

DBMS LAB

Name – RAVI SHEKHAR

Roll No. – 22CS3075

1.

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

```
#include <stdlib.h>
```

```
#define MAX_ATTR 10
```

```
#define MAX_DEP 10
```

```
struct FunctionalDependency {  
    char determinant[MAX_ATTR];  
    char dependent[MAX_ATTR];  
};
```

```
int main() {  
    int num_attributes, num_dependencies;  
    struct FunctionalDependency dependencies[MAX_DEP];  
  
    printf("Enter the number of attributes: ");  
    scanf("%d", &num_attributes);  
    if (num_attributes <= 0 || num_attributes > MAX_ATTR) {  
        printf("Invalid number of attributes.\n");  
        return 1;  
    }  
}
```

```

printf("Enter the number of functional dependencies: ");

scanf("%d", &num_dependencies);

if (num_dependencies <= 0 || num_dependencies > MAX_DEP) {
    printf("Invalid number of dependencies.\n");
    return 1;
}

printf("Enter the functional dependencies in the format (determinant -> dependent):\n");

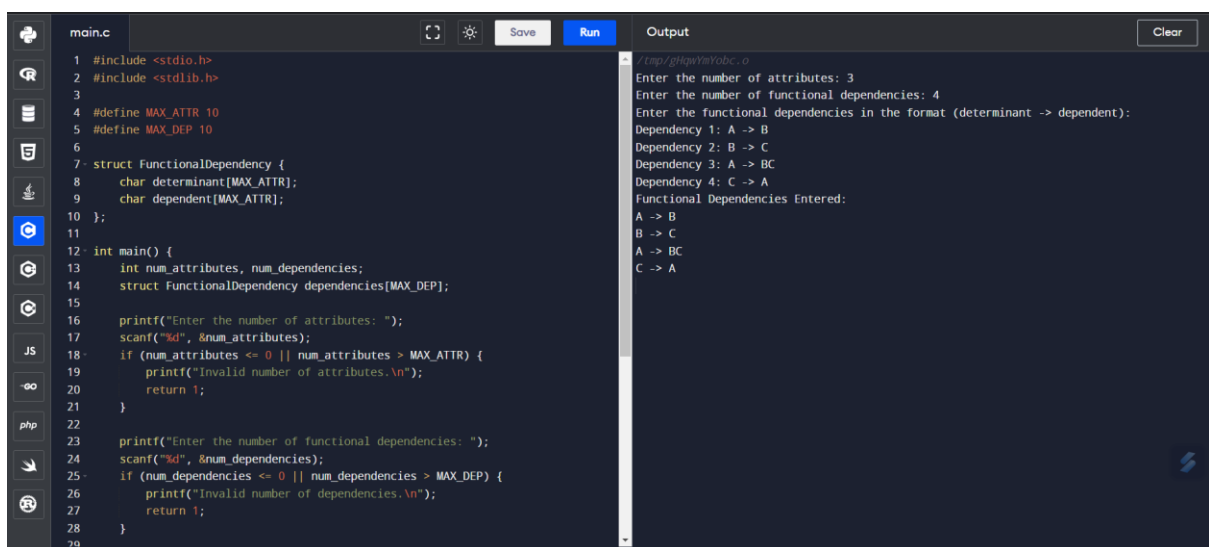
for (int i = 0; i < num_dependencies; i++) {
    printf("Dependency %d: ", i + 1);
    scanf("%s -> %s", dependencies[i].determinant, dependencies[i].dependent);
}

printf("\nFunctional Dependencies Entered:\n");

for (int i = 0; i < num_dependencies; i++) {
    printf("%s -> %s\n", dependencies[i].determinant, dependencies[i].dependent);
}

return 0;
}

```



The screenshot shows a C++ IDE with a file named `main.c`. The code is a C program that takes the number of attributes and functional dependencies as input, and then lists the functional dependencies. The output window shows the program's execution with the following text:

```

Enter the number of attributes: 3
Enter the number of functional dependencies: 4
Enter the functional dependencies in the format (determinant -> dependent):
Dependency 1: A -> B
Dependency 2: B -> C
Dependency 3: A -> BC
Dependency 4: C -> A
Functional Dependencies Entered:
A -> B
B -> C
A -> BC
C -> A

```

2.

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

```
#include <stdlib.h>
```

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

```
#define MAX_ATTR 10
```

```
#define MAX_DEP 10
```

```
struct FunctionalDependency {  
    char determinant[MAX_ATTR];  
    char dependent[MAX_ATTR];  
};
```

```
void computeClosure(char set[], int num_attributes, int num_dependencies, struct  
FunctionalDependency dependencies[], char closure[]) {
```

```
    int changed;
```

```
    // Initialize closure with the given set of attributes
```

```
    strcpy(closure, set);
```

```
    do {
```

```
        changed = 0;
```

```
        for (int i = 0; i < num_dependencies; i++) {
```

```
            int allInClosure = 1;
```

```
            // Check if the determinant is a subset of the closure
```

```
            for (int j = 0; j < strlen(dependencies[i].determinant); j++) {
```

```
                if (strchr(closure, dependencies[i].determinant[j]) == NULL) {
```

```
                    allInClosure = 0;
```

```
                    break;
```

```
                }
```

```

    }

    // If the determinant is a subset of the closure, add the dependent to the closure
    if (allInClosure) {
        int dependentLength = strlen(dependencies[i].dependent);
        for (int j = 0; j < dependentLength; j++) {
            if (strchr(closure, dependencies[i].dependent[j]) == NULL) {
                closure[strlen(closure)] = dependencies[i].dependent[j];
                changed = 1;
            }
        }
    }
} while (changed);
}

```

```

int main() {
    int num_attributes, num_dependencies;
    char set[MAX_ATTR], closure[MAX_ATTR];
    struct FunctionalDependency dependencies[MAX_DEP];

    printf("Enter the number of attributes: ");
    scanf("%d", &num_attributes);
    if (num_attributes <= 0 || num_attributes > MAX_ATTR) {
        printf("Invalid number of attributes.\n");
        return 1;
    }

    printf("Enter the set of attributes: ");
    scanf("%s", set);

```

```

printf("Enter the number of functional dependencies: ");

scanf("%d", &num_dependencies);

if (num_dependencies <= 0 || num_dependencies > MAX_DEP) {

    printf("Invalid number of dependencies.\n");

    return 1;

}

printf("Enter the functional dependencies in the format (determinant -> dependent):\n");

for (int i = 0; i < num_dependencies; i++) {

    printf("Dependency %d: ", i + 1);

    scanf("%s -> %s", dependencies[i].determinant, dependencies[i].dependent);

}

// Compute closure

computeClosure(set, num_attributes, num_dependencies, dependencies, closure);

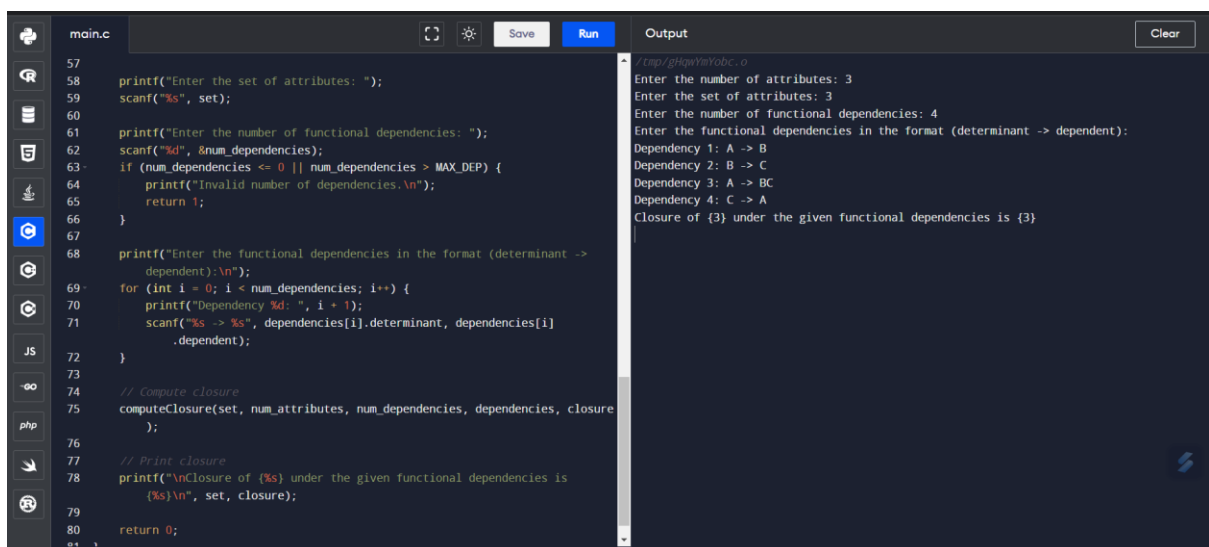
// Print closure

printf("\nClosure of {%s} under the given functional dependencies is {%s}\n", set, closure);

return 0;

}

```



The screenshot shows a C++ IDE with a file named `main.c`. The code implements a program to calculate the closure of a set of attributes under a set of functional dependencies. The code is as follows:

```

57
58 printf("Enter the set of attributes: ");
59 scanf("%s", set);
60
61 printf("Enter the number of functional dependencies: ");
62 scanf("%d", &num_dependencies);
63 if (num_dependencies <= 0 || num_dependencies > MAX_DEP) {
64     printf("Invalid number of dependencies.\n");
65     return 1;
66 }
67
68 printf("Enter the functional dependencies in the format (determinant ->
69     dependent):\n");
70 for (int i = 0; i < num_dependencies; i++) {
71     printf("Dependency %d: ", i + 1);
72     scanf("%s -> %s", dependencies[i].determinant, dependencies[i]
73         .dependent);
74 }
75
76 // Compute closure
77 computeClosure(set, num_attributes, num_dependencies, dependencies, closure
78 );
79
80 // Print closure
81 printf("\nClosure of {%s} under the given functional dependencies is
82 {%s}\n", set, closure);
83
84 return 0;

```

The Output window shows the following output:

```

/tmp/gHawYmYobc.o
Enter the number of attributes: 3
Enter the set of attributes: 3
Enter the number of functional dependencies: 4
Enter the functional dependencies in the format (determinant -> dependent):
Dependency 1: A -> B
Dependency 2: B -> C
Dependency 3: A -> BC
Dependency 4: C -> A
Closure of {3} under the given functional dependencies is {3}

```

3.

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

```
#include <stdlib.h>
```

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

```
#define MAX_ATTR 10
```

```
#define MAX_DEP 10
```

```
struct FunctionalDependency {  
    char determinant[MAX_ATTR];  
    char dependent[MAX_ATTR];  
};
```

```
void computeClosure(char set[], int num_attributes, int num_dependencies, struct  
FunctionalDependency dependencies[], char closure[]) {
```

```
    int changed;
```

```
    // Initialize closure with the given set of attributes
```

```
    strcpy(closure, set);
```

```
    do {
```

```
        changed = 0;
```

```
        for (int i = 0; i < num_dependencies; i++) {
```

```
            int allInClosure = 1;
```

```
            // Check if the determinant is a subset of the closure
```

```
            for (int j = 0; j < strlen(dependencies[i].determinant); j++) {
```

```
                if (strchr(closure, dependencies[i].determinant[j]) == NULL) {
```

```
                    allInClosure = 0;
```

```

        break;
    }
}

// If the determinant is a subset of the closure, add the dependent to the closure
if (allInClosure) {
    int dependentLength = strlen(dependencies[i].dependent);
    for (int j = 0; j < dependentLength; j++) {
        if (strchr(closure, dependencies[i].dependent[j]) == NULL) {
            closure[strlen(closure)] = dependencies[i].dependent[j];
            changed = 1;
        }
    }
}
} while (changed);
}

```

```

int isSuperkey(char attributes[], char closure[]) {
    // Check if closure contains all attributes in the given set
    for (int i = 0; i < strlen(attributes); i++) {
        if (strchr(closure, attributes[i]) == NULL) {
            return 0; // Not a superkey
        }
    }
    return 1; // It is a superkey
}

```

```

int isCandidateKey(char attributes[], char closure[], struct FunctionalDependency
dependencies[], int num_dependencies) {

```

```

// Check if it's a superkey first
if (!isSuperkey(attributes, closure)) {
    return 0; // Not a candidate key if not a superkey
}

// Check if removing any attribute makes it not a superkey
for (int i = 0; i < strlen(attributes); i++) {
    char modifiedClosure[MAX_ATTR] = "";
    strncpy(modifiedClosure, closure, strlen(closure));
    char withoutAttr[MAX_ATTR] = "";
    strcpy(withoutAttr, attributes);
    memmove(&withoutAttr[i], &withoutAttr[i + 1], strlen(withoutAttr) - i); // Remove one attribute

    computeClosure(withoutAttr, strlen(withoutAttr), num_dependencies, dependencies, modifiedClosure);

    if (isSuperkey(withoutAttr, modifiedClosure)) {
        return 0; // If removing an attribute still forms a superkey, not a candidate key
    }
}

return 1; // It is a candidate key
}

```

```

int isPrimaryKey(char attributes[], char closure[], struct FunctionalDependency dependencies[], int num_dependencies) {
    return isCandidateKey(attributes, closure, dependencies, num_dependencies);
}

```

```

int main() {
    int num_attributes, num_dependencies;
    char set[MAX_ATTR], closure[MAX_ATTR];

```



```

struct FunctionalDependency dependencies[MAX_DEP];

printf("Enter the number of attributes: ");
scanf("%d", &num_attributes);
if (num_attributes <= 0 || num_attributes > MAX_ATTR) {
    printf("Invalid number of attributes.\n");
    return 1;
}

printf("Enter the set of attributes: ");
scanf("%s", set);

printf("Enter the number of functional dependencies: ");
scanf("%d", &num_dependencies);
if (num_dependencies <= 0 || num_dependencies > MAX_DEP) {
    printf("Invalid number of dependencies.\n");
    return 1;
}

printf("Enter the functional dependencies in the format (determinant -> dependent):\n");
for (int i = 0; i < num_dependencies; i++) {
    printf("Dependency %d: ", i + 1);
    scanf("%s -> %s", dependencies[i].determinant, dependencies[i].dependent);
}

// Compute closure
computeClosure(set, num_attributes, num_dependencies, dependencies, closure);

// Check if the set of attributes is a superkey, candidate key, or primary key

```

```

    if (isSuperkey(set, closure)) {
        printf("\nThe set of attributes is a superkey.\n");
    } else {
        printf("\nThe set of attributes is not a superkey.\n");
    }

    if (isCandidateKey(set, closure, dependencies, num_dependencies)) {
        printf("The set of attributes is a candidate key.\n");
    } else {
        printf("The set of attributes is not a candidate key.\n");
    }

    if (isPrimaryKey(set, closure, dependencies, num_dependencies)) {
        printf("The set of attributes is a primary key.\n");
    } else {
        printf("The set of attributes is not a primary key.\n");
    }

    return 0;
}

```

The screenshot shows a C++ IDE with a file named `main.cpp`. The code implements a program that checks if a set of attributes is a superkey, candidate key, or primary key based on user input. The output window shows the following sequence of events:

```

/tmp/r2qfcdvWd.o
Enter the number of attributes: 3
Enter the set of attributes: 3
Enter the number of functional dependencies: 3
Enter the functional dependencies in the format (determinant -> dependent):
Dependency 1: A -> B
Dependency 2: B -> C
Dependency 3: C -> A
The set of attributes is a superkey.
The set of attributes is not a candidate key.
The set of attributes is not a primary key.

```

4.

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

```
#include <stdlib.h>
```

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

```
#define MAX_ATTR 10
```

```
#define MAX_DEP 10
```

```
struct FunctionalDependency {  
    char determinant[MAX_ATTR];  
    char dependent[MAX_ATTR];  
};
```

```
void computeClosure(char set[], int num_attributes, int num_dependencies, struct  
FunctionalDependency dependencies[], char closure[]) {
```

```
    int changed;
```

```
    // Initialize closure with the given set of attributes
```

```
    strcpy(closure, set);
```

```
    do {
```

```
        changed = 0;
```

```
        for (int i = 0; i < num_dependencies; i++) {
```

```
            int allInClosure = 1;
```

```
            // Check if the determinant is a subset of the closure
```

```
            for (int j = 0; j < strlen(dependencies[i].determinant); j++) {
```

```
                if (strchr(closure, dependencies[i].determinant[j]) == NULL) {
```

```

        allInClosure = 0;
        break;
    }
}

// If the determinant is a subset of the closure, add the dependent to the closure
if (allInClosure) {
    int dependentLength = strlen(dependencies[i].dependent);
    for (int j = 0; j < dependentLength; j++) {
        if (strchr(closure, dependencies[i].dependent[j]) == NULL) {
            closure[strlen(closure)] = dependencies[i].dependent[j];
            changed = 1;
        }
    }
}
} while (changed);
}

int isSuperkey(char attributes[], char closure[]) {
    // Check if closure contains all attributes in the given set
    for (int i = 0; i < strlen(attributes); i++) {
        if (strchr(closure, attributes[i]) == NULL) {
            return 0; // Not a superkey
        }
    }
    return 1; // It is a superkey
}

```

```

int isCandidateKey(char attributes[], char closure[], struct FunctionalDependency
dependencies[], int num_dependencies) {

    // Check if it's a superkey first

    if (!isSuperkey(attributes, closure)) {

        return 0; // Not a candidate key if not a superkey

    }


    // Check if removing any attribute makes it not a superkey
    for (int i = 0; i < strlen(attributes); i++) {

        char modifiedClosure[MAX_ATTR] = "";

        strncpy(modifiedClosure, closure, strlen(closure));

        char withoutAttr[MAX_ATTR] = "";

        strcpy(withoutAttr, attributes);

        memmove(&withoutAttr[i], &withoutAttr[i + 1], strlen(withoutAttr) - i); // Remove one
attribute

        computeClosure(withoutAttr, strlen(withoutAttr), num_dependencies, dependencies,
modifiedClosure);

        if (isSuperkey(withoutAttr, modifiedClosure)) {

            return 0; // If removing an attribute still forms a superkey, not a candidate key

        }

    }

    return 1; // It is a candidate key

}

```

// Function to calculate the factorial of a number

```

int factorial(int n) {

    if (n == 0)

        return 1;

    else

        return n * factorial(n - 1);

}

```

```
}
```

```
// Function to calculate the number of combinations
```

```
int nCr(int n, int r) {  
    return factorial(n) / (factorial(r) * factorial(n - r));  
}
```

```
int main() {  
    int num_attributes, num_dependencies;  
    char set[MAX_ATTR], closure[MAX_ATTR];  
    struct FunctionalDependency dependencies[MAX_DEP];  
    int candidate_keys_count = 0;  
  
    printf("Enter the number of attributes: ");  
    scanf("%d", &num_attributes);  
    if (num_attributes <= 0 || num_attributes > MAX_ATTR) {  
        printf("Invalid number of attributes.\n");  
        return 1;  
    }  
  
    printf("Enter the set of attributes: ");  
    scanf("%s", set);  
  
    printf("Enter the number of functional dependencies: ");  
    scanf("%d", &num_dependencies);  
    if (num_dependencies <= 0 || num_dependencies > MAX_DEP) {  
        printf("Invalid number of dependencies.\n");  
        return 1;  
    }  
}
```

```

printf("Enter the functional dependencies in the format (determinant -> dependent):\n");
for (int i = 0; i < num_dependencies; i++) {
    printf("Dependency %d: ", i + 1);
    scanf("%s -> %s", dependencies[i].determinant, dependencies[i].dependent);
}

// Generate all possible combinations of attributes
for (int i = 1; i <= strlen(set); i++) {
    candidate_keys_count += nCr(strlen(set), i);
}

printf("Number of Candidate Keys: %d\n", candidate_keys_count);

return 0;
}

```

The screenshot shows the Programiz C Online Compiler interface. The code editor on the left contains the C program, and the output window on the right shows the program's execution results.

Code Editor (main.c):

```

78 int factorial(int n) {
79     if (n == 0)
80         return 1;
81     else
82         return n * factorial(n - 1);
83 }
84
85 // Function to calculate the number of combinations
86 int nCr(int n, int r) {
87     return factorial(n) / (factorial(r) * factorial(n - r));
88 }
89
90 int main() {
91     int num_attributes, num_dependencies;
92     char set[MAX_ATTR], closure[MAX_ATTR];
93     struct FunctionalDependency dependencies[MAX_DEP];
94     int candidate_keys_count = 0;
95
96     printf("Enter the number of attributes: ");
97     scanf("%d", &num_attributes);
98     if (num_attributes <= 0 || num_attributes > MAX_ATTR) {
99         printf("Invalid number of attributes.\n");
100        return 1;
101    }
102
103    printf("Enter the set of attributes: ");
104    scanf("%s", set);
105
106    printf("Enter the number of functional dependencies: ");

```

Output Window:

```

http://2qfcdv6w6.o
Enter the number of attributes: 3
Enter the set of attributes: 3
Enter the number of functional dependencies: 3
Enter the functional dependencies in the format (determinant -> dependent):
Dependency 1: A -> B
Dependency 2: B -> C
Dependency 3: C -> A
Number of Candidate Keys: 1

```