Experiment No. 7
Implement Booth's algorithm using c-programming
Name: Archita Deepak Gupta
Roll Number: 19
Date of Performance:
Date of Submission:

Aim: To implement Booth's algorithm using c-programming.

Objective -

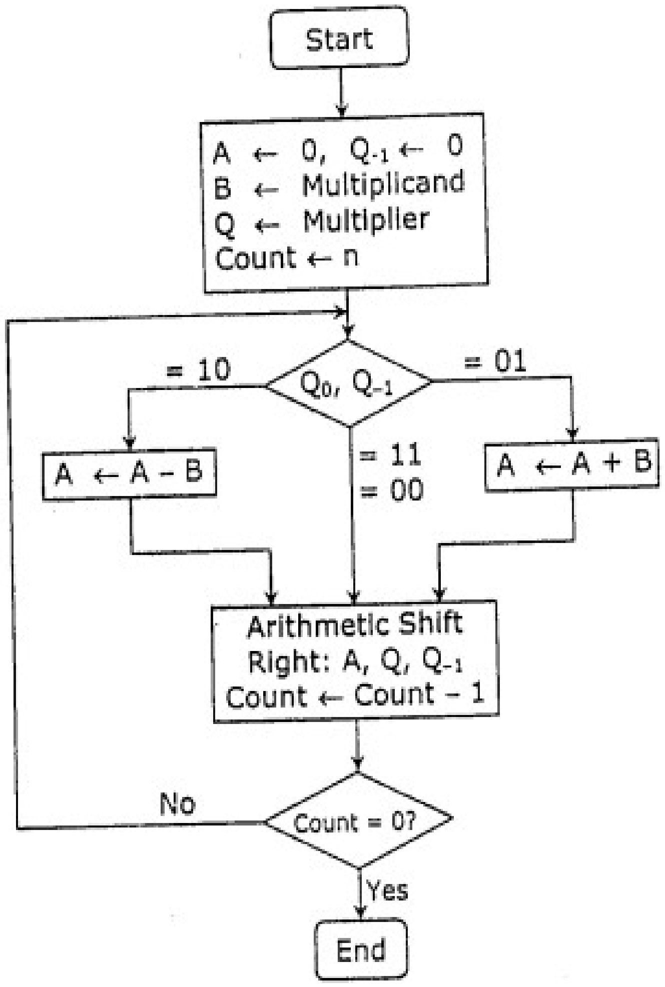
1. To understand the working of Booths algorithm.
2. To understand how to implement Booth's algorithm using c-programming.

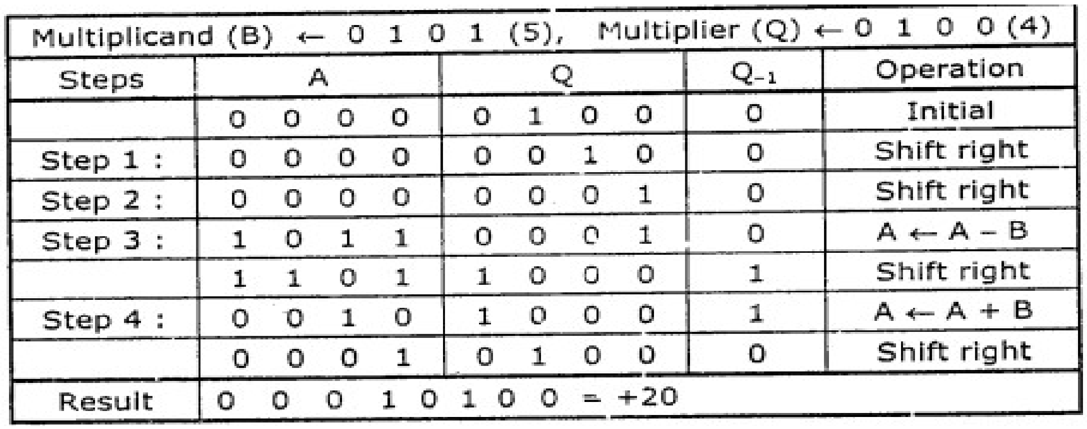
Theory:

Booth's algorithm is a multiplication algorithm that multiplies two signed binary numbers in 2's complement notation. Booth used desk calculators that were faster at shifting than adding and created the algorithm to increase their speed.

The algorithm works as per the following conditions :

1. If Qn and Q_{-1} are same i.e. 00 or 11 perform arithmetic shift by 1 bit.
2. If Qn Q_{-1} = 10 do A= A - B and perform arithmetic shift by 1 bit.
3. If Qn Q_{-1} = 01 do A= A + B and perform arithmetic shift by 1 bit.





Multiplicand (B) ← 0 1 0 1 (5), Multiplier (Q) ← 0 1 0 0 (4)				
Steps	A	Q	Q_{-1}	Operation
	0 0 0 0	0 1 0 0	0	Initial
Step 1 :	0 0 0 0	0 0 1 0	0	Shift right
Step 2 :	0 0 0 0	0 0 0 1	0	Shift right
Step 3 :	1 0 1 1	0 0 0 1	0	A ← A − B
	1 1 0 1	1 0 0 0	1	Shift right
Step 4 :	0 0 1 0	1 0 0 0	1	A ← A + B
	0 0 0 1	0 1 0 0	0	Shift right
Result	0 0 0 1 0 1 0 0 = +20			

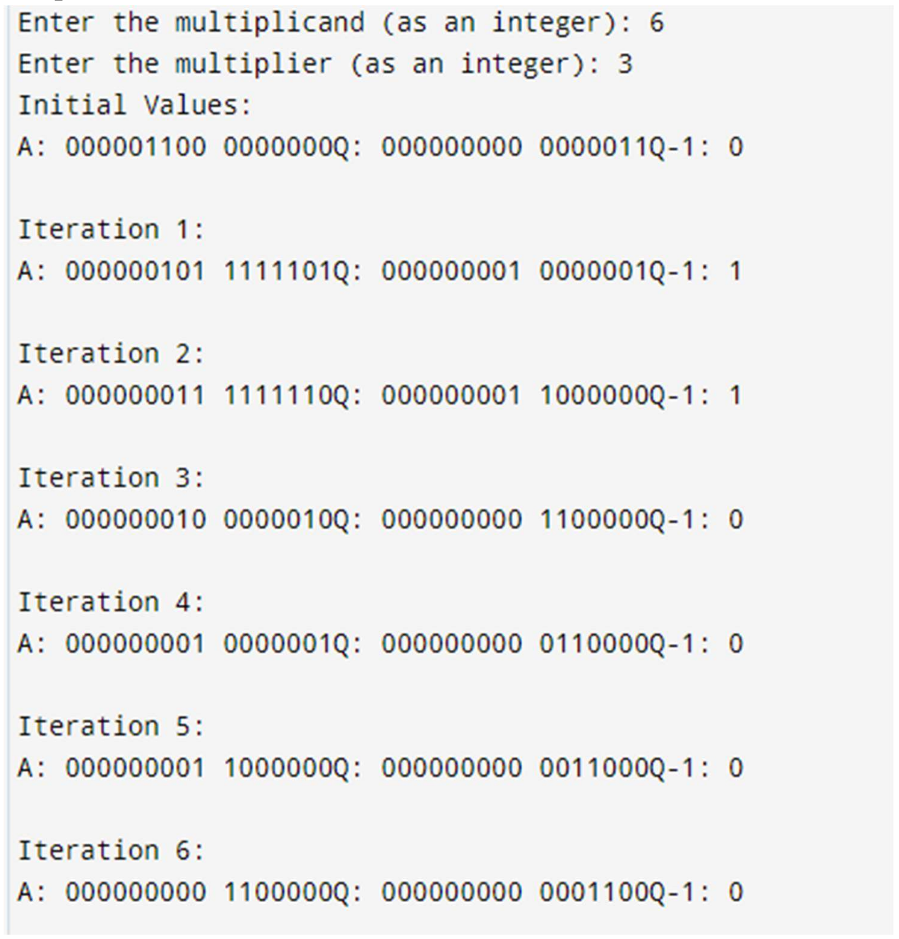
Program:

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

// Define the number of bits for the multiplication
#define N 8

// Function prototypes
void boothAlgorithm(int multiplicand, int multiplier);
void printBinary(int num);

int main() {
    int multiplicand, multiplier;

    // Input the multiplicand and multiplier
    printf("Enter the multiplicand (as an integer): ");
    scanf("%d", &multiplicand);
    printf("Enter the multiplier (as an integer): ");
    scanf("%d", &multiplier);

    // Perform Booth's Algorithm
    boothAlgorithm(multiplicand, multiplier);

    return 0;
}

// Function to perform Booth's algorithm
void boothAlgorithm(int multiplicand, int multiplier) {
```

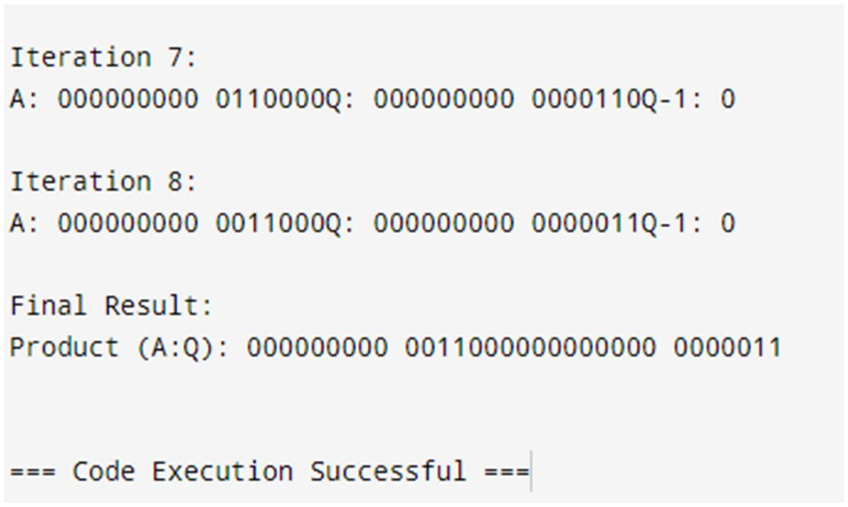
```
int A = multiplicand;                // Accumulator
int Q = multiplier;                 // Multiplier
int M = multiplicand;               // Multiplicand
int Q_1 = 0;                        // Q-1 (initialized to 0)
int Q_temp;                         // Temporary variable for shifts

// Initialize A and Q
A = (A & ((1 << N) - 1)) << N;      // Extend sign bit if necessary
Q = Q & ((1 << N) - 1);             // Mask to keep only N bits

printf("Initial Values:\n");
printf("A: ");
printBinary(A);
printf("Q: ");
printBinary(Q);
printf("Q-1: %d\n", Q_1);

for (int i = 0; i < N; i++) {
    printf("\nIteration %d:\n", i + 1);

    // Check Q0 and Q-1
    if ((Q & 1) == 0 && Q_1 == 1) {
        // Q0Q-1 == 01: A = A + M
        A = (A + M) & ((1 << (2 * N)) - 1);
    } else if ((Q & 1) == 1 && Q_1 == 0) {
        // Q0Q-1 == 10: A = A - M
        A = (A - M) & ((1 << (2 * N)) - 1);
    }

    // Arithmetic shift right
    Q_1 = Q & 1;
    Q_temp = (A & 1) << (N - 1);  // Save the MSB of A
    A = (A >> 1) | Q_temp;        // Shift A right and insert MSB of A
    Q = (Q >> 1) | (Q_1 << (N - 1));  // Shift Q right and insert Q-1

    // Print the values after the shift
    printf("A: ");
    printBinary(A);
    printf("Q: ");
    printBinary(Q);
    printf("Q-1: %d\n", Q_1);
}

// The result is in A and Q
```

```
    printf("\nFinal Result:\n");
    printf("Product (A:Q): ");
    printBinary(A);
    printBinary(Q);
    printf("\n");
}

// Function to print the binary representation of a number
void printBinary(int num) {
    for (int i = (N * 2 - 1); i >= 0; i--) {
        printf("%d", (num >> i) & 1);
        if (i == N - 1) printf(" ");  // Space between A and Q
    }
}
```

Output:



```
Enter the multiplicand (as an integer): 6
Enter the multiplier (as an integer): 3
Initial Values:
A: 000001100 0000000Q: 000000000 0000011Q-1: 0

Iteration 1:
A: 000000101 1111101Q: 000000001 0000001Q-1: 1

Iteration 2:
A: 000000011 1111110Q: 000000001 1000000Q-1: 1

Iteration 3:
A: 000000010 0000010Q: 000000000 1100000Q-1: 0

Iteration 4:
A: 000000001 0000001Q: 000000000 0110000Q-1: 0

Iteration 5:
A: 000000001 1000000Q: 000000000 0011000Q-1: 0

Iteration 6:
A: 000000000 1100000Q: 000000000 0001100Q-1: 0
```



```
Iteration 7:
A: 000000000 0110000Q: 000000000 0000110Q-1: 0

Iteration 8:
A: 000000000 0011000Q: 000000000 0000011Q-1: 0

Final Result:
Product (A:Q): 000000000 0011000000000000 0000011


=== Code Execution Successful ===
```

Conclusion - I conclude that I have understood the working of Booth's algorithm and how to implement Booth’s algorithm using c-programming.