THE AMERICAN UNIVERSITY IN CAIRO
الجامعة الأمريكية بالقاهرة

The American University in Cairo

CSCE 2301

Digital Design I

Section 02

K-map C++ Program

Project Report

Instructor: Dr. Mohamed Taher

By: Joseph Boulis - Seifeldin Ashraf
900182870 - 900183864

March 14th, 2020

The K-map Program

This program is implemented using C++ language, and it works using any C++ compiler.

To use this program, do the following:

- If you work with Linux terminal, you have to get GCC compiler installed on your system, or you can use any C++ IDE. Run the code, and enter the logic function you want for it the k-map, by entering the minterms of that function as decimal numbers separated by a comma. Then the program will output the corresponding K-map, and the simplified Boolean expression.

The Program Design

The K-map program is mainly implemented using the C++ language. The design of the program lead to the existence of four main function: Find_Common, add_to_code, fill_code_letters, fill_array_with_grey_code.

The Logic of the program:

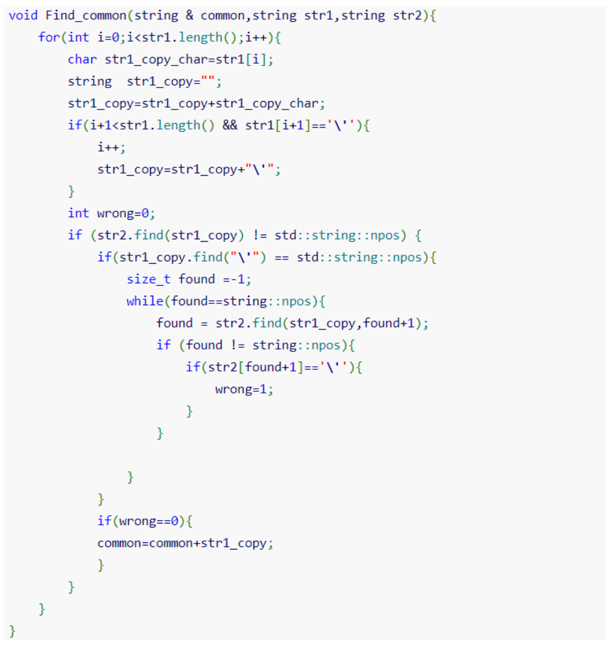
-To Implement a program to produce K-map and Boolean Expression for logic function, we needed to understand how K-map works. K-map needs from you to know the truth table, the Boolean expression, the Minterms, or the Maxterms. In this problem, we used the Minterms as an indicator for the K-map.

-Our program is designed to take the input from the user as a string, which are the Minterms for the Boolean function, then it will parse it, and take the numbers which represent the Minterms, then stores them in an array. Then, we all know that K-map follows the grey code technique, so we created a function called fill_array_with_grey_code, which basically fill two vectors of string, which represents the horizontal and the vertical line of the K-map with grey code like 00,01,11,10.

-We created two 2D arrays, one for producing the original K-map, and another one which works as an indicator for which (1's) have not been grouped yet with other (1's). For producing the K-map from the minterms expression, it was super easy, the program just takes each decimal number from the minterms and convertes it to its binary form. Then, it iterates over the 2D array, when we get to a row number and a column number which its grey code corresponding to a number in the minterms array, we just make this element equals to (1). Then, the K-map and ready for printing.

-As mentioned above, we have in the program two 2D arrays, one for storing the original K-map, and another one for checking ungrouped elements. Here comes the hardest problem in the program. From the top view, we copy the first array elements into the second array, and for each grouping process of any cell contains (1), we mark it as done by making its value to be zero and so on till we finish the whole K-map, and within each checking operation, the add_to_code, fill_code_letters functions do their job. The fill_code_letters function is responsible for generating the Boolean expression which comes from the two vectors mentioned before for grey coding (the horizontal and the vertical one). the add_to_code function just adds to the Boolean expression what the fill_code_letters produces. Basically, we have six plans/cases for generating the Boolean expression.

-We have four main functions in this program for four main functionalities, which are Find_Common, add_to_code, fill_code_letters, fill_array_with_grey_code.

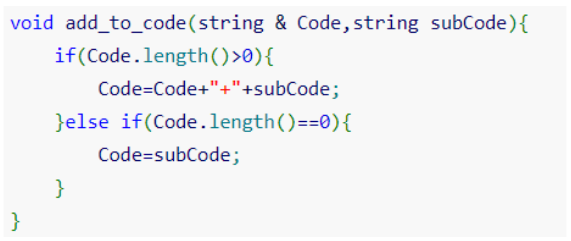


```
void Find_common(string & common,string str1,string str2){
    for(int i=0;i<str1.length();i++){
        char str1_copy_char=str1[i];
        string  str1_copy="";
        str1_copy=str1_copy+str1_copy_char;
        if(i+1<str1.length() && str1[i+1]=='\''){
            i++;
            str1_copy=str1_copy+"\'";
        }
        int wrong=0;
        if (str2.find(str1_copy) != std::string::npos) {
            if(str1_copy.find("\'") == std::string::npos){
                size_t found =-1;
                while(found==string::npos){
                    found = str2.find(str1_copy,found+1);
                    if (found != string::npos){
                        if(str2[found+1]=='\''){
                            wrong=1;
                        }
                    }


                }
            }
            if(wrong==0){
            common=common+str1_copy;
            }
        }
    }
}
```

Find_Common:

It is the function responsible for creating the Boolean expression using the K-map. After grouping a number of ones, we can a term of the Boolean expression by evaluating the case of the group, like if the group is a horizontal one, we care about the common 1's or the common zeros in the horizontal gery code, and also if the group is a vertical one, we care about the grey code in the horizontal part of the map. Basically, this function takes three inputs of type string. The first one is defined by reference as it is the string which the translation of common elements would be stored in.



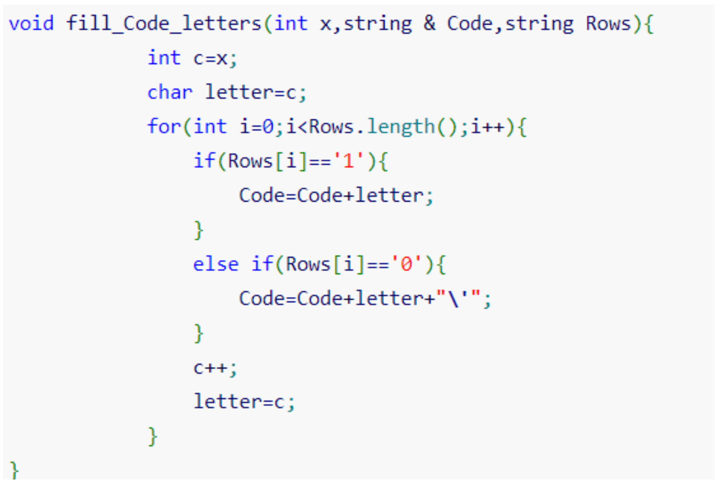
```
void add_to_code(string & Code,string subCode){
    if(Code.length()>0){
        Code=Code+"+"+subCode;
    }else if(Code.length()==0){
        Code=subCode;
    }
}
```

add_to_code:

-This function takes two strings, one to generate the Boolean expression form (A+B+C…), by taking the result of the Find_Common function.

Fill_code_letters:

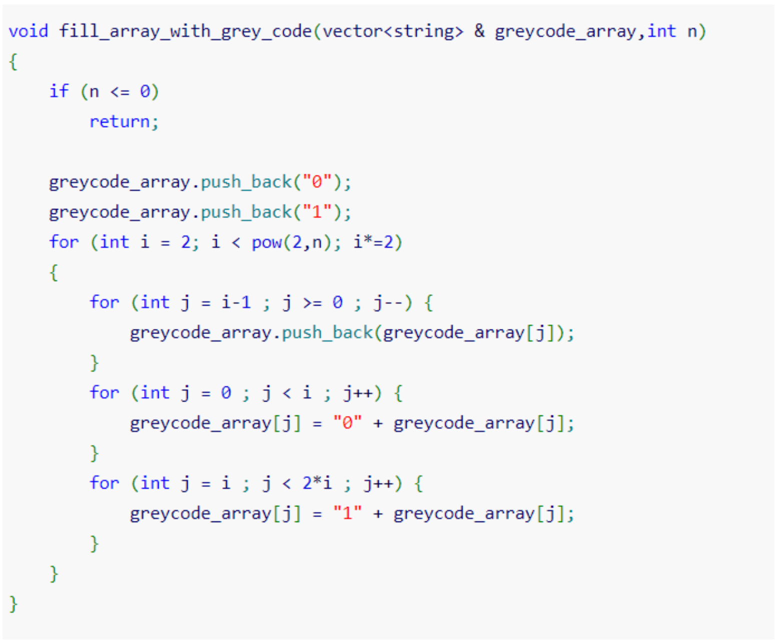
-This function is responsible for generating characters for the Boolean expression depending on the result of the Find_Common function which indicates the zero and the ones, and then this function translates the results into characters.



```
void fill_Code_letters(int x,string & Code,string Rows){
            int c=x;
            char letter=c;
            for(int i=0;i<Rows.length();i++){
                if(Rows[i]=='1'){
                    Code=Code+letter;
                }
                else if(Rows[i]=='0'){
                    Code=Code+letter+"\'";
                }
                c++;
                letter=c;
            }
}
```

Fill_array_with_grey_code:

This function is responsible for creating the grey coding for the K-map like 00,01,11,10. This idea of this function is creating two 2Ds vector. Each one corresponding to a horizontal row in the K-map and for each element in the vector the grey coding will be applied.



```
void fill_array_with_grey_code(vector<string> & greycode_array,int n)
{
    if (n <= 0)
        return;

    greycode_array.push_back("0");
    greycode_array.push_back("1");
    for (int i = 2; i < pow(2,n); i*=2)
    {
        for (int j = i-1 ; j >= 0 ; j--) {
            greycode_array.push_back(greycode_array[j]);
        }
        for (int j = 0 ; j < i ; j++) {
            greycode_array[j] = "0" + greycode_array[j];
        }
        for (int j = i ; j < 2*i ; j++) {
            greycode_array[j] = "1" + greycode_array[j];
        }
    }
}
```

After creating the K-map, then comes the grouping part. To create the Boolean expression, you need to group the ones in the k-map and generate a term of the Boolean expression. To achieve this we created plans for each possible combination of ones, we created **if** conditions, and set a value for each Plan determines the mode of the plan, and if the value is zero, that means that the plan is not working here. Also, we can combine more than one plan. For example, **Plan B, 1** means to group with the left one. **2** means to group with the right element, **3,** to group the element with the left down one, the down, and the right one **Like L shaped. Plan C** has two values, **1** means to group the element with the down cell, and **2** is the opposite. **Plan D** has two values, **1** means group the element to the right or the left and is the reflection of the element like 01, 11 or the corners elements, and **2** is for grouping the reflection element which is up or down the element. **Plan E, 1** means to group with right element, **2** means to group with the right down, **3** mean to group with the right up element. **Plan F,** the element creates a group of itself.