Vergleichsprogramm Sprachen / Paradigmen

Gemeinsame Entwicklung verschiedener Programme, die ein komplexeres und aussagekräftiges Problem lösen. Die Programme werden in unterschiedlichen Sprachen mit verschiedenen Entwicklungssystemen und Paradigmen implementiert.

Ziel ist es nicht die „beste“ Sprache zu finden, sondern eine Vergleichbarkeit zu haben, Vor- und Nachteile von Sprachen und Paradigmen herauszuarbeiten und einen Qualifikationsbedarf der Nutzergruppen in den jeweiligen Sprachen aufzuzeigen.

# Inhaltsverzeichnis

[Inhaltsverzeichnis 2](#_Toc89186047)

[Das Vergleichsprogramm 4](#_Toc89186048)

[Anforderungen 6](#_Toc89186049)

[Kataster- Daten 7](#_Toc89186050)

[Aufbau der Datei 7](#_Toc89186051)

[Vorverarbeitungen der Kataster- Daten 8](#_Toc89186052)

[Einlesen der Daten aus der sequenziellen Eingabedatei 8](#_Toc89186053)

[Berechnen der Entfernung und des Kurswinkel vom Berechnungsziel zum jeweiligen Haus 9](#_Toc89186054)

[Sortierung der Grunddaten 11](#_Toc89186055)

[Schreiben in eine sequenzielle Datei 11](#_Toc89186056)

[Arbeiten mit der Verzeichnisstruktur 12](#_Toc89186057)

[Schreiben in eine Verzeichnisstruktur 12](#_Toc89186058)

[Löschen der Verzeichnisstruktur 14](#_Toc89186059)

[Freigeben des dynamischen Speichers mit den Kataster- Daten 14](#_Toc89186060)

[Einlesen der Kataster- Daten aus der Verzeichnisstruktur 15](#_Toc89186061)

[Nachverarbeitung der Daten 15](#_Toc89186062)

[Partitionierung von Datensätze 15](#_Toc89186063)

[Neusortierung mit Entfernung und Winkel 15](#_Toc89186064)

[Berechnungen mit Koordinaten 17](#_Toc89186065)

[Beschreibung der verwendeten Maßzahlen 17](#_Toc89186066)

[Minimum und Maximum 17](#_Toc89186067)

[Mittelwert 17](#_Toc89186068)

[Arithmetischer Mittelwert 18](#_Toc89186069)

[Median 18](#_Toc89186070)

[Berechnungen 18](#_Toc89186071)

[Datei mit interessanten Spots 20](#_Toc89186072)

[Aufbau der Spots- Datei 20](#_Toc89186073)

[Vorverarbeitung 21](#_Toc89186074)

[Einlesen der Datei 21](#_Toc89186075)

[Sortieren der Datei 21](#_Toc89186076)

[Schreiben der Datei 21](#_Toc89186077)

[Komplexe Berechnungen 22](#_Toc89186078)

[Bestimme Adresse mit Stadtbezirk / Stadtteil zu jedem Spot 22](#_Toc89186079)

[Ausgabe 23](#_Toc89186080)

# Das Vergleichsprogramm

Da immer wieder Vergleiche von Programmiersprachen und Paradigmen auftauchen, und diese am Ende daran „Scheitern“, dass es keine vergleichbaren Programme gibt, stattdessen nur einzelne (Mikro-) Algorithmen oder allgemeine Betrachtungen, habe ich vor einigen Tagen die Idee dieses Programms entwickelt, dass in einer überschaubaren Zeit mit unterschiedlichen Programmier­sprachen implementiert werden kann. Mir geht es dabei darum, eine Vergleichbarkeit von Pro­grammier­sprachen, von Entwicklungssystemen innerhalb einer Sprache, der Qualität von Algo­rithmen und verschiedenen Programmparadigmen zu ermöglichen. Deshalb möchte ich ein verteiltes Projekt initiieren, an dem sich Entwicklerinnen und Entwickler mit unterschiedlichen Programmier­sprachen, Kenntnisstand und Entwicklungswerkzeugen beteiligen können. Dabei geht es vorder­gründig nicht um einen Vergleich der Programmiersprachen, wir können hier gemeinsam Vor- und Nachteile herausarbeiten, unterschiedliche Algorithmen und Paradigmen bewerten, aber auch Nach­holbedarf bei der Qualifikation in verschiedenen Nutzergruppen sichtbar machen.

Wichtig war bei der Überlegung zu diesem Programm auch, dass die Aufgabe in einer überschaubaren Zeit umsetzbar sein sollte, da ansonsten kaum mit einer Beteiligung gerechnet werden kann.

Trotzdem muss die Anwendung komplex genug sein, um wirklich eine Vergleichbarkeit zu bieten, es müssen verschiedene Anforderungen kombiniert werden und auf das Programm sollte auf einem größeren Datenbestand arbeiten. Wobei sich sicher, wie bei jedem Beispiel, auch hier über Sinn und Unsinn diskutieren lässt. Ich habe auch versucht auf Technologien zu verzichten, die eher von zentralen Systemen und in anderen Sprachen (meisten in Assembler oder C/C++) geschriebenen Treibern abhängen.

Deshalb geht um in diesem Projekt um die Fähigkeiten in den verschiedenen Programmiersprachen Daten aus Dateien sequenziell einzulesen und in dynamischen Speicherstrukturen für Bearbeitungen bereitzustellen, zu sortieren, komplexere Berechnungen durchzuführen, Daten auszuwerten und auch wieder in sequenzielle Dateien zu schreiben. Parallel soll in einem zweiten Bereich auch eine umfassende Verzeichnisstruktur erzeugt und wieder eingelesen werden, um die Unterschiede bei der Arbeit mit Verzeichnisstrukturen mit sehr vielen verschiedenen Dateien und Dateioperationen zu zeigen.

Jeder Teilnehmende sollte das Projekt mit seinem aktuellen Wissensstand umsetzen. So ein Projekt eignet sich aber natürlich auch sehr gut, um wieder zu lernen und seine eigenen Fähigkeiten zu vertiefen. Deshalb kann man später einfach neue Techniken in einem 2. Anlauf ausprobieren. Beide Programme sollten dann aber eingereicht werden.

Bei den jeweiligen Sprachen sollten nur Mittel der Sprachen selber verwendet werden. Zum Beispiel wäre es ein falsches Bild, wenn in JavaScript ein spezielles natives Framework oder in Python eine mit C geschriebene Spezialbibliothek verwendet werden, mit der Begründung, dass die Unterschiede zu C/C++ nicht so schlimm wären. Mit Sprachen, wo wirklich keine andere Möglichkeit besteht, sollte ein Hinweis auf die jeweilige Verwendung erfolgen.

Sollte es sich um ein proprietären Entwicklungssystem handeln, können die dazu gehörenden Bibliotheken verwendet werden, wenn sie die selber Technologie verwenden (nativ, VM oder Skript). Dieses gilt zum Beispiel beim C++Builder für die Bibliotheken mit den Datentypen und Funktionen, die von Delphi stammen, aber in einer Art und Weise in den Embarcadero C++Builder ergänzt wurden, dass diese von eher unbedarften Entwicklerinnen und Entwicklern (leider) oft nicht als nicht zu C/C++ gehörend erkannt werden. Auch hier ist der anschließende Vergleich zu Standard C++ und diesen Bibliotheken, aber auch im Vergleich zu anderen Compilern und Bibliotheken interessant.

Mir geht es also auch darum, Entwicklungen und Unterschiede innerhalb einer Programmiersprache zu zeigen, wie zum Beispiel klassisches gegen modernes C++, und Unterschiede der C++ Compiler, hier besonders auch die Unterschiede zwischen aktuellen und älteren Entwicklungssystemen. Tatsächlich entstand die erste Idee für dieses Programm vor dem Hintergrund unterschiedliche C++ Entwicklungssysteme Compiler (Visual Studio, Embarcadero C++Builder, Qt, Eclipse C++, …)

Auch möchte ich die Unterschiede in den verschiedenen Paradigmen auf Komplexität, Erweiterbarkeit und Ressourcenverbrauch herausarbeiten. Deshalb sollten auch die verwendeten Entwicklungssysteme, Mindeststandards (für C++ zum Beispiel C++17, für Java zum Beispiel Java 8) genannt werden, auf welchen Plattformen das Programm läuft und welche Laufzeitumgebung benötigt werden.

Außerdem gibt es sicher Unterschiede bei der individuellen Umsetzung und den verwendeten Algorithmen, und diese können am Ende verglichen und ausgewertet werden und dann allen, die sich beteiligen ein neues Verständnis bringen. Es geht also nicht nur um das beste Ergebnis, auch der Vergleich von erfahrenen und unerfahrenen Programmierenden, notwendige Qualifikationen und vieles andere. Es ist also wichtig, dass auch unerfahrene sich an diesem Projekt beteiligen, und die Anwendung mit dem aktuellen Kenntnisstand umzusetzen. Nur so werden wir auch einen Median der Sprachen und Schlussfolgerungen auf einen Qualifikationsbedarf ziehen.

Ich erwarte auch Unterschiede in den einzelnen Teilaufgaben des Programms, womit sich wieder Erkenntnisse über die Eignung einer speziellen Sprache für Spezialaufgaben zeigen werden. Außerdem ist der Umfang (Quelldateien) und die Komplexität der jeweiligen Lösungen in einer Sprache interessant, und eine Einschätzung, wie erweiterbar und wartbar die jeweiligen Quellen für sprachkundige Entwicklerinnen und Entwickler in der jeweiligen Sprache für eine Lösung sind.

Um eine Vergleichbarkeit zu gewährleisten, sollten alle Programme inklusive der Source- Dateien gesammelt und zentral ausgeführt werden. Daher beteiligen sich alle Teilnehmenden unter einer freien Lizenz an diesem Vergleichsprojekt. Es sollten also keine firmeneigenen Bibliotheken, lizenzpflichtige Drittbibliotheken oder ähnliches verwendet werden.

Es ist nicht das Ziel, einen Gewinner zu finden, am Ende sollten wir alle gewinnen und neue Erkenntnisse gefunden haben. Wir sollten Sprachen, Algorithmen und Paradigmen vergleichbar machen, aber auch die Bedeutung von Algorithmen und der notwendigen Qualifikation herausarbeiten. So gewinnen wir alle zusammen.

Das Projekt ist in 3 große Bereiche aufgeteilt, von denen die beiden ersten jeweils unabhängig sind. Nur für den dritten Teil müssen die beiden vorherigen umgesetzt sein. Damit können bereits nach dem ersten Teil Ergebnisse geliefert werden, oder nur einer oder beide Teile am Anfang umgesetzt werden.

Die Vorgaben sind jeweils exakt umzusetzen, Programmierer neuen dazu, Abkürzungen zu finden. Die Arbeitsschritte sind meistens so gewählt, einen Vergleich bei einer bestimmten Belastung zu ermöglich.

# Anforderungen

Die Idee des Programms ist das Arbeiten mit einer größeren Anzahl von Adressen (Katasterdaten) mit geografischen Koordinaten und geodätische Berechnungen. Das System besteht am Anfang nur aus zwei unabhängigen Datendateien. Alle Eingabedateien und Ausgabedateien werden in einem zentralen Arbeitsverzeichnis gespeichert.

Die Datei mit Berliner Kataster- Daten (alle konkreten Adressen mit Geo- Koordinaten in Berlin, Stand ca. 2010), die 384.860 Datensätze enthält, und eine weitere Datei mit Berliner Hotspots (englisch „Places of Interest“), wie Denkmälern, Museen, Theatern, Gewässern, Krankenhäusern, Hochschulen …. Diese Datei wird von mir aktuell fortlaufend ergänzt, sie umfasst ungefähr 260 Datensätze. Diese Datei erstelle ich manuell, da ich aber nicht alle Hotspots von Berlin kenne, ist es schwierig und leider auch zeitaufwändig. Deshalb die fortlaufende Ergänzung.

Beide Dateien sind jeweils im nationalen Zeichensatz erstellt, während die Daten in der ersten Datei mit Semikolon getrennt gespeichert werden, und die Datei damit eine variable Zeilenlänge hat, verwende ich bei der zweiten Datei bewusst eine feste Satzlänge.

Beide Dateien sind so zu verwenden und dürfen vor der Verwendung nicht verändert werden. Auch in der Realität erhalten wir immer wieder Datenlieferungen in unterschiedlichen Formaten, die wir regelmäßig verarbeiten müssen.

Ich habe mich bewusst gegen eine XML- oder JSON- Datei entschieden, da hier auf jeden Fall Fremdbibliotheken zum Einsatz kommen, die mit großer Wahrscheinlichkeit in einer anderen Sprache oder Technologie erstellt wurden. Das gleiche trifft auf die Verwendung von Datenbanken zu, hier spielen die jeweiligen Treiber eine größere Rolle als die verwendeten Programmiersprachen und die Technologien dieser Sprache.

Die Dateien befinden sich in einem vom Programm unabhängigen Arbeitsverzeichnis und dürfen während der Verarbeitung nicht geändert werden. Alle Ausgaben erfolgen ebenfalls nur in diesem Verzeichnis. Bei jedem Arbeitsschritt wird die Zeit in Millisekunden mit 3 Nachkommastellen ausgeben (Mikrosekunden / 1000), am Ende kann zentral noch einmal die Zeit vom Programmstart bis zur Beendigung ausgegeben werden.

Sollten in der jeweiligen Programmiersprache keine genauen Zeitangaben möglich sein, muss immer aufgerundet werden (sollen zum Beispiel nur Millisekunden angegeben werden können, muss 1 Millisekunde zur Zeitdifferenz addiert werden).

Während der Verarbeitung werden die Daten in einem dynamischen Array (Speicherung in einem zusammenhängendem Speicherbereich, der sich vergrößert/verkleinert), in diesem sortiert und alle Berechnungen durchgeführt.

# Kataster- Daten

Die Datei hat den Namen „berlin\_infos.dat“ und wurde von mir aus den von dem Land Berlin im Rahmen der Open-Data Initiative bereitgestellten Katasterdaten erstellt. Berlin ist die größte Stadt in Deutschland und in 12 Stadtbezirke eingeteilt. Zu jedem Bezirk gehören mehrere Ortsteile. Alle Stadtbezirke zusammen haben 95 Ortsteile. Die Datei enthält 384.860 Datensätze und hat eine Größe von 33,2 MB.

Diese Daten sind keine personenbezogenen Daten, wie Adressen im Allgemeinen suggerieren, auch wenn jeweils konkrete Personen hier wohnen könnten. Es handelt sich aber um ein Register aller Grundstücke und Häuser. Der Datenstand ist nicht aktuell, die Daten dürften um das Jahr 2010 herum erstellt worden sein. Das spielt aber für die Idee der Vergleichbarkeit von Programmiersprachen und Paradigmen in diesem Projekt keine Rolle. Es geht nur darum, eine aussagekräftige Menge an Daten zu haben.

Die folgende Karte zeigt die Verwaltungsgliederung von Berlin mit Stadtbezirken und Stadtteilen nach der Verwaltungsreform von 2001.

Ein Bild, das Karte enthält.

Automatisch generierte Beschreibung

Karte aus Wikipedia ([Verwaltungsgliederung Berlin](https://de.wikipedia.org/wiki/Verwaltungsgliederung_Berlins)), *Wikimedia Commons,* Urheber [Tubs](https://commons.wikimedia.org/wiki/User:TUBS)

Neben den Adressen sind auch die Stadtbezirke (aktuell und vor der Reform) und Ortsteile in der Datei erfasst. Dazu kommen die geografischen Koordinaten.

## Aufbau der Datei

Die Adressdatei ist eine mit Semikolon getrennte CSV- Datei ohne Überschriften. Die Datei ist im nationalen Zeichensatz gespeichert, sie enthält auch nur nationale Zeichen (Ansi Zeichensatz Windows-1252, basierend auf ISO 8859-1). Die Gleitkommazahlen sind in der Punktschreibweise, ohne Tausenderzeichen geschrieben.

Enthalten sind die Felder in der angegebenen Reihenfolge:

|  |  |  |
| --- | --- | --- |
| Feld | Beschreibung | Datentyp |
| City | Stadt | string |
| Street | Straße | string |
| StreetNumber | Hausnummer mit Ergänzung (a, b, …) | string |
| ZipCode | PLZ | string |
| urban\_unit | Stadtbezirk | string |
| old\_unit | Stadtbezirk vor Reform | string |
| District | Stadtteil | string |
| Latitude | Breitengrad der geografischen Position | double |
| Longitude | Längengrad der geografischen Position | double |

Der Datentyp double sollte in allen verwendeten Sprachen eine Gleitkommazahl nach [IEEE 754 / IEC 60559](https://de.wikipedia.org/wiki/IEEE_754) mit doppelter Genauigkeit sein (64bit). Bei Programmiersprachen ohne feste Typen muss ein entsprechender Hinweis erfolgen.

Neben den Feldern, die aus der Datei eingelesen werden, soll die im Programm verwendete Datenstruktur zusätzlich noch folgende Felder enthalten, die später für Berechnungen und Auswertungen genutzt werden.

|  |  |  |
| --- | --- | --- |
| Feld | Beschreibung | Datentyp |
| Distance | Entfernung vom Referenzobjekt zu den Koordinaten der Adresse | double |
| Angle | Kurswinkel vom Referenzobjekt zu den Koordinaten der Adresse | double |

Alle Verarbeitungen und Berechnungen erfolgen mit dieser Datenstruktur. Die Verarbeitungsfunktionen können Bestandteil der Klasse sein, oder als separate freie Methoden implementiert werden. Alle Funktionen müssen aber vom Hauptprogramm getrennt und als eine Bibliothek nutzbar und wiederverwendbar sein.

## Vorverarbeitungen der Kataster- Daten

In einem ersten Schritt werden die Daten eingelesen und einige Operationen mit den Kataster- Daten durchgeführt, die unterschiedlich komplex sind.

### Einlesen der Daten aus der sequenziellen Eingabedatei

Die oben beschriebene Datei soll sequenziell eingelesen und in eine Datenstruktur *(für objektorientierte Sprache zum Beispiel eine Klasse, strukturierte Sprache eine Struktur oder freie Variablen)* eingelesen und in ein dynamisches Array gespeichert werden *(zusammenhängender Speicher, der sich dynamische vergrößert)*. Dabei ist die Größe der Datei vorher unbekannt und **darf nicht verwendet werden**, auch darf dieser Speicherbereich semantisch nie größer sein

Der Speicherbereich darf also nicht für 500.000 angelegt und später verkleinert werden.

Auch wenn es im Moment nur Daten aus Berlin sind, wird die Stadt in den Operationen und Vergleichen immer so verwendet, als wenn es auch Daten anderer Städte geben könnte. Eventuell finden wir ähnliche Daten von anderen Orten und ergänzen diese später. Die Daten werden im dynamischen Speicherbereich in der Reihenfolge gespeichert, in der sie eingelesen werden.

### Berechnen der Entfernung und des Kurswinkel vom Berechnungsziel zum jeweiligen Haus

Hierzu soll eine Berechnung des Abstands von einem Berechnungsziel (Geo- Koordinaten) zu den in den Kataster- Daten gespeicherten Positionen implementiert werden. Außerdem soll der Kurswinkel von dem Berechnungsziel zu der Position in den Daten berechnet werden.

Die Berechnung erfolgt nach den Formeln der sphärischen Trigonometrie über Orthodrome.

|  |  |  |
| --- | --- | --- |
| math. Symbol | Beschreibung | Variable |
|  | Geografische Breite |  |
|  | Geografische Länge |  |
| A( | Anfangspunkt der Berechnung |  |
| B( | Endpunkt der Berechnung |  |
|  | Geografische Breite des Anfangspunkts im Bogenmaß | phiA |
|  | Geografische Länge des Anfangspunkts im Bogenmaß | lambdaA |
|  | Geografische Breite des Endpunkts im Bogenmaß | phiB |
|  | Geografische Länge des Endpunkts im Bogenmaß | lambdaB |
|  | A | Zeta |
| R0 | Mittlerer Radius der Erde 6.371.000,785 m | R |
|  | Kurswinkel bei A | alpha |
|  | Kurswinkel bei B | beta |
|  | Kurswinkel |  |

Die Geo- Koordinaten im Programm werden als Grad- Werte angegeben, werden aber für die Berechnung im Bogenmaß benötigt. Die Geo- Koordinaten und müssen deshalb zu Beginn Berechnung ins Bogenmaß umgerechnet werden.

Der Zentriwinkel wird mit folgender Formel berechnet. Dabei gehen wir davon aus, dass alle Werte im Bogenmaß vorliegen.

Für die Bestimmung des Abstands wird jetzt mit dem mittleren Erdradius R0 multipliziert, da wir nur kurze Entfernungen bestimmen, ist dieser in Metern angegeben. Da wir die Koordinaten der Punkte bereits im Bogenmaß angegeben haben, ist das Ergebnis der Abstand in Metern.

Da wir die Richtung vom Berechnungspunkt bestimmen wollen, benötigen wir den Winkel für die Berechnung. Hier müssen wir zwischen rechtsweisenden und linksweisenden Richtungen unterscheiden, der Kurswinkel wird immer mit 0° in Nordrichtung angegeben, 90° entspricht der Ostrichtung.

Die Bestimmung der Kurswinkel von und erfolgt mit Hilfe von .

Im nächsten Schritt muss der Winkel aus dem Bogenmaß wieder in eine Gradzahl umgerechnet werden.

Zur Bestimmung des Kurswinkels müssen wir abschließend zwischen (mehr) ostwärts und (mehr) westwärts weisenden Winkel unterschieden und eventuell negative Startwerte beachtet werden, da sich diese viertelkreisig ergeben (das liegt in der Natur der trigonometrischen Funktionen. Durch die Formel wird immer der Winkel von Norden gerechnet (für Nordhalbkugel, sonst Süden), da für die Navigation und die Geodäsie die Kurswinkel aber immer als voller Kreis, beginnend im Norden über Osten berechnet werden, müssen wir dieses abschließend korrigieren.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| Östliche Richtung |  |  |  |  |  |
| Westliche Richtung |  |  |  |  |  |

Damit haben wir für die Formeln zur Umsetzung der Berechnung.

Um diese Funktionen zu testen, berechnen wir jetzt den Abstand jeder Position in den Kataster- Daten (Endpunkt der Berechnung) zum Berliner Fernsehturm (Anfangspunkt der Berechnung). Für den Anfangspunkt werden die geografischen Koordinaten 52.520803, 13.40945 verwendet. Außerdem wird der Kurswinkel von dem Fernsehturm zur jeweiligen Position bestimmt.

Die Funktion hat den zulässigen Wertebereich (-1, 1). Bei der Berechnung kommt es, eigenartiger Weise in unterschiedlicher Häufung bei verschiedenen Sprachen, zu für Gleitkommazahlen übliche minimale Rundungsdifferenzen für den gewählten Datentyp double. Für Argumente kleiner als -1 wird dieses auf -1 gesetzt, für Argumente größer als 1 auf 1.

Diese beiden Informationen werden auf 3 Nachkommastellen gerundet und dann in die Datenfelder Distance and Angle gespeichert. Bei der Berechnung kann es zu minimalen Abweichungen kommen, die durch die Rundung ausgeglichen werden und die Berechnungen so nachvollziehbarer gemacht werden. Dabei soll #,###5 aufgerundet werden (kaufmännische Rundung).

Weiterführende Informationen zu der Berechnung finden sich auf den folgenden Wikipedia- Seiten:

* <http://de.wikipedia.org/wiki/Geografische_Koordinaten>
* <https://de.wikipedia.org/wiki/Orthodrome>

### Sortierung der Grunddaten

In diesem Schritt soll die zuvor in den Arbeitsspeicher eingelesene Datenstruktur sortiert werden. Das Sortieren ist ein essentieller Algorithmus, und gerade beim effektiven Sortieren einer Datenmenge kann es Unterschiede geben. Sollte die Sprache über Sortierfunktionen verfügen, sollten diese verwendet werden. Um eine Vergleichbarkeit zu ermöglichen, muss diese Sortierung im Programm immer physikalisch erfolgen.

Bei der alphanumerischen Sortierung soll standardmäßig Groß- und Kleinbuchstaben unterschieden werden *(case sensitive)*. Außerdem sollen die deutschen Umlaute und Sonderzeichen nach DIN 5007 sortiert werden.

Nach DIN 5007 werden deutsche Spezialbuchstaben ersetzt (ä = ae, ß = ss). Für alle sonstigen (fremdsprachigen) diakritischen Zeichen gilt im deutschsprachigen Raum, dass sie einheitlich weggelassen werden; so auch alle Akzente, Tilde, Makron: á und a, é und e, è und e, ç und c, ñ und n sind jeweils gleich.

Dabei werden folgenden Kriterien berücksichtigt:

* City, alphanumerisch aufsteigend nach DIN 5007
* urban\_unit, alphanumerisch aufsteigend nach DIN 5007
* District, alphanumerisch aufsteigend nach DIN 5007
* ZipCode, alphanumerisch aufsteigend (enthält nur Ziffern)
* Street, alphanumerisch aufsteigend nach DIN 5007
* Streetnumber, Hausnummer numerisch, Ergänzung alphanumerisch aufsteigend sortiert   
  **Achtung:** case insenstive, nur Standardbuchstaben a..z und A..Z

Alternativ können andere Algorithmen, wie Indizierung individuell untersucht werden. Die Ergebnisse sollten dann aber auch geteilt werden und sind nicht Bestandteil des Vergleichs. Aber wir alle wollen lernen.

Später muss eine zweite Sortierung nach den Feldern Distance und Angle erfolgen können, aber auch völlig andere Sortierung sollten auf der Datenstruktur möglich sein.

### Schreiben in eine sequenzielle Datei

Die in dem dynamischen Speicher befindlichen Daten sollen in der sequenziellen, formatierten Textdatei „output\_streets.txt“ im Arbeitsverzeichnis des Programms gespeichert werden. Diese Datei ist im nationalen Zeichensatz zu speichern (unter Windows ANSI).



Alternativ sollte diese Ausgabe auch auf der Konsole / Bildschirm möglich sein. Für die Kontrolle und Auswertbarkeit ist aber eine Datei besser geeignet. Mit dieser Datei können über KDiff3 oder ein anderes Programm wie der Total Commander die Ergebnisse der Sortierung und Berechnung überprüft werden, und die Geschwindigkeit beim Schreiben einer sequenziellen Datei im nationalen Zeichensatz ermittelt werden.

Dabei werden folgende Informationen mit den angegebenen Formaten und Ausrichtungen geschrieben. Zusätzliche Zeichen sind separat aufgeführt. Einige Daten werden zusammengefasst, bevor sie gespeichert werden. Alternativ könnte man natürlich auch Längen rechnen, um eine entsprechende Ausgabe zu erzeugen. Da es sich um eine Ausgabe handelt, werden die Zahlen mit Tausender- Separator und Dezimalpunkt mit deutschem Format geschrieben. Zusätzlich werden Trennzeichen und die Maßeinheiten geschrieben (m für Distance, ° für Angle). Um eine Vergleichbarkeit der Dateien zu ermöglichen, muss das Format exakt eingehalten werden.

Die folgende Tabelle definiert die Ausgabe:

|  |  |  |
| --- | --- | --- |
| Felder | Ausrichtung | Breite, Komma |
| ZipCode | Links | 5 |
| Leerzeichen | links | 1 |
| City + " - " + District + ", " | links | 75 |
| Street + " " + Streetnumber | links | 65 |
| = | links | 1 |
| Distance | rechts | 15,3 |
| "m /" | links | 3 |
| Angle | rechts | 7,3 |
| "°" | links | 1 |

Die folgende Abbildung zeigt einen Ausschnitt der Ausgabe dieser Datei.

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

Damit ist der erste Aufgabenblock abgeschlossen.

## Arbeiten mit der Verzeichnisstruktur

Um das Arbeiten in Verzeichnisstrukturen für verschiedene Programmiersprachen und Entwicklungssysteme vergleichbar zu machen, werden die Informationen aus den Daten im dynamischen Speicher nun in einer komplexen Verzeichnisstruktur abgespeichert und später wieder eingelesen.

Dabei wird in Verzeichnisbaum für die Stadt, Stadtbezirke und Stadtteile erstellt, in dem die Dateien für alle Straßen gespeichert werden. Diese Struktur besteht, basierend auf den verwendeten Eingabedaten, aus 109 Ordner und 10.853 Dateien, mit einer logischen Gesamtgröße von 23,4 Mbyte.

### Schreiben in eine Verzeichnisstruktur

Diese Verzeichnisse sollen in einem Prozess erstellt, und die Dateien in diesen geschrieben werden. Dabei entspricht die notwendige Sortierung nicht mehr der vorher durchgeführten. Sollte es notwendig sein, die Daten sortiert zu verarbeiten, muss diese entsprechend der hier genutzten Reihenfolge vorher durchgeführt werden und gehört mit zu diesem Arbeitsschritt. Generell sollte es also auch möglich sein, eine völlig unsortierte Datenmenge zu verarbeiten.

Es gelten folgende Vorgaben. Die Verzeichnisse können in einem beliebigen Pfad gespeichert werden. Dieser wird als Parameter übergeben. Für dieses Vergleichsprogramm wird hier das Arbeitsverzeichnis verwendet.

In der ersten Verzeichnisebene werden die Namen der Orte (Datenfeld City) als Verzeichnis angelegt. In unserem Fall ist das nur Berlin.

In der zweiten Ebene folgen die Namen der Stadtbezirke für den jeweiligen Ort (Datenfeld urban\_unit) als Unterverzeichnisse erzeugt.

In der dritten Ebene folgen die Namen der im jeweiligen Stadtbezirk existieren Stadtteile (Datenfeld district) als Unterverzeichnisse.

Das folgende Bild zeigt den Verzeichnisbaum.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

In den erstellten Verzeichnissen der dritten Ebene werden alle Hausnummern zu der zugeordneten Straße in eine CSV- Datei mit dem Namen der jeweiligen Straße mit der Extension „.csv“ gespeichert. Die Kataster- Datei enthält nur Straßennamen, die sich auch als Dateiname eignen.

Alle Dateien werden mit dem nationalen Zeichensatz gespeichert.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Da sich beim Einlesen aus den drei Ebenen der Verzeichnisstruktur und dem Namen der jeweiligen CSV- Datei notwendige Informationen ergeben, werden in diese Datei nur noch die verbleibenden Daten in die Dateien geschrieben.

Die folgende Tabelle enthält alle Daten, die mit Semikolon getrennt in die Dateien gespeichert werden.

|  |  |  |  |
| --- | --- | --- | --- |
| Feld | Beschreibung | Datentyp | max Genauigkeit |
| StreetNumber | Hausnummer mit Ergänzung (a, b, …) | string |  |
| ZipCode | PLZ | string |  |
| old\_unit | Stadtbezirk vor Reform | string |  |
| Latitude | Breitengrad der geografischen Position | double | 9 |
| Longitude | Längengrad der geografischen Position | double | 9 |
| Distance | Breitengrad der geografischen Position | double | 3 |
| Angle | Kurswinkel vom Referenzziel zu dieser Adresse | double | 3 |

Die folgende Abbildung zeigt einen Ausschnitt aus der Datei Bismarckstraße.csv im Verzeichnis Berlin/Charlottenburg-Wilmersdorf/Charlottenburg. **Die geografischen Angaben müssen mit genügend Nachkommastellen gespeichert werden (maximale Genauigkeit in der obigen Tabelle), um keine Genauigkeit zu verlieren. Es sollten aber auch keine unnötigen Stellen verwendet werden (Ressourcen sparen).**

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

Dabei müssen die Hausnummern (Feld: StreetNumber) aufsteigend sortiert werden. Wieder haben dabei die in der Haunummer vorkommenden Zahlen numerisch, und die Ergänzung alphanumerisch sortiert zu werden. Es gibt Adressen, die mehreren PLZ- Bereichen zugeordnet und daher mehrfach in der Datei enthalten sind. In diesem Fall werden die Datensätze als letztes auch noch nach der PLZ (Feld: ZipCode) sortiert.

Die folgende Abbildung zeigt dieses am Beispiel einer Straße im Stadtbezirk Pankow. Die Sortierung ist aber generell diese Möglichkeit berücksichtigen.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

### Löschen der Verzeichnisstruktur

Um Verarbeitungen wiederholen zu können, müssen die Verzeichnisse und Dateien mit den Adressdaten auch wieder gelöscht werden können. Dabei dürfen im Verzeichnis alle Unterverzeichnisse rekursiv gelöscht werden, nicht aber die Dateien, die sich im Verzeichnis selber befinden. Das zu verarbeitende Verzeichnis wird als Parameter an die Funktion übergeben.

Dieser Schritt muss für Wiederholungen in der Verarbeitungsreihenfolge später als erster ausgeführt werden, um vor dem Erstellen eventuell vorhandene Daten zu löschen.

Ich habe diesen Schritt aber zum besseren Verständnis in den Anforderungen erst nach dem Erzeugen der Strukturen aufgeführt.

### Freigeben des dynamischen Speichers mit den Kataster- Daten

Hier werden die dynamisch im Speicher abgelegten Daten vollständig gelöscht. Dabei ist ein Vergleich mit den Sprachen, die mit einer virtuellen Maschine arbeiten, sehr schwierig, da sie diesen Prozess verschleiern. Da aber die regelmäßige Freigabe von Speicher zu Prozessen gehört, die dynamische Belegung auch zu einer Fragmentierung des Speichers führen muss, ist es notwendig, diese mit zu integriert. Gerade bei einer hohen Nutzung des Arbeitsspeichers kommt es in den VM- Sprachen immer wieder zur einer schwer zu kontrollierenden Garbage Collection. Dabei steigt die Auslastung der CPU sehr schnell und kann zu massiven Beeinträchtigung von aktuellen Prozessen führen. Wenn sehr viel Prozessorzeit für die Garbage Collection aufgewendet wird, erfolgen diese zudem häufiger, oder sie dauern zu lang. Eine zu hohe Belegungsrate für Objekte auf dem verwalteten Heap führt außerdem zu noch häufigeren Garbage Collections.

Sollte ein gezieltes Freigeben nicht möglich sein, sollte die Zeit der folgenden Carbage Collage gemessen werden. Ist dieses nicht möglich, wird die Zeit geschätzt.

### Einlesen der Kataster- Daten aus der Verzeichnisstruktur

In diesem Prozessschritt geht es darum, die Kataster- Daten, die verteilt in den Dateien der vorher erzeugten Verzeichnisstruktur gespeichert sind, wieder in den Arbeitsspeicher einzulesen. Wie vorher werden diese in einem dynamischen Array abgespeichert. Dafür sind die im ersten Bereich erstellten Datenstrukturen und Variablen wiederzuverwenden.

Um die zu erwartenden, negativen Wechselwirkungen in Sprachen mit einem Garbage Collector zu vermeiden, sollte für diese Sprachen die Verarbeitung gegebenenfalls unterbrochen werden, und nach der erfolgten Freigabe des Speichers wieder aufgenommen werden.

Diese Dateien sind im nationalen Zeichensatz gespeichert. Auch hier darf die Anzahl der Sätze nicht verwendet werden, die Daten sind wieder dynamisch aufzubauen.

Während des Einlesens werden folgende Daten zugeordnet:

|  |  |
| --- | --- |
| Name des Verzeichnisses der 1. Ebene | City |
| Name des Verzeichnisses der 2. Ebene | Urban\_Unit |
| Name des Verzeichnisses der 3. Ebene | District |
| Name der Datei | Street |

## Nachverarbeitung der Daten

In diesem Abschnitt werden die Berechnungsergebnisse mit den geografischen Koordinaten aus dem vorherigen Teil (Abstand und Kurswinkel) genutzt, um das Partitionieren des dynamischen Arrays zu testen und nur mit einem Teil der Daten weiterzuarbeiten.

### Partitionierung von Datensätze

In diesem Schritt werden die Berechnungsergebnisse (Entfernung und Kurswinkel vom Fernsehturm zu jeder Adresse) genutzt, um den dynamischen Speicher aufzuteilen (partition). Dazu werden alle Adressen, deren Entfernung vom Fernsehturm kleiner als 1km (1.000 m) ist, in dem dynamischen Array nach vorne geschoben, alle Elemente die weiter entfernt sind, in den hinteren Bereich.

Dabei geht es um die Geschwindigkeit der Speicheroperationen im Zusammenhang mit den notwendigen Vergleichsoperatoren.

Sollte die Programmiersprache keinen partition- Algorithmen (Aufteilung) unterstützen, ist die Datenmenge nach dem Datenfeld Distance zu sortieren und der Index des ersten Datensatzes, der sich außerhalb des definieren Bereiches befindet, zu bestimmen.

### Neusortierung mit Entfernung und Winkel

Nachdem die Datensätze im Speicher verschoben wurden, werden sie jetzt sortiert. Dabei sollen nur die im vorherigen Schritt ausgewählten Datensätze sortiert werden. Dazu werden die Datensätze nach dem Abstand (das Feld Distance) aufsteigend sortiert. Bei gleichem Abstand wird zusätzlich der Kurswinkel (das Datenfeld Angle) verwendet. Damit wird eine neue Sortierreihenfolge verwendet.

Zur Kontrolle werden die Datensätze in die Ausgabedatei "output\_distance.txt" geschrieben. Dabei wird folgendes Format verwendet.

|  |  |  |
| --- | --- | --- |
| Felder | Ausrichtung | Breite, Dezimalstellen |
| ZipCode | Links | 5 |
| City + " - " + District | Links | 75 |
| Street + " " + Streetnumber | Links | 65 |
| "=" | Links | 1 |
| Distance | Rechts | 15,3 |
| "m /" | Links | 3 |
| Angle | Rechts | 7,3 |
| "°" | Links | 1 |

Die folgende Abbildung zeigt einen Ausschnitt der Datei mit den ersten Datensätzen. Mit Datendatei sollten 1.653 Datensätze der Bedingung eines Abstands < 1km vom Referenzpunkt (Fernsehturm) entsprechen und in die Datei geschrieben werden.

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

# Ausgabe

Das Programm sollte folgende Ausgabe generieren. Die Zeitmessung sollte vor dem Aufruf der jeweiligen Methode beginnen (nicht erst innerhalb), und beendet werden, nachdem die Funktion zurückgekommen ist.

Die Anzahl zeigt die jeweils gelesenen Datensätze.

Ich habe das in einem einheitlichen template durchgeführt. Die Funktion f1() entspricht dem jeweiligen Schritt, die Funktion f2() liefert hier die Anzahl der Datensätze für die Ausgabe.

template <typename func1, typename func2>

void Call(std::string const& text, func1 f1, func2 f2) {

auto func\_start = chrono::high\_resolution\_clock::now();

f1();

auto func\_ende = chrono::high\_resolution\_clock::now();

auto time = chrono::duration\_cast<chrono::microseconds>(func\_ende - func\_start).count();

cout << left << setw(55) << text

<< " Size = " << right << setw(10) << f2()

<< " in " << setw(12) << setprecision(3) << time/1000.0 << " ms" << endl;

}

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ich habe die konkreten Zeiten in der Ausgabe durch xx.xxx,xxxx ms ersetzt, da es im ersten Entwurf nicht um einen Vergleich geht.

# Berechnungen mit Koordinaten

In diesem Teil sollen statistische Berechnungen mit den Koordinaten durchgeführt werden. Gerade hier ist mit Unterschieden in den Algorithmen, Paradigmen und damit auch Programmiersprachen zu rechnen, da externe Einflussfaktoren, wie die Geschwindigkeit der Festplatte keine große Rolle mehr spielen. Es geht um das Handling im dynamischen Arbeitsspeicher und die Ausführung von Vergleichen und mathematischen Berechnungen. Neben den, für die Berechnungen notwendigen Katasterdaten, soll für diese Berechnungen eine zweite Datenstruktur als dynamisches Feld aufgebaut werden.

Ich habe mich bei der Auswahl der Maßzahlen auf einfache Größen beschränkt, um die Schwierigkeit und Komplexität des Programms nicht unnötig zu steigern, und weiterhin eine Umsetzung in einer überschaubaren Zeit zu ermöglichen. Im folgenden Abschnitt erfolgt erst einmal die Erklärung der verwendeten Maßzahlen. Dazu kommen auch in diesem Abschnitt die Berechnungen der Abstände und Kurswinkel aus dem Abschnitt „Berechnen der Entfernung und des Kurswinkel vom Berechnungsziel zum jeweiligen Haus“ (Seite 9).

## Beschreibung der verwendeten Maßzahlen

### Minimum und Maximum

Das größte beziehungsweise kleinste Element sind Begriffe aus dem mathematischen Teilgebiet der Ordnungstheorie. Das größte Element wird auch als Maximum bezeichnet, dementsprechend spricht man beim kleinsten Element vom Minimum.

Ein Element einer geordneten Menge ist das größte Element der Menge, wenn alle anderen Elemente kleiner sind. Es ist das kleinste Element der Menge, wenn alle anderen Elemente größer sind. Weder das größte noch das kleinste Element einer Menge muss existieren, ist aber im Fall seiner Existenz jeweils bis auf Assoziiertheit eindeutig bestimmt. Da in unserem Fall die Elemente nicht eindeutig sind, erweitern wir die Bedingung und es gilt folgendes:

* ist größtes Element von
* ist kleinstes Element von

Eine Maximum- Funktion liefert das größte ihrer Argumente als Wert, eine Minimum- Funktion liefert das kleinste ihrer Argumente.

Die Abkürzungen *„max“* und *„min“* sind gebräuchlich, seltener auch *„Max“* und *„Min“*.

Quelle: [Wikipedia](https://de.wikipedia.org/wiki/Gr%C3%B6%C3%9Ftes_und_kleinstes_Element)

### Mittelwert

Ein Mittelwert (kurz auch nur Mittel; anderes Wort Durchschnitt) ist eine Zahl, die aus gegebenen Zahlen nach einer bestimmten Rechenvorschrift ermittelt wird. Gebräuchlich sind Rechenvorschriften für das arithmetische, das geometrische und das quadratische Mittel. Mit dem Wort Mittel oder Durchschnitt ist meistens das arithmetische Mittel gemeint.

Mittelwerte werden am häufigsten in der Statistik angewendet. Der Mittelwert ist ein Kennwert für die zentrale Tendenz einer Verteilung. Eng verwandt ist der arithmetische Mittelwert mit dem Erwartungswert einer Verteilung. Während der Mittelwert aus konkreten vorliegenden Zahlenwerten ermittelt wird, beruht der Erwartungswert auf der theoretisch zu erwartender Häufigkeit.

Quelle: [Wikipedia](https://de.wikipedia.org/wiki/Mittelwert)

#### Arithmetischer Mittelwert

Das arithmetische Mittel, auch arithmetischer Mittelwert genannt (umgangssprachlich auch als Durchschnitt bezeichnet) ist ein Begriff in der Statistik. Es ist ein Lageparameter. Man berechnet diesen Mittelwert, indem man die Summe der betrachteten Zahlen durch ihre Anzahl teilt.

Das arithmetische Mittel einer Stichprobe wird auch empirischer Mittelwert genannt.

Im folgenden Text bezieht sich der Begriff „Mittelwert“ immer auf das arithmetische Mittel.

### Median

In der Statistik ist der Median – auch Zentralwert genannt – ein Mittelwert und Lageparameter. Der Median der Messwerte einer Urliste ist derjenige Messwert, der genau „in der Mitte“ steht, wenn man die Messwerte der Größe nach sortiert. Beispielsweise ist für die ungeordnete Urliste 4, 1, 37, 2, 1 der Messwert 2 der Median, der zentrale Wert in der geordneten Urliste 1, 1, 2, 4, 37.

Im Allgemeinen teilt ein Median einen Datensatz, eine Stichprobe oder eine Verteilung so in zwei gleich große Teile, dass die Werte in der einen Hälfte nicht größer als der Medianwert sind und in der anderen nicht kleiner.

Der Median teilt eine Liste von Werten in zwei Teile. Er kann auf folgende Weise bestimmt werden:

* Alle Werte werden (aufsteigend) geordnet.
* Wenn die Anzahl der Werte ungerade ist, ist die mittlere Zahl der Median.
* Wenn die Anzahl der Werte gerade ist, wird der Median meist als arithmetisches Mittel der beiden mittleren Zahlen definiert, die dann Unter- und Obermedian heißen.

Quelle: [Wikipedia](https://de.wikipedia.org/wiki/Median)

## Berechnungen für den gesamten Datenbestand

Für die folgenden Berechnungen dürfen bisherige, spezielle Informationen über die Daten nicht verwendet werden. Generell sollten diese Berechnungen für jedes Cluster von Daten mit Katasterdaten und Geokoordinaten durchführbar sein. Alle notwendigen Berechnungen sind immer auf allen Daten der zu untersuchenden Gesamtheit durchzuführen, es dürfen auch keine vorher bestimmten Werte wiederverwendet werden.

Die folgenden Schritte sind im ersten Schritt auf allen Katasterdaten durchgeführt, später auf Teilmengen von dieser wiederholt.

Bestimmung der Anzahl der jeweiligen Datensätze

* count

Dann werden die Adressen mit der nördlichsten, östlichsten, südlichsten und westlichsten Geokoordinate bestimmt (Minimum und Maximum, jeweils von Latitude und Longitude). Dadurch kann gedanklich ein Rechteck um Berlin gezeichnet und die vier Eckpunkte bestimmt werden.

* northwest corner
* northeast corner
* southwest corner
* southeast corner

Aus den Eckpunkten lässt sich die Länge der Seiten des Rechtsecks berechnen (Breite, Höhe).

* height
* width

Da es bereits bei der Größe von Berlin eine Abweichung zwischen der nördlichen und südlichen Breite ergibt, ist die Breite unbedingt als Entfernung zwischen den nördlichen Extrempunkten zu berechnen.

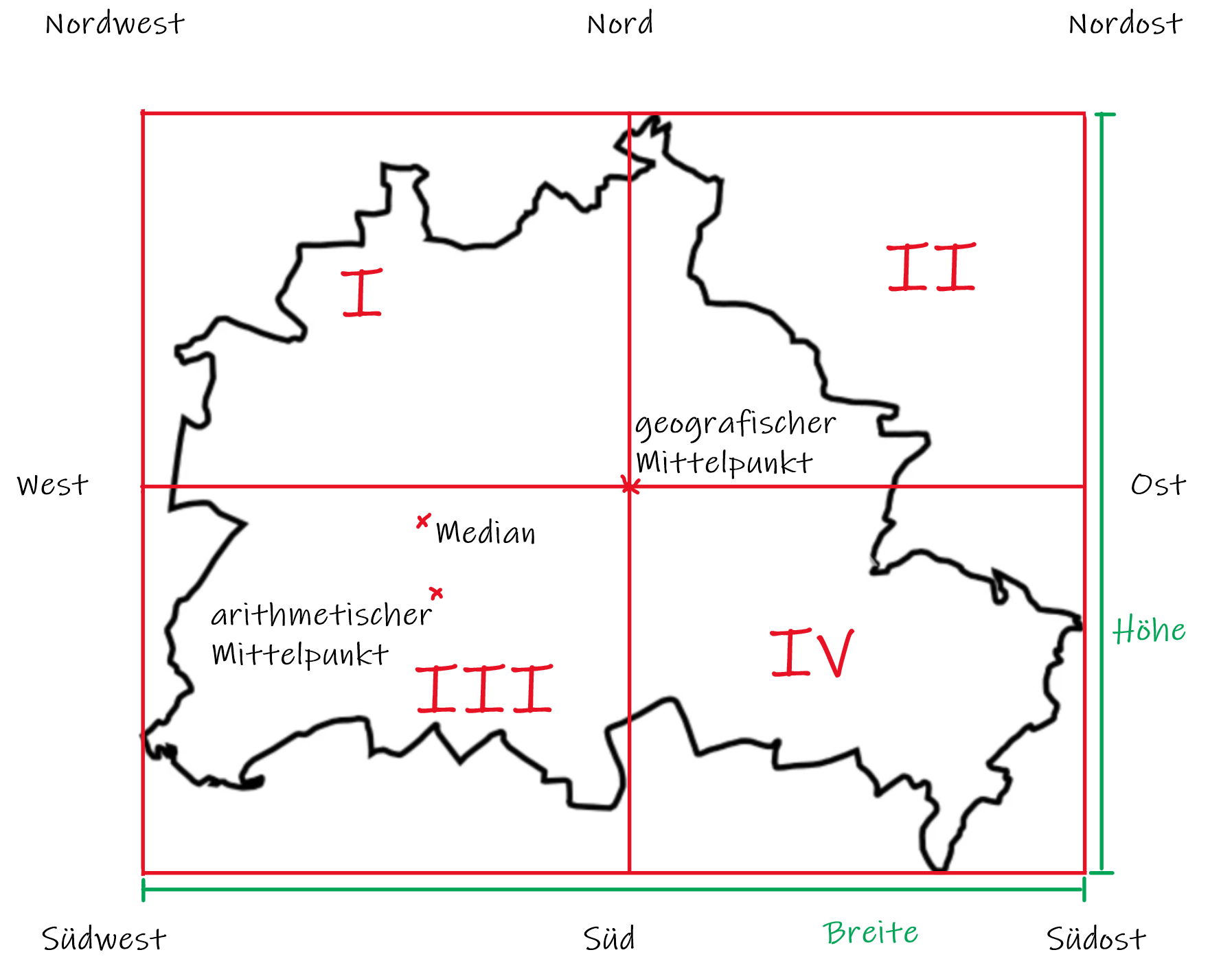
Anschließend werden die Seiten des Rechtecks halbiert und neben dem geografischen Zentrum auch die Mittelpunkte der Strecken (Nord, Ost, Süd, West) bestimmt. So wird Berlin nochmals in vier gleichgroße Rechtecke (I, II, III, IV) eingeteilt (siehe folgende Skizze).

* north center
* western center
* eastern center
* south center

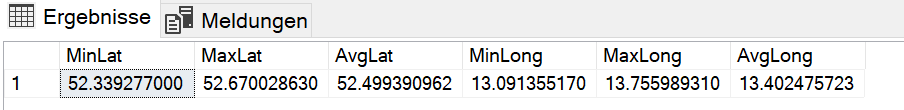
Schließlich werden anhand der Daten der geografische Mittelpunkt, der arithmetische Mittelpunkt und der Median für die Geokoordinaten berechnet.

* center
* average
* median

Die folgende Abbildung zeigt die eben beschriebenen Punkte und sich daraus ergebene Rechtecke mit dem Umriss von Berlin. Die Angaben in der Skizze sind nur geschätzt und dienen der Verdeutlichung der Aufgabenstellung. Das sich ergebene Rechteck wird sich außerdem innerhalb des hier dargestellten befinden, da wir konkrete Adressen verwenden, die sich immer innerhalb der Stadtgrenzen befinden.



Die folgende Abbildung zeigt die Extremwerte und die Durchschnittswerte aus der ursprünglichen Datenbank mit den Katasterdaten zur Kontrolle. Bei der Berechnung kann es wieder zu geringfügigen Rundungsdifferenzen kommen.

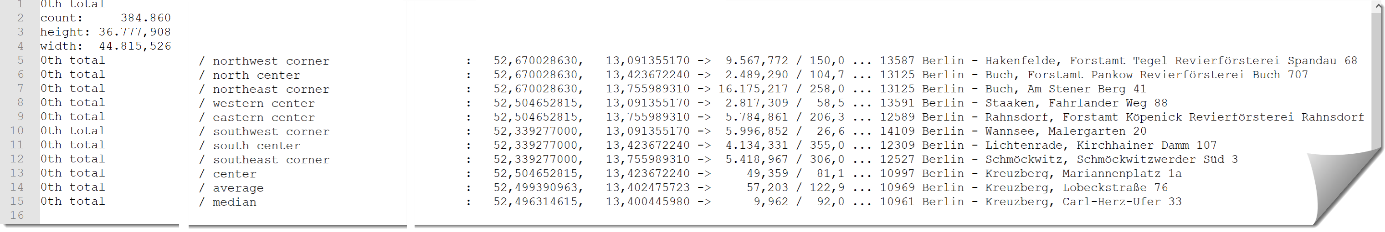


In der folgenden Tabelle sind die Punkte und Rechtecke noch einmal eindeutig definiert.

|  |  |
| --- | --- |
| math. Symbol | Beschreibung |
|  | Geokoordinaten der nordwestlichsten Ecke |
|  | Geokoordinaten des nördlichen Mittelpunkts |
|  | Geokoordinaten der nordöstlichen Ecke |
|  | Geokoordinaten des östlichen Mittelpunkts |
|  | Geokoordinaten der südöstlichen Ecke |
|  | Geokoordinaten des südlichen Mittelpunkts |
|  | Geokoordinaten der südwestlichsten Ecke |
|  | Geokoordinaten des westlichsten Mittelpunkts |
|  | Geokoordinaten des geografischen Mittelpunkts |
|  | Geokoordinaten des arithmetischen Mittelpunkts |
|  | Geokoordinaten des Mittelpunkts aus den Medianen |
|  | Rechteck um alle Adressen |
| Width | Distanz zwischen südwestlicher und südöstlicher Ecke |
| Height | Distanz zwischen nordöstlicher und südöstlicher Ecke |
|  | nordwestliches Viertel (1. Quarter) |
|  | nordöstliches Viertel (2. Quarter) |
|  | südwestliches Viertel (3. Quarter) |
|  | südöstliches Viertel (4. Quarter) |

Neben der genauen Angabe der obigen Geokoordinaten für die Eck- und Mittelpunkte und der Länge und Breite des umschließenden Rechtecks sollen auch die Adressen bestimmt werden, die diesen Punkten jeweils am nächsten liegen. Dazu sind alle Adressen der Kataster- Datei zu prüfen. Außerdem soll zu der jeweiligen Adresse auch der Abstand angegeben werden, und der Kurswinkel, in dem die nächstgelegene Adresse von dem Punkt aus liegt.

Die folgende Abbildung zeigt einen (zusammengeschnittenen) Ausschnitt für den ersten Berechnungsschritt.



### Speicherung der Ergebnisse in eine dynamische Speicherstruktur

Die berechneten geografischen Punkte werden in eine dynamische Speicherstruktur geschrieben.

|  |  |
| --- | --- |
| Inhalt | Typ |
| name of the point | string |
| category of the point |  |
| level |  |
|  |  |
|  |  |

## Berechnungen für die Rechtecke -

Die oben beschriebenen Berechnungen sind für die Teilmengen bis zu wiederholen. Dabei dürfen sowohl für die Bestimmungen der statistischen Maßzahlen, als auch bei der Berechnung der nächstliegenden Adressen mit Abstand und Kurswinkel, nur Daten aus dem jeweiligen Teilmengen verwendet werden.

## Berechnung für alle Stadtbezirke

# Datei mit interessanten Spots

Die Datei enthält interessante Orte in Berlin (Point of Interest). Diese sollen verwendet werden, um umfangreichere Berechnungen im Vergleichsprogramm zu ermöglichen, und damit auch kleine Abweichungen zwischen den verwendenden Technologien aufzuzeigen.

Zu den ausgewählten Orten gehören zum Beispiel:

* Sehenswürdigkeiten
* Denkmäler
* Rathäuser
* Schlösser
* Kirchen
* Bahnhöfe
* Krankenhäuser
* …

Diese Datei ist manuell erstellt, einige Positionen sind genau, andere aus der Google Karte übernommen. Ich werde diese Datei im Laufe der Zeit noch ergänzen, Stand 13.11.2021 enthält sie ungefähr 280 Datensätze.

## Aufbau der Spots- Datei

Die Datei ist im nationalen Zeichensatz gespeichert. Dezimalzahlen werden ohne zusätzliche Formatierungen mit einem "." als Dezimalzeichen gespeichert. Die Datensätze haben eine festen Datenlänge.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feld | Beschreibung | Typ | Länge | Ausrichtung |
| Name | Name des Point of Interest | string | 72 | Links |
| Category | Kategorie | string | 20 | Links |
| Latitude | Breitengrad der geografischen Position | double | 10 | Rechts |
| Longitude | Längengrad der geografischen Position | double | 10 | Rechts |

Zusätzlich werden die Daten im Speicher um folgende Daten ergänzt. Diese werden verwendet, um später die nächstliegende Adresse aus der Kataster- Datei zu speichern.

|  |  |  |
| --- | --- | --- |
| Feld | Beschreibung | Datentyp |
| City | Stadt | string |
| Street | Straße | string |
| StreetNumber | Hausnummer mit Ergänzung (a, b, …) | string |
| ZipCode | PLZ | string |
| urban\_unit | Stadtbezirk | string |
| District | Stadtteil | String |
| Distance | Abstand dieser Adressdaten vom Spot | double |
| Angle | Kurswinkel von dieser Adresse zum Spot | double |

## Vorverarbeitung

### Einlesen der Datei

Beim Einlesen werden die Daten in einen dynamischen Speicherbereich gelesen. Die Anzahl der Sätze darf nicht vorausgesetzt werden, die Gesamtzahl darf beim Einlesen nicht überschritten werden. Die Datenfelder sind entsprechend zu trimmen.

### Sortieren der Datei

Nach dem Einlesen soll die Datei nach den Namen der Spots sortiert werden. Dabei soll nicht zwischen Groß- und Kleinbuchstaben unterschieden werden.

#### Schreiben der Datei

Zur Kontrolle soll die Datei wieder geschrieben werden. Der Datei Name ist "output\_spots.txt", die Datei wird im nationalen Zeichensatz geschrieben, und für die Dezimalzahlen werden ohne Tausenderseparator und mit dem "." als Dezimalpunkt geschrieben.

|  |  |  |
| --- | --- | --- |
| Feld | Ausrichtung | Größe, Dezimalstellen |
| Name | Links | 70 |
| Category | Links | 20 |
| Latitude | Rechts | 10,6 |
| Longitude | Rechts | 10,6 |

# Komplexe Berechnungen

Die Beschreibung für die Teile ist noch in Vorbereitung

## Bestimme Adresse mit Stadtbezirk / Stadtteil zu jedem Spot