // To determine whether to carry in a normal situation, one calculates ((X & Y)|(XY) & carry). Record the overflow at the same time.
        *overflow = *carry , *carry = ((((X>>i) & 1) & ((Y>>i) & 1)) | (((X>>i) & 1 ^ (Y>>i) & 1) & *carry)) , *overflow ^= *carry ;
    }
}
```

Last but not least, in the main function, I scan for the input variables X, operator, and Y and break while X and Y are both equal to 0. Next, set the carry in value as 1 while the operator variable is ‘-‘; otherwise, set the value as 0. Then, call the function adder to get the answer and overflow status and print the X, Y, and ans variables in decimal and binary base. Finally, print the sentence according to its correctness and the origin equation.



```c
int main()
{
    while (1)
    {
        int X , Y , carry=0 , ans=0 , overflow , compare ;
        char operator , str[32] ;
        printf ( Format: "Enter \"X + Y\" or \"X - Y\" (X, Y: -2,147,483,648 to 2,147,483,647): ") , fflush( File: stdout) ;
        scanf( Format: "%d %c %d", &X, &operator, &Y) ;
        if (X==0 && Y==0) break ; // Break the loop while X and Y are both equal to 0.
        (operator=='-') ? (compare = X - Y , carry = 1) : (compare = X + Y); // Determine carry in at the first digit equal to 1 while the operator is negative.
        adder(X , Y: (operator=='-') ? ~Y : Y , carry: &carry , ans: &ans , overflow: &overflow) ; // Use the function adder to sum up X and Y.
        // print the variables X, Y, and S in both decimal and binary.
        printf ( Format: "X = %-10d  Binary value: " , X) , printBinary ( n: X) , printf ( Format: "\n") ;
        printf ( Format: "Y = %-10d  Binary value: " , Y) , printBinary ( n: Y) , printf ( Format: "\n") ;
        printf ( Format: "S = %-10d  Binary value: " , ans) , printBinary ( n: ans) , printf ( Format: "\n") ;
        int flag=(compare==ans) ;
        printf( Format: "%s Adder-subtractor operation test: %d %c %d %s %d\n" , (flag) ? "Correct!" : "Incorrect!" ,  X , operator , Y  , (flag) ? "=" : "!=" , ans) ;
        // Use overflow to catch the flag, and if overflow is true then print this sentence.
        if (overflow) printf( Format: "**** The addition-subtraction operation is overflow.\n") ;
        printf ( Format: "-----------------------------------------------------\n") ;
    }
    return 0;
}
```