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

*Факултет по математика и информатика*

*Специалност: “Информационни системи ”*

*Курс: 3, Група: 1*

*Дисциплина: “СОЗ”*

***ДОМАШНА РАБОТА №4***

***Изготвил:***

Сава Станимиров Димитров (ФН:)

София

зимен семестър 2018/2019

Съдържание

[**1.** **Описание на използвания метод** 3](#_Toc535350553)

[**1.1** **Структура на програмата** 3](#_Toc535350554)

[**1.2** **Описание на самия алгоритъм k-NN** 3](#_Toc535350555)

[**2.** **Описание на реализацията с псевдокод** 4](#_Toc535350556)

[**2.1** **Псевдокод** 4](#_Toc535350557)

[**3.** **Инструкции за компилиране на програмата** 5](#_Toc535350558)

[**4.** **Примерни резултати** 5](#_Toc535350559)

[**4.1** **Примери** 5](#_Toc535350560)

# **Описание на използвания метод**

## **Структура на програмата**

Програмата се състой от:

* Person – структура за тренировъчните данни, по които ще определяме нов Person какви продукти ще купува. Като неговите данни са:
  + Spendings – колко е изхарчил
  + Frequency – колко често е харчил
  + discountOrNo – дали е купувал с намаление или не
  + distance – стойност за разстоянието между всеки човек
* Classification – клас в който са описани всички нужни функции за метода k-NN
  + readFile(char\* fileName) – чете данните от файла с тренировъчните данни и ги записва в масив от хора(Person people[])
  + sort() – сортира данните в масива по разстояние (distance)
  + classifyAPerson(Person[], arraySize, k, Person) – самия метод за класификация на точка чрез k-NN алгоритъма
  + classification(k) – функция, която връща резултата на classifyAPerson()
* main() – създава обект(Classification); записва тренировъчните данни в него; потребителя въвежда k; извежда резултата от класификацията
  1. **Описание на самия алгоритъм k-NN**

Функцията, която изпълнява алгоритъма е classifyAPerson(). Тя приема като аргументи:

* масив от хора(Person people[]) - запълнен с тренировъчните данни от .csv файла
* големина на масива от хора(n) – показва колко е голям масива
* стойност за k - най-близките съседи
* нов човек – човека който искаме да класифицираме

В началото на алгоритъма изчисляваме разстоянията на тренировъчните данни спрямо новите данни(новия човек). Като това става чрез изчисление на Евклидовото разстояние. След което сортираме масива във възходящ ред по полученото разстояние, за да може най-близките съседи на новия човек да бъдат в началото. Сортиращият алгоритъм, който използвам е bubble sort, тъй като данните са малко и ще бъдат сортирани достатъчно бързо. Ако данните ни бяха повече бих използвал merge sort, тъй като работи доста бързо за големи масиви от данни.

След това създаваме две променливи за покупките с намаление(1) и без намаление(0), които в началото са равни на 0. Когато програмата е стигнала до този момент, вече можем да проверим k най-близките съседи на новия човек. Това става чрез for цикъл до k-тия елемент на масива, като когато i-я по ред елемент в масива е правил покупки без намаления увеличаваме променливата за без намаление(0), а когато е правил с намаление – променливата за с намаление(1). Накрая проверяваме коя от двете променливи е по-голяма и извеждаме съответния резултат – (0) -> 0 ; (1) -> 1.

# **Описание на реализацията с псевдокод**

## **Псевдокод**

**function** classifyAPerson(array of people, size of array(n), number for k, new person)

**returns** 0 **or** 1

**for** **initialize** i **with** 0 **to** n **do**

**initialize** array of people[ i ].distance **with**

**square root of** ((array of people[ i ]**.**spendings – new person**.**spendings) \* (array of people[ i ]**.**spendings – new person**.**spendings) + (array of people[ i ]**.**frequency – new person**.**frequency) \*

(array of people[ i ]**.**frequency – new person**.**frequency))

**call function** **->** sort

// group 0

**initialize** freq0 **with** 0

// group 1

**initialize** freq1 **with** 0

**for** **initialize** i **with** 0 **to** k **do**

**if** array of people[ i ]**.**discountOrNo **equal to** 0

**increase** freq0 **with** 1

**else if** array of people[ i ]. discountOrNo **equal to** 1

**increase** freq1 **with** 1

# **Инструкции за компилиране на програмата**

Програмата е написана на C++, затова за компилацията е нужно IDE с компилатор за C++. Не са използвани допълнителни странични библиотеки, освен вградените.

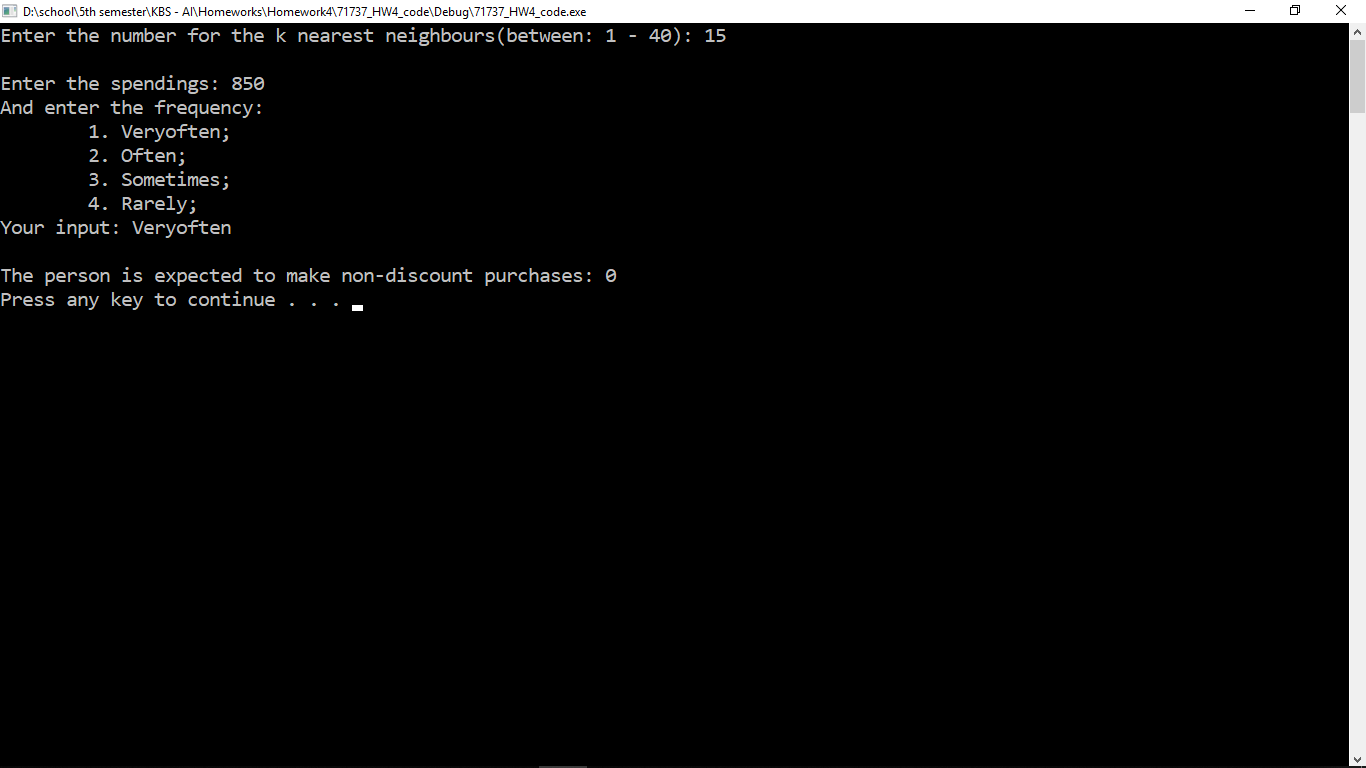
Освен C++ компилатор, файлът с тренировъчните данни е нужно да бъде в директорията на .cpp файла. Ако не е в същата директория, то тогава променливата „fileName“, в main функцията, трябва да има стойност равна на целия път до файла с тренировъчните данни.

Прекачвам и файла(.csv) с тренировъчните данни, тъй като освен да го запазя като .csv файл съм го допроменил, за да може програмата да го чете правилно.

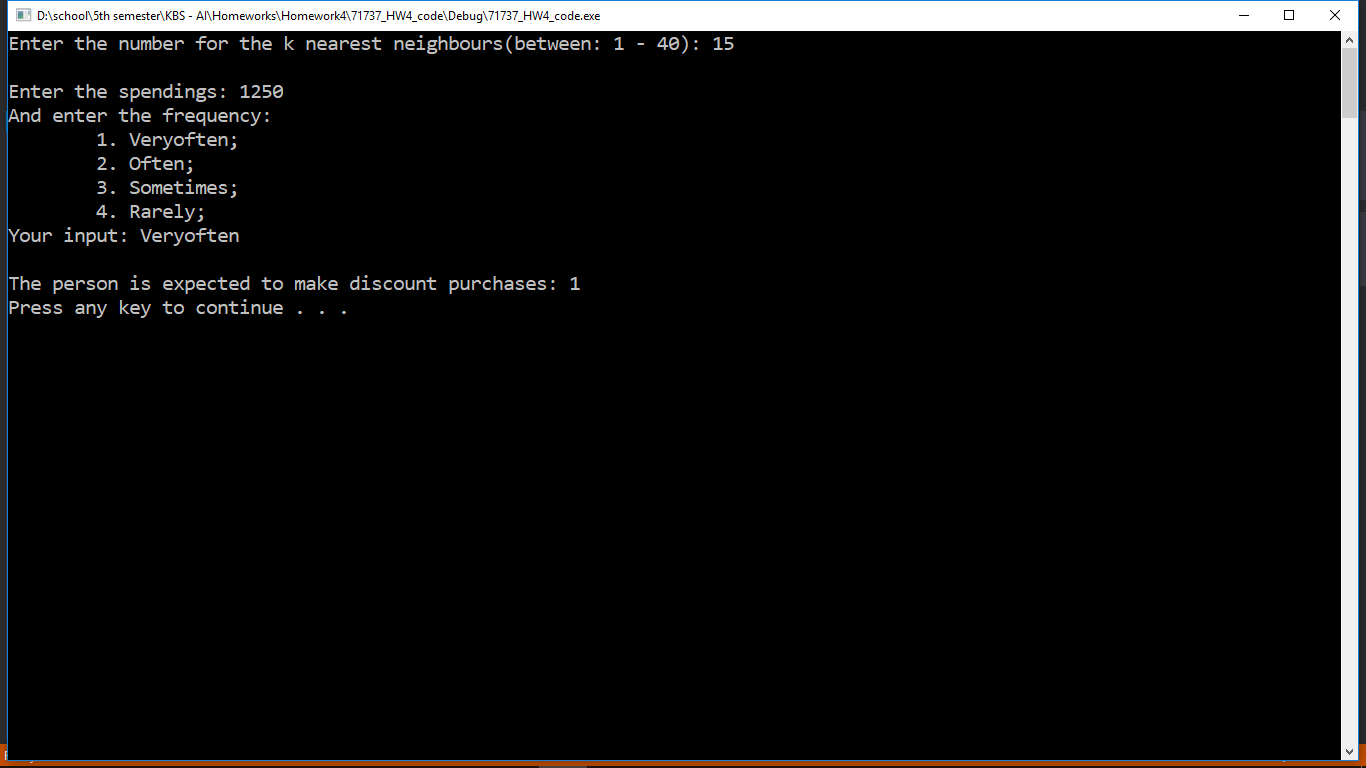
# **Примерни резултати**

## **Примери**

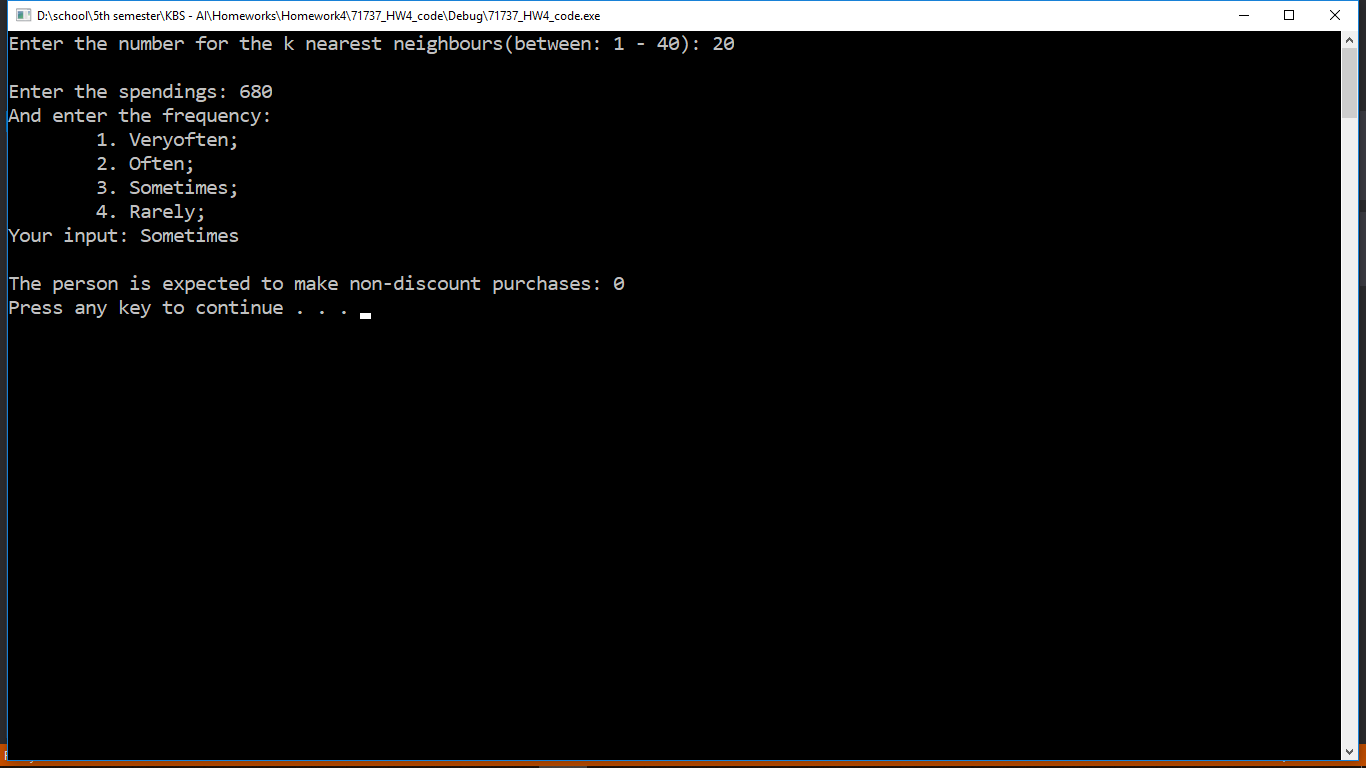
Пример 1:



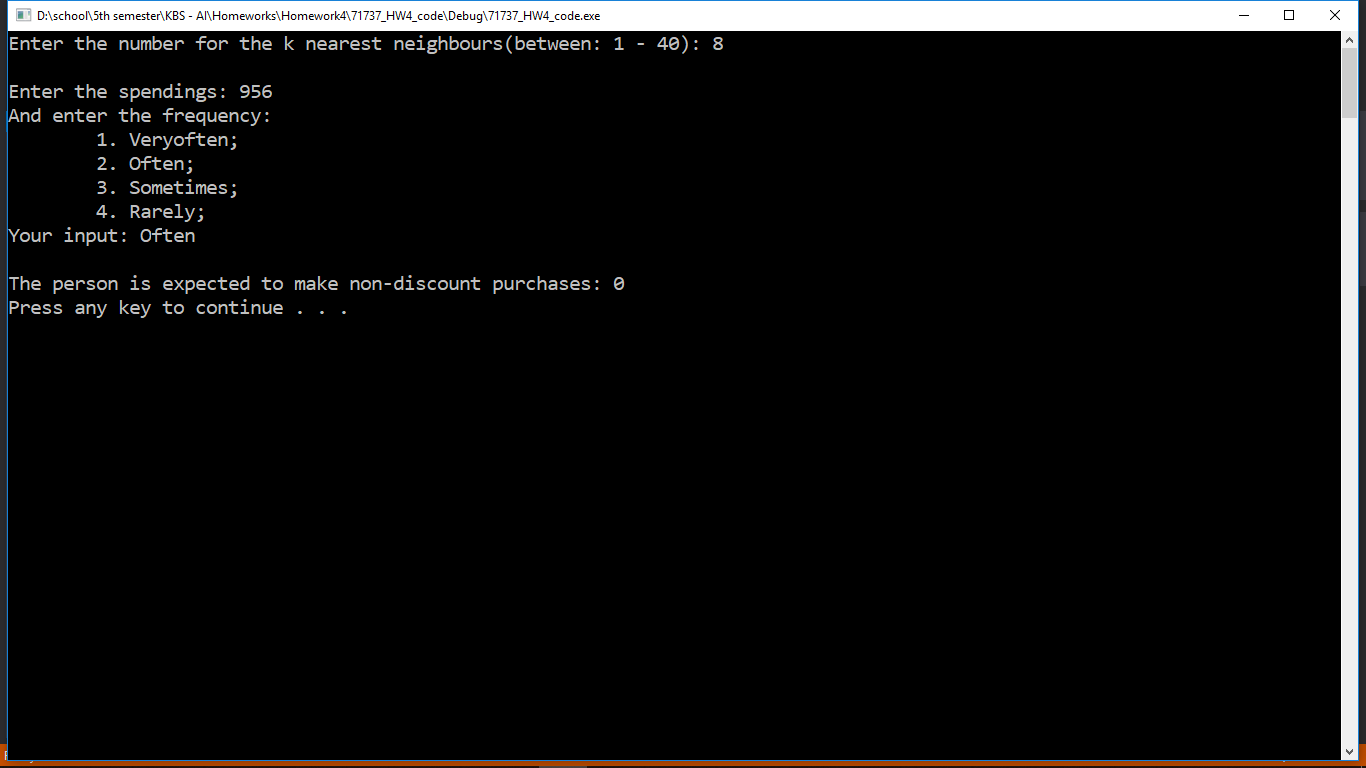
Пример 2:



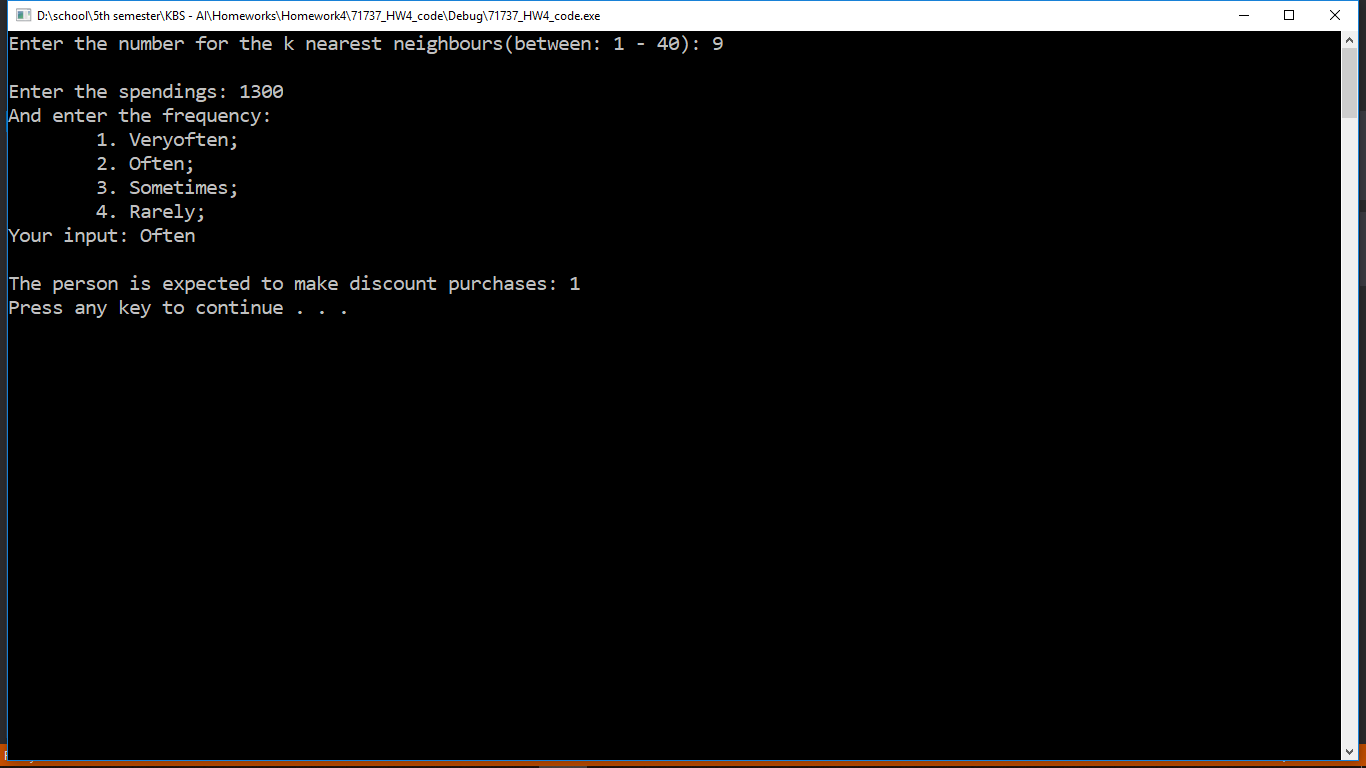
Пример 3:



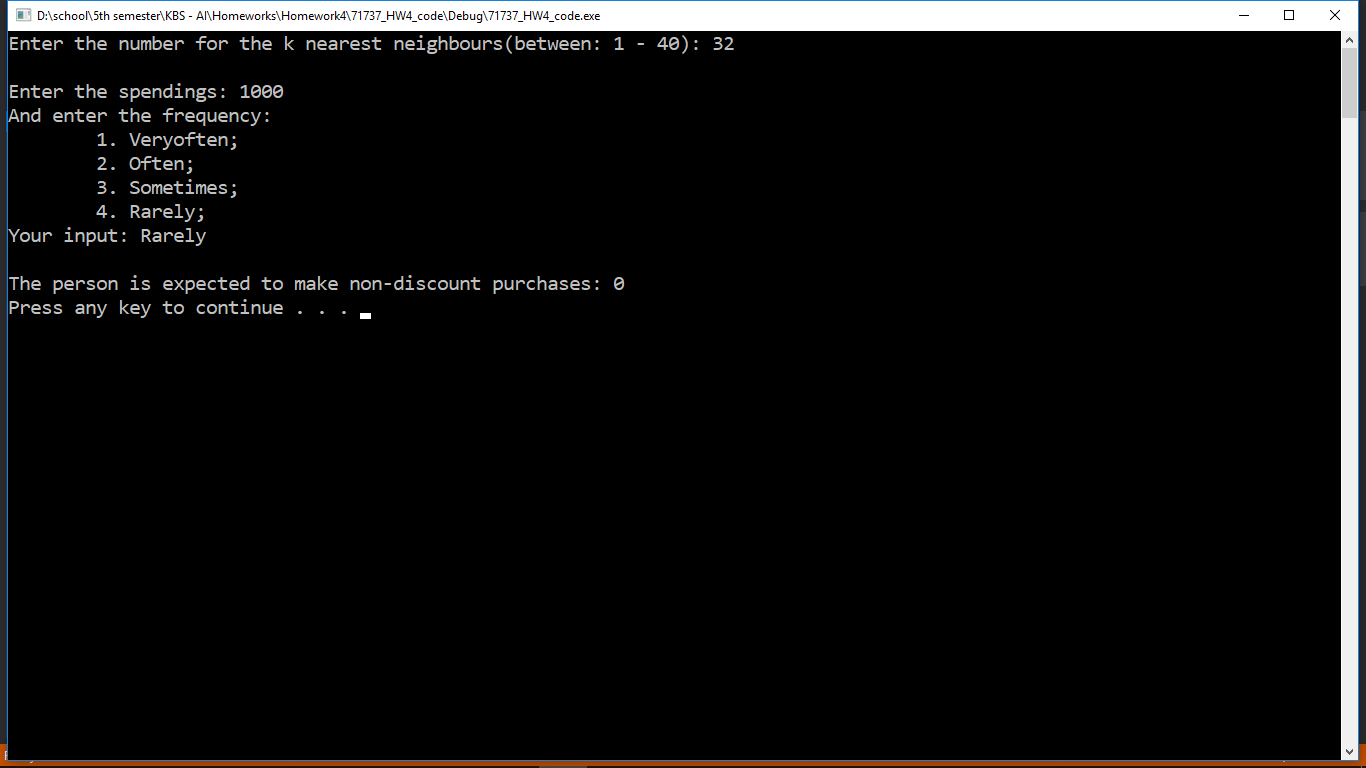
Пример 4:



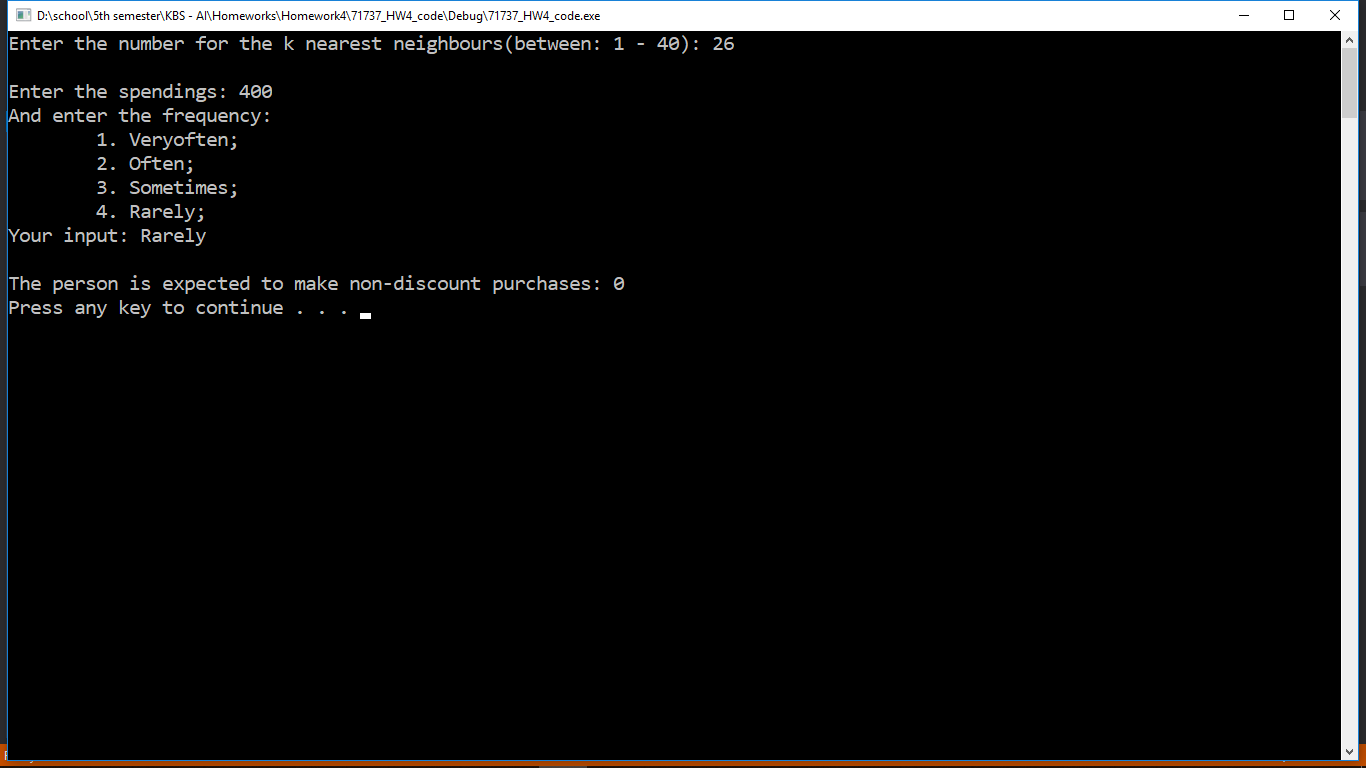
Пример 5:



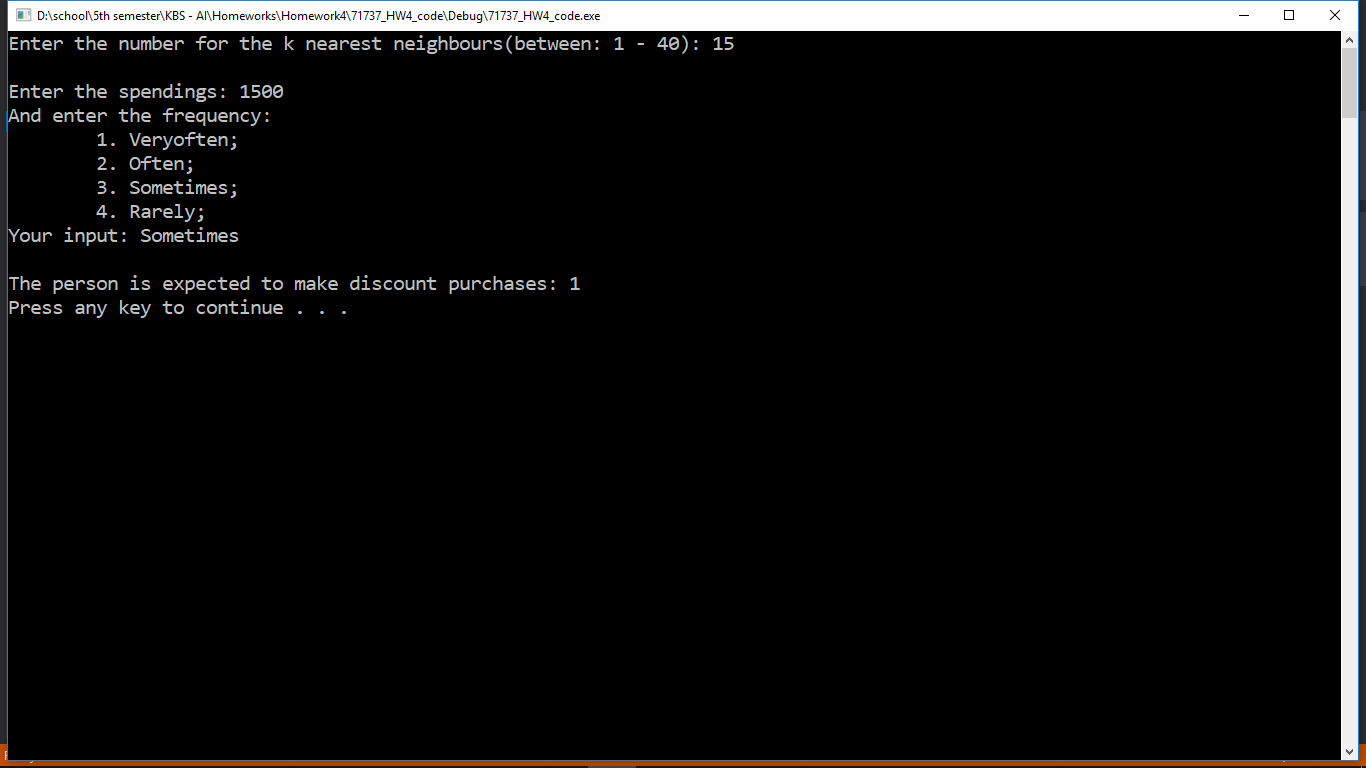
Пример 6:



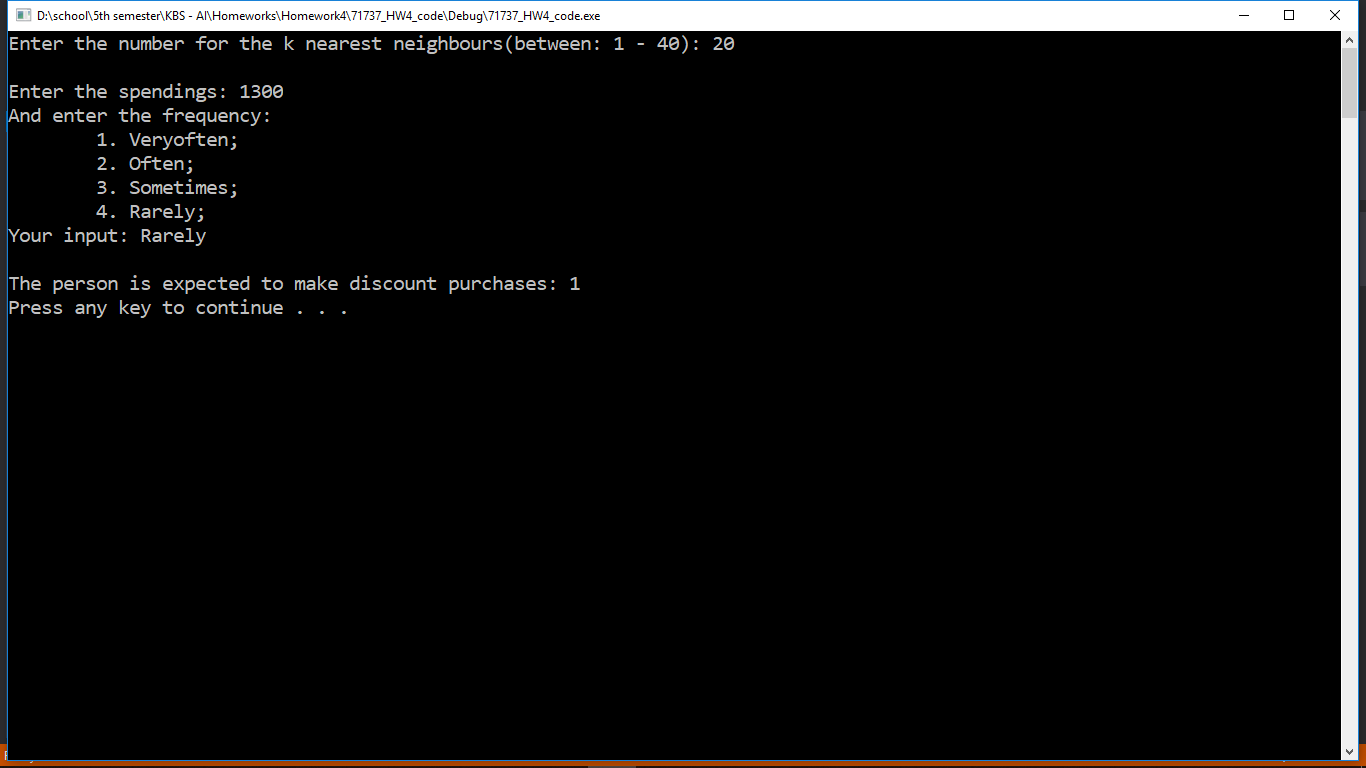
Пример 7:



Пример 8:



Пример 9:



Пример 10:

