BCNF:
According to the algorithm in the book.

Code:

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

#define MAX_ATTR 4
#define MAX_FD 3
#define MAX_LEN 6

char attributes[MAX_ATTR][MAX_LEN];
char left[MAX_FD][MAX_LEN];
char right[MAX_FD][MAX_LEN];
char candidate[MAX_ATTR];
int nattr, nfd;
int found[26] = {0};

void readInput() {
  printf("Enter number of attributes: ");
  scanf("%d", &nattr);
  printf("Enter attributes (space separated): ");
  for (int i = 0; i < nattr; i++) {
    scanf("%s", attributes[i]);
    if (isalpha(attributes[i][0]))
      found[tolower(attributes[i][0]) - 'a'] = 1;
  }
  printf("Enter number of functional dependencies: ");
  scanf("%d", &nfd);
  getchar();  // Consume the newline character left by scanf
  for (int i = 0; i < nfd; i++) {
    printf("Enter the functional dependencies LHS for %d: ", i);
    scanf("%s", left[i]);
  }
  for (int i = 0; i < nfd; i++) {
    printf("Enter the functional dependencies RHS for %d: ", i);
    scanf("%s", right[i]);
  }
  printf("Enter the candidate key: ");
  scanf("%s", candidate);
}
```

```
int isSubset(const char *set, const char *subset) {
    for (int i = 0; subset[i] != '\0'; i++) {
        if (!strchr(set, subset[i])) {
            return 0; // Not a subset
        }
    }
    return 1; // Is a subset
}

int Set(const char *s1, const char *s2) {
    if (strlen(s1) != strlen(s2))
        return 0;
    for (int i = 0; s1[i] != '\0'; i++) {
        if (strchr(s2, s1[i]) == NULL)
            return 0;
    }
    return 1;
}

char *closure(char *result) {
    int changed;
    do {
        changed = 0; // Reset changed flag
        for (int i = 0; i < nfd; i++) {
            if (isSubset(result, left[i])) {
                for (int j = 0; right[i][j] != '\0'; j++) {
                    if (!strchr(result, right[i][j])) {
                        strncat(result, &right[i][j], 1);
                        changed = 1;
                    }
                }
            }
        }
    } while (changed);
    return result;
}

int notSet(char * a,char * b,char* c){
    for(int i=0;a[i]!='\0';i++){
        if(strchr(c,a[i])==NULL)
        return 1;
    }
    for(int i=0;b[i]!='\0';i++){
        if(strchr(c,b[i])==NULL)
        return 1;
    }
    return 0;
}


void find_closure(const char *attr) {
    int s1 = 0;
    int totalKeys = 1 << nattr;
    char *left_1[50] = {NULL};
    char *right_1[50] = {NULL};

    // Generate all combinations of attributes
    for (int i = 1; i < totalKeys; i++) {
        char key[MAX_ATTR + 1] = "";
        for (int j = 0; j < nattr; j++) {
            if (i & (1 << j)) {
                strncat(key, attributes[j], 1);

            }
        }

        char *new_key = strdup(key);
        char *rhs = closure(key);
        int length = strlen(rhs);
        int totalKeys_1 = 1 << length;

        // Generate all combinations of RHS
        for (int k = 1; k < totalKeys_1; k++) {
            char key_1[MAX_ATTR + 1] = "";
            for (int l = 0; l < length; l++) {
                if (k & (1 << l)) {
                    strncat(key_1, rhs + l, 1);
                }
            }
            left_1[s1] = new_key; // Store the new key
            right_1[s1] = strdup(key_1); // Store a copy of key_1
            s1++;
        }
    }

    // // Print generated pairs
    // for (int o = 0; o < s1; o++)
    //     printf("%s -> %s\n", left_1[o], right_1[o]);
```

```
int done = 0;
char *result[15] = {NULL};
result[0] = strdup(attr);
int t = 1;

// Iteratively find new closures
while (!done) {
    done = 1;
    for (int i = 0; i < t; i++) {
        for (int j = 0; j < s1; j++) {
            printf("%s -> %s\n", left_1[j], right_1[j]);
            if (isSubset(left_1[j], right_1[j]) || isSubset(right_1[j], left_1[j])){
                printf("1st\n");
                continue; // Skip if subset condition met
            }
            if (notSet(left_1[j], right_1[j], result[i])) {
                printf("2nd\n");
                continue; // Skip if not set condition met
            }
            char temp_x[5];
            strcpy(temp_x,left_1[j]);
            if(isSubset(closure(temp_x), result[i])){
                printf("3rd\n");
                continue;
            }

            // Construct new result
            done = 0;
            char a[MAX_ATTR + 1] = "";
            for (int k = 0; result[i][k] != '\0'; k++) {
                if (strchr(right_1[j], result[i][k]) == NULL) {
                    strncat(a, (char[]){result[i][k], '\0'}, 1);
                }
            }
            result[i] = strdup(a); // Update with new closure
            char *temp = malloc(strlen(left_1[j]) + strlen(right_1[j]) + 1);
            if (!temp) {
                fprintf(stderr, "Memory allocation failed!\n");
                exit(EXIT_FAILURE);
            }
            strcpy(temp, left_1[j]);
            strcat(temp, right_1[j]);
            result[t++] = temp; // Store the new combined string
            }
        }
    }
    printf("The resulting sets are: \n");
    for (int i = 0; i < t; i++) {
        printf("%s\n", result[i]);
    }
}

int isbcnf(const char *attributes) {
    for(int i=0;i<nfd;i++){
        if(isSubset(left[i],right[i]) || (Set(left[i],candidate))){
            continue;
        }
        else
        {
            printf("Not in BCNF\n");
            find_closure(attributes);
            return 0;
        }
    }
    return 1;
}

int main() {
    readInput();
    char attr[MAX_ATTR * MAX_LEN] = ""; // Create a sufficiently large array
    for (int i = 0; i < nattr; i++) {
        strcat(attr, attributes[i]);
    }
    if (isbcnf(attr)) {
        printf("In BCNF\n");
    }
    return 0;
}
```

Output:

```
reethi@DESKTOP-8744EFO:~/dir1/dbms$ ./a.out
Enter number of attributes: 3
Enter attributes (space separated): a b c
Enter number of functional dependencies: 2
Enter the functional dependencies LHS for 0: a
Enter the functional dependencies LHS for 1: b
Enter the functional dependencies RHS for 0: b
Enter the functional dependencies RHS for 1: c
Enter the candidate key:
a
Not in BCNF
a -> a
1st
a -> b
3rd
a -> ab
1st
a -> c
3rd
a -> ac
1st
a -> bc
3rd
a -> abc
1st
b -> b
1st
b -> c
b -> bc
1st
ab -> a
1st
ab -> b
1st
ab -> ab
1st
ab -> c
2nd
ab -> ac
2nd
ab -> bc
2nd
ab -> abc
1st
c -> c
1st
ac -> a
1st
ac -> c
1st
ac -> ac
```

```
1st
ac -> b
2nd
ac -> ab
2nd
ac -> cb
2nd
ac -> acb
1st
bc -> b
1st
bc -> c
1st
bc -> bc
1st
abc -> a
1st
abc -> b
1st
abc -> ab
1st
abc -> c
1st
abc -> ac
1st
abc -> bc
1st
abc -> abc
1st
a -> a
1st
a -> b
2nd
a -> ab
1st
a -> c
2nd
a -> ac
1st
a -> bc
2nd
a -> abc
1st
b -> b
```

```
1st
b -> c
3rd
b -> bc
1st
ab -> a
1st
ab -> b
1st
ab -> ab
1st
ab -> c
2nd
ab -> ac
2nd
ab -> bc
2nd
ab -> abc
1st
c -> c
1st
ac -> a
1st
ac -> c
1st
ac -> ac
1st
ac -> b
2nd
ac -> ab
2nd
ac -> cb
2nd
ac -> acb
1st
bc -> b
1st
bc -> c
1st
bc -> bc
1st
abc -> a
1st
abc -> b
```

1st
abc -> ab
1st
abc -> c
1st
abc -> ac
1st
abc -> bc
1st
abc -> abc
1st
a -> a
1st
a -> b
3rd
a -> ab
1st
a -> c
2nd
a -> ac
1st
a -> bc
2nd
a -> abc
1st
b -> b
1st
b -> c
2nd
b -> bc
1st
ab -> a
1st
ab -> b
1st
ab -> ab
1st
ab -> c
2nd
ab -> ac
2nd
ab -> bc
2nd
ab -> abc

```
1st
c -> c
1st
ac -> a
1st
ac -> c
1st
ac -> ac
1st
ac -> b
2nd
ac -> ab
2nd
ac -> cb
2nd
ac -> acb
1st
bc -> b
1st
bc -> c
1st
bc -> bc
1st
abc -> a
1st
abc -> b
1st
abc -> ab
1st
abc -> c
1st
abc -> ac
1st
abc -> bc
1st
abc -> abc
1st
a -> a
1st
a -> b
2nd
a -> ab
1st
a -> c
```

2nd
a -> ac
1st
a -> bc
2nd
a -> abc
1st
b -> b
1st
b -> c
3rd
b -> bc
1st
ab -> a
1st
ab -> b
1st
ab -> ab
1st
ab -> c
2nd
ab -> ac
2nd
ab -> bc
2nd
ab -> abc
1st
c -> c
1st
ac -> a
1st
ac -> c
1st
ac -> ac
1st
ac -> b
2nd
ac -> ab
2nd
ac -> cb
2nd
ac -> acb
1st
bc -> b

```
1st
bc -> c
1st
bc -> bc
1st
abc -> a
1st
abc -> b
1st
abc -> ab
1st
abc -> c
1st
abc -> ac
1st
abc -> bc
1st
abc -> abc
1st
The resulting sets are:
ab
bc
```