**SOURCE CODE**

0 | #include <stdio.h>

1 | #include <math.h>

2 | #include <stdlib.h>

3 | ​

4 | ​

5 | int requiredBits(int n) {

6 |     if (n == 0) return 2;

7 |     return (int)(log2(n > 0 ? n + 1 : abs(n))) + 2;

8 | }

9 | ​

10| void dectobinary(int nbit,int num, int arry[]){

11|  while(nbit>=0){

12| ​

13|  if(num>0){

14|  arry[nbit-1] = num%2;

15|  num = num/2;

16|  nbit--;

17|  }

18|  else{

19|  arry[nbit-1] = 0;

20|  nbit--;

21|  }

22|  }

23|

24| }

25| ​

26| ​

27| void twoComplement(int arry[], int nbit){

28|  int bitsum,i,carry=1;

29|

30|  for(i=0;i<nbit;i++){

31|  arry[i] = !arry[i];

32|  }

33|

34|  for(i=nbit-1;i>=0;i--){

35|  bitsum = arry[i]+carry;

36|  arry[i] = bitsum%2;

37|  carry = bitsum/2;

38|  }

39| }

40| ​

41| void showbinary(int \*p, int size) {

42|     int i;

43|     for (i = 0; i<size; i++) {

44|         printf("%d ", \*(p+i));

45|    }

46|     printf("\n");

47| }

48| ​

49| ​

50| int addbinary(int arry1[],int arry2[],int nbit,int sum[]){

51|  int i, bitsum,carry =0;

52|  for(i=nbit-1;i>=0;i--){

53|  bitsum = arry1[i]+arry2[i]+carry;

54|  sum[i] = bitsum%2;

55|  carry = bitsum/2;

56|  }

57|  return carry;

58| }

59| ​

60| int binarytodec(int arry[], int nbit) {

61|     int result = 0, i;

62| ​

63|     if (arry[0] == 1) {

64|         int temp[32];

65|         for (i = 0; i < nbit; i++) temp[i] = arry[i];

66|         twoComplement(temp, nbit);

67| ​

68|         for (i = 0; i < nbit; i++) {

69|             result += temp[nbit - 1 - i] \* (1 << i);

70|        }

71| ​

72|         return -result;

73|    } else {

74|         for (i = 0; i < nbit; i++) {

75|             result += arry[nbit - 1 - i] \* (1 << i);

76|        }

77| ​

78|         return result;

79|    }

80| }

81| ​

82| ​

83| ​

84| ​

85| int main() {

86|     int num1, num2;

87|     int arry1[32] = {0}, arry2[32] = {0}, sum[32] = {0};

88| ​

89|     printf("Enter the two numbers: ");

90|     scanf("%d %d", &num1, &num2);

91| ​

92|     // Dynamically calculate the number of bits needed

93|     int bits1 = requiredBits(num1);

94|     int bits2 = requiredBits(num2);

95|     int nbit = (bits1 > bits2) ? bits1 : bits2;

96| ​

97|     if (nbit > 32) {

98|         printf("Number too large to handle in 32 bits.\n");

99|         return 1;

100|    }

101| ​

102|     // Convert num1

103|     if (num1 < 0) {

104|         dectobinary(nbit, abs(num1), arry1);

105|         twoComplement(arry1, nbit);

106|    } else {

107|         dectobinary(nbit, num1, arry1);

108|    }

109|     printf("Binary of %d: ", num1);

110|     showbinary(arry1, nbit);

111| ​

112|     // Convert num2

113|     if (num2 < 0) {

114|         dectobinary(nbit, abs(num2), arry2);

115|         twoComplement(arry2, nbit);

116|    } else {

117|         dectobinary(nbit, num2, arry2);

118|    }

119|     printf("Binary of %d: ", num2);

120|     showbinary(arry2, nbit);

121| ​

122|     // Add both binaries

123|     int carry\_out = addbinary(arry1, arry2, nbit, sum);

124|     printf("Sum in binary: ");

125|     showbinary(sum, nbit);

126| ​

127|     int result = binarytodec(sum, nbit);

128|     printf("Decimal equivalent of sum: %d\n", result);

129| //overflow

130|     if ((num1 > 0 && num2 > 0 && result < 0) ||

131|        (num1 < 0 && num2 < 0 && result > 0)) {

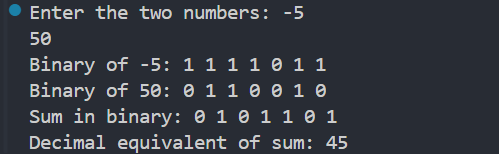
132|         printf("Overflow Detected\n");

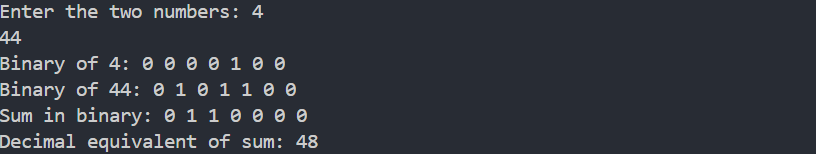
133|    }

134| ​

135|     return 0;

136| }

**OUTPUT**



*Fig: Output of Code*