**Софийски универитет „Св. Климент Охидски“**

*Факултет по математика и информатика*

*Специалност: “Информационни системи ”*

*Курс: 3, Група: 1*

*Дисциплина: “СОЗ”*

***ДОМАШНА РАБОТА №1***

***Изготвил:***

Сава Станимиров Димитров (ФН:)

София

зимен семестър 2018/2019

**Условие на задачата.**

3𝑑адачата

Програмата трябва да установи дали задачата има решение и да намери всички нейни решения, ако има такива. Накрая програмата трябва да извежда на конзолата всички намерени решения или индикация за липса на решение.

В термините на задача за удовлетворяване на ограничения (Constraint

Satisfaction Problem, CSP) съм формализирал условието по следния начин:

• Променливи: A, B, C, D, E, F, G, H, I;

• Област на допустимите стойности: {1,2,3,4,5,6,7,8,9} за всяка от

променливите;

• Ограничения:

o Alldiff (A, B, C, D, E, F, G, H, I),

**Описание на метод за решаване на задачата.**

Алгоритъмът, който избрах да използвам е чисто за удовлетвоняване на ограничения. Целият алгоритъм се описва от класа Cryptarithmetic, който ще опиша надолу.

В самия клас съм създал променливите A, B, C, D, E, F, G, H, I и също така съм създал по една променлива за всеки отделен ред от умножението и събирането както следва: baseline1(описва ABC), baseline3(описва IAG0), baseline4(описва EHFA00), baseline5(описва крайния резултат EDBDFC). Не съм създавал baseline2(DEB), тъй като умножението с него правя не като цяло 3-цифрено число, а като отделни цифри.

Методите, които ползвам са:

* swap() –разменя местата на две цифри в масива, който подаваме като аргумент на основната функция в класа. Използваме метода, за да преминем през всички възможни пермутации на цифрите 1-9.
* changeVars() –променя стойностите на променливите A, B, C, D, E, F, G, H, I за всяка възможна пермутация.
* allDiff() – проверява дали всички променливите са с различни стойности
* buildBaselines() – създава всеки отделен ред от условието на задачата
* isCorrectSolution() – проверява дали подадените стойности на променливите удовлетворяват равенството в условието
* printSolution() –изписва всяко правилно решение
* cryptarithmeticProblem() –основният метод в класа и ползва всички методи описани по-горе

В класа са описани два от начините за решаване на поставения проблем. Единият е така нареченият „Brute-force“ алгоритъм, а другият - „Backtracking“ алгоритъм(т.е. рекурсивно). Двата алгоритъма използват едни и същи помощни методи, които са описани по-горе. „Brute-force“ алгоритъмът не приема никакви аргументи в основната функция и при него променливата B има стойност по подразбиране, която е B = 1.

При „Backtracking“ алгоритъма основната функция приема 3 аргумента: масив от числа, начало(= 0) и край(= 9) (имаме 9 променливи, върти го до < края). Тук алгоритъмът проверява дали началото и края са равни, ако са равни, то тогава променя променливите ни със съответните цифри( changeVars() ), проверява дали са различни(allDiff()), ако са различни, сглобява всеки baseline(buildBaselines()), проверява дали удовлетворяват условието(isCorrectSolution()) и ако го удовлетворяват ни печата решението на конзолата(printSolution()), ако не, то продължава да търси решение измежду останалите пермутации.

В „Brute-force“ алгоритъма, идеята е подобна, но по-бавна и не използва методите swap() и changeVars(), тъй като променливитепроменям директно в самите for() цикли, но принципът е същият, отново обикаляме всички възможни пермутации и правим съответните проверки.

В програмата използвам „Backtracking“ алгоритъма като основен. „Brute-force“ алгоритъмът може да се пусне когато се разкоментират съответните парчета код и се закоментират съответните за „Backtracking“ алгоритъма.

В двата алгоритъма съм покрил случая, ако няма решение.

**Описание на реализацията с псевдокод.**

**„Backtracking“ алгоритъм:**

**function** CryptarithmeticProblem (array of ints, start of array, end of array)

**returns** the correct solution

**initialize** i with 0

**if** start of array **equal to** end of array **then**

change value of variables **<-** ChangeVars(array of ints)

**if** AllVariablesDifferent() **then**

create baselines **<-** BuildBaselines()

**if** SolutionIsCorrect() **then**

print us the solution **<-** printSolution()

**else**

**for** i = start of array **to** (end of array – 1) **do**

swap two elements **<-** swap(array of ints[0] + start of array,

array of ints[0] + i)

recursively call the function **<-**

cryptarithmeticProblem(array of ints, start of array + 1, end of array)

swap two elements **<-** swap(array of ints[0] + start of array,

array of ints[0] + i)

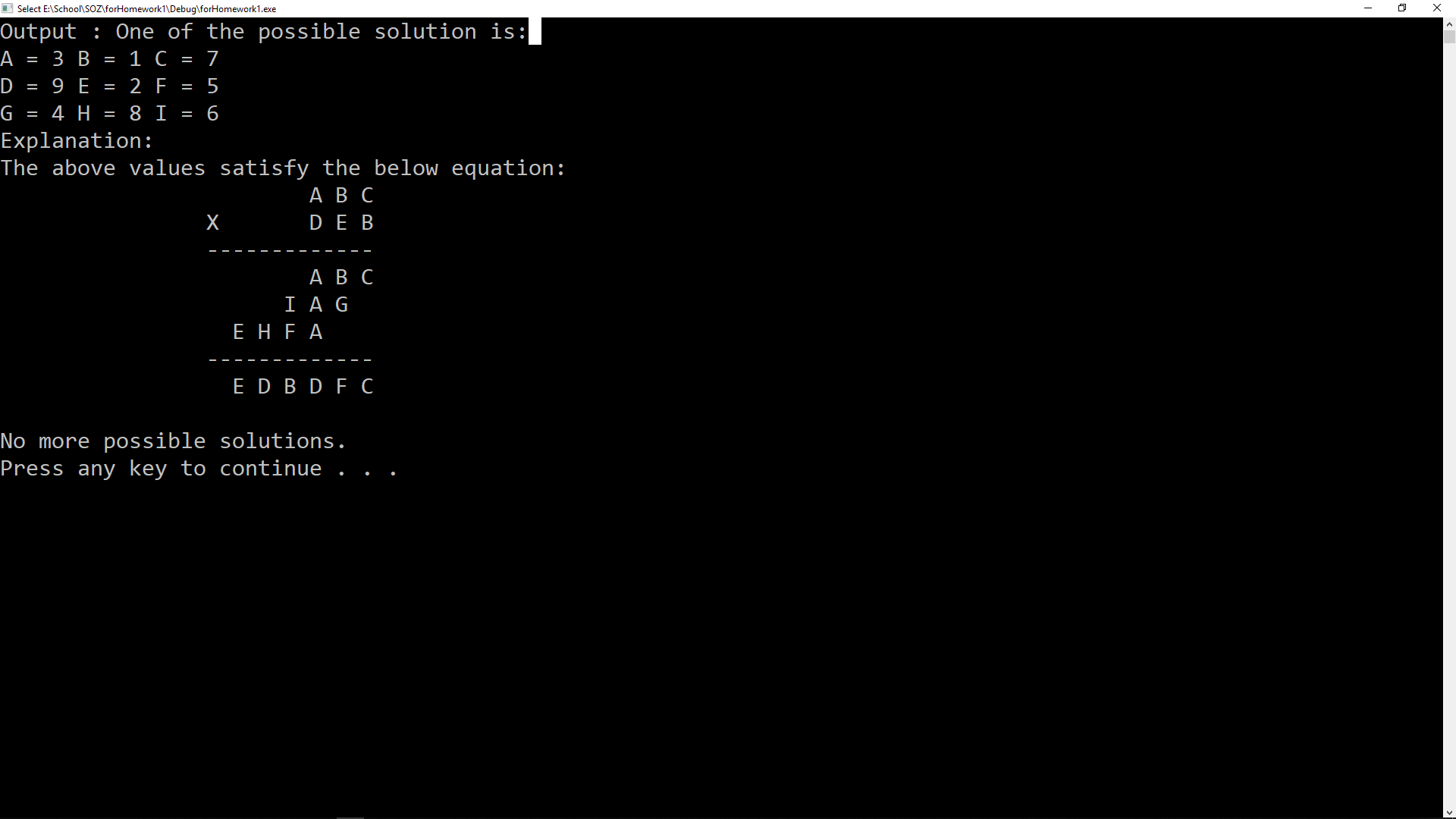
**Инструкции за компилиране на програмата:**

Програмата е писана на Visual Studio 2015, а езикът е C++. Няма допълнително добавени библиотеки за ползване от програмата, използва стандартните на C++.

**Примерни резултати.**

Двата алгоритъма дадоха един и същ резултат, което предполага, че програмата работи коректно.

Това е самият резултат:



Конкретно за този резултат използвах „Backtracking“ алгоритъма, като както се вижда A = 3, B = 1, C = 7, D = 9, E = 2, F = 5, G = 4, H = 8, I = 6. Самите алгоритми си казват, че няма други резултати, тъй като минават през всички възможни комбинации на цифрите [1..9].

Проверката за коректност се изпълнява по време на изпълнението на самата програма, а проверката се осъществява с метода: isCorrectSolution().

Също така, ако няма решение то програмата ще ни го каже. Това се вижда на долната картинка:

