Ziele sind das Arbeiten mit Reihentyp, (Daten-)Strukturen , und Sortierverfahren

Problem:
Die Verwaltung von Kundendaten (Stammdaten) ist eine dauerhafte und lästige Aufgabe in nahezu allen Bereichen.

Die Aufgabe:
Zur Unterstützung der Sachbearbeiter soll ein Programm geschrieben werden, welches eine minimale Kundenverwaltung ermöglicht. Folgende Aufgaben sollen dadurch gelöst werden:

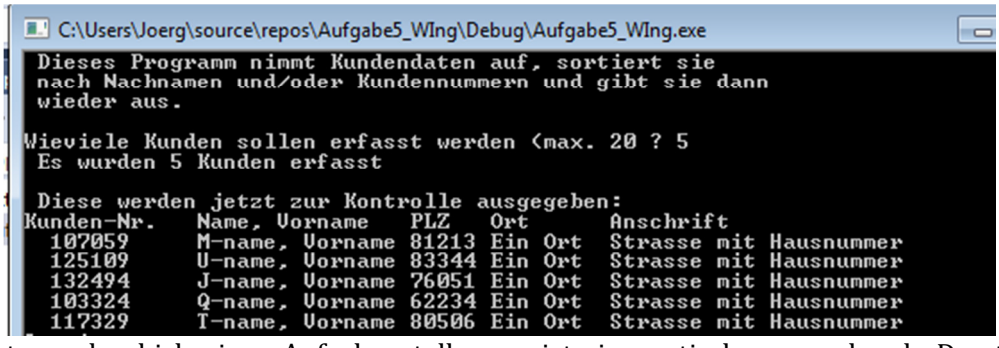
a) Erfassen der Kundendaten über Tastatur
b) Einlesen der Kundendaten aus einer Datei (optional)
c) Ausgabe der Kundendaten in Listenform auf dem Bildschirm
d) Schreiben der Kundendaten in eine Datei (optional)
e) Sortieren der Kundendaten nach Nachname
f) Sortieren der Kundendaten nach Kundenummer
g) Programm beenden

Die Kundendaten setzen sich aus der Kundenummer, dem Namen und Vorname, sowie der Adresse (bestehend aus Straße, Postleitzahl und Ort) zusammen. Die genaue Struktur ist in der Header-Datei beschrieben (Sie dürfen weitere Felder nach Belieben hinzufügen, aber keine entfernen!)

Programmaufbau:
Realisieren Sie die einzelnen Teilaufgaben durch ein Menüsystem, die einzelnen Teilaufgaben werden jeweils als eigene Funktion (Unterprogramm) programmiert. Die Header-Datei kann als Beispiel verwendet werden. Änderungen in den Schnittstellen und den Funktionen sind erlaubt.

Zur Vereinfachung des Programmtests ist es erlaubt, zu Beginn die Eingabe der Kundendaten durch eine entsprechende Initialisierung der Strukturen vorzunehmen, um die Zeit für das ständige Eingeben von Testwerten zu sparen. **Am Ende der Übung muss aber die Eingabe per Tastatur funktionieren.**

Die Ausgabe der Kundendaten soll in Listenform erfolgen, dazu wird eine Überschrift ausgegeben und darunter die entsprechenden Daten. Das könnte z.B. so aussehen:



Im Gegensatz zu den bisherigen Aufgabenstellungen ist eine optisch ansprechende Darstellung also gefordert! Dabei helfen die Bibliothek <*iomanip*> und die entsprechenden Seiten (S. 88ff) im „Informatik für Nicht-Informatiker“.

Beginnen Sie also wie üblich mit dem Hauptprogramm und der Menü-Funktion, danach schreiben Sie die Funktion für die Eingabe (oder nutzen zunächst die Möglichkeit zur Initialisierung).

Danach benötigen Sie die Funktion zur Ausgabe der Kundendaten.

Danach steht die wichtigste Teil-Funktion an, das Sortieren. Diese Funktion soll mittels Selection-Sort (siehe unten) die eingegebenen Kundendaten nach Nachname sortieren, danach ist eine weitere Funktion zum Sortieren nach Kundennummern gefordert.

Selection-Sort
Dieses Sortierverfahren funktioniert ähnlich wie das bekannte Bubble-Sort, benötigt aber deutlich weniger Vertauschungen und ist damit etwas schneller. Zunächst wird das gesamte Feld nach dem Maximum durchsucht, dann wird das gefundene Maximum mit dem letzten Element getauscht.

Im nächsten Durchgang wird das Maximum der verbliebenen Elemente gesucht und mit dem vorletzten Platz getauscht, dies wiederholt sich solange, bis das Feld sortiert ist.

Hier das Sortieren anhand von ganzen Zahlen
Das unsortierte Feld sieht so aus:

17	22	1	8	45	13	4

Das Maximum steht an Index 4 (!) und hat den Wert 45 (gelb hinterlegt) . Dies wird jetzt mit dem letzten Element (an Position 6) getauscht:

17	22	1	8	4	13	45

Damit ist das letzte Element sortiert, es müssen nur noch die Elemente 0 bis 5 sortiert werden. Das neue Maximum liegt jetzt bei Index 1 und hat den Wert 22. Dieses Element wird mit dem letzten, noch nicht sortierten Element (also an Index 5) getauscht:

17	13	1	8	4	22	45

Jetzt sind bereits zwei Elemente am richtigen Platz, das neue Maximum ist diesmal an Index 0 und hat den Wer t 17. Es tauscht den Platz mit dem Element an Index 5 (der 4).

4	13	1	8	17	22	45

Neues Maximum an Index 1 mit 13, tauscht mit Index 3:

4	8	1	13	17	22	45

4	1	8	13	17	22	45

Im letzten Schritt tauschen dann noch die beiden führenden Elemente den Platz und das Feld ist fertig sortiert:

1	4	8	13	17	22	45

Technisch wird auch dieses Sortieren durch zwei geschachtelte for-Schleifen gelöst, die äußere zählt rückwärts, die innere bis zum aktuellen Wert der äußeren Schleife.

Tipp: Beachten Sie den Unterschied zwischen Position des Maximums und dem Wert des Maximums!

Die Headerdatei:

Die unten angegebene Header-Datei kann/muss erweitert warden (z.B. fehlt die *menue()* Funktion)

```
#pragma once
// aufgabe5_SS2019.h
// Header-Datei zu Aufgabe 5 – Sommersemester 2019
#include <iostream>
#include <iomanip>
#include <cmath>
#include <string>
#include <fstream>     // erlaubt Dateiverarbeitung
#include <stdlib.h>    // benötigt für atoi(), atof()
using namespace std;

/************************************
***  Konstanten
************************************/
// Max. Groesse des Arrays fuer die Kundenkartei
const int MAXKUNDEN = 20;

/*****************************************************
***  Definition der Datenstrukturen
***   Diese Strukturen duerfen ergaenzt/erweitert,
***   aber nicht gekuerzt werden!!
*****************************************************/
struct sadresse
{
  string strasse;
  unsigned int plz;
  string ort;
};

struct skunde
{
  string name;
  string vorname;
  sadresse anschrift;
  unsigned long kdnummer;   // 6-stellig, bei der Eingabe zu ueberpruefen!!
};

/* Die Funktion kunden_erfassen(…) liest die gewuenschte Anzahl von Kunden
     über die Tastatur ein
*/
void kunden_erfassen( skunde kartei[], int & anzahl, const int MAXKUNDEN );

/* Die Funktion kunden_ausgeben(…) gibt die Daten eines ‚expliziten‘ Kunden
     Formatiert auf dem Bildschirm aus
*/
void kunden_ausgeben( skunde kartei[], int nr );

/* Die Funktion selection_sort_name(…) sortiert die Datensaetze mittels
     Selection-Sort nach Nachname der Kunden
*/
void selection_sort_name( skunde kartei[], int anzahl );

/* Die Funktion selection_sort_kdnr(…) sortiert die Datensaetze mittels
     Selection-Sort nach Kundennummer
*/
void selection_sort_kdnr( skunde kartei[], int anzahl );
```

```
/* Die optionale Funktion werte_aus_datei_lesen(…) liest die Kundenliste
     aus einer Datei ein.
     Der Rueckgabewert gibt an, ob die Datei erfolgreich (= true)
     verarbeitet werden konnte, ansonsten false
*/
bool werte_aus_datei_lesen(skunde kartei[], int & anzahl,
                           string dateiname, const int MAXKUNDEN );

/* Die optionale Funktion werte_in_datei_schreiben(…) schreibt die Kundenliste
     in eine Datei
     Der Rueckgabewert gibt an, ob die Datei erfolgreich (= true)
     verarbeitet werden konnte, ansonsten false
*/
bool werte_in_datei_schreiben(skunde kartei[], int anzahl, string dateiname );
```

Für Spezialisten:
Man kann die beiden Sortierfunktionen in einer Funktion schreiben und über einen (zusätzlichen) Parameter steuern, nach welchem Attribut sortiert werden soll

Hinweise zur Dateiverarbeitung:
Beispielhaft Beginn der Funktion `werte_aus_datei_lesen()`

```
bool werte_aus_datei_lesen( skunde kartei[], int & anzahl,
                            string dateiname, const int MAXKUNDEN )
{
  // Variablen
  bool retWert = false;
  string zeile;
  ifstream eingabeDatei;       // ifstream:  Datei kann nur zum Lesen geöffnet werden
                               // ofstream:  Datei kann nur zum Schreiben geöffnet werden

  // Datei zum Lesen öffnen
  eingabeDatei.open( dateiname, ios::in );
  if ( true == eingabeDatei.is_open() )
  {
    // solange Dateiende nicht erreicht && noch Platz in der Kartei
    while ( ( false == eingabeDatei.eof()) && ( anzahl < MAXKUNDEN ) )
    {
      // Zeile einlesen für Nachname
      getline( eingabeDatei, zeile, '\n' );
      if ( 0 != zeile.length() )
      {
        // Zeile zuweisen
        kartei[anzahl].name = zeile;
        . . .

        // Zeile einlesen für Kundennummer
        getline( eingabeDatei, zeile, '\n' );
        // Zeile einem Integerwert zuweisen
        kartei[anzahl].kdnummer = atoi(zeile.c_str());
        . . .

        // Zeile einlesen für "*****"
        // = Datensatzende
        . . .
```