Dateien analysieren

- `ana` erstellt Statistiken der Verteilungen von Byte-Folgen in Dateien
- die erste Version des Programms analysiert nur Byte-Folgen der Länge eins
- also die die Häufigkeiten der einzelnen Bytes
- das Programm wird später für beliebige Byte-Folgen fester Länge erweitert

- die Funktion `main` wertet Argumente der Kommandozeile aus
- dann liest `main` die Standard-Eingabe komplett und wertet sie aus
- zum Schluss schreibt `main` die resultierende Statistik in die Standard-Ausgabe

```
@Def(file: ana.cpp)
    @put(main prereqs);
    int main(
        int argc, const char *argv[]
    ) {
        @Put(parse args);
        @put(read input);
        @put(write table);
    }
@End(file: ana.cpp)
```

Datenstruktur für Statistik

- für jedes gelesene Byte wird gezählt, wie häufig der Byte-Wert in der gelesenen Eingabe vorkommt

- für jedes Byte wird ein eigener Zähler benutzt
- später wird die `Collection` umdefinieren
- daher ist die Definition in einem eigenen globalen Fragment gekapselt

```
@Def(def collection)
    using Collection =
        std::map<char, int>;
@End(def collection)
```

```
@def(main prereqs)
    #include <map>
    @Put(def collection);
    Collection collection;
@end(main prereqs)
```

- es gibt eine globale Variable, welche die Häufigkeiten enthält

- benötigt Standard Ein- und Ausgabe

```
@add(main prereqs)
    #include <iostream>
@end(main prereqs)
```

```
@def(read input)
    @Put(init state);
```

- jedes gelesene Zeichen wird in die Statistik integriert

```
        char ch;
        while (std::cin.get(ch)) {
            @Put(add to collection);
        }
    @end(read input)
```

- später benötigt das Programm einen Ort, um den aktuellen Schlüssel zu initialisieren
- das passende Fragment wird schon vorab definiert
- aber nicht gefüllt

- fügt Zeichen in die Statistik ein
- eigenes globales Fragment, um es später zu ersetzen

```
@Def(add to collection)
    ++collection[ch];
@End(add to collection)
```

```
@def(write table)
    for (const auto &e : collection) {
        @Put(write key);
        std::cout << "\t" <<
            e.second << "\n";
    }
@end(write table)
```

- jeder Eintrag der Statistik wird ausgegeben

- druckbare Zeichen werden direkt ausgegeben
- andere Bytes werden escaped
- so können Unix-Tools die Datei in Spalten aufteilen

```
@add(main prereqs)
    #include <cctype>
    void write_byte(char b) {
        if (isprint(b) &&
            b != '%' && b > ' '
        ) {
            std::cout << b;
        } else {
            @put(write escaped);
        }
    }
@end(main prereqs)
```

```
@def(write escaped)
    static const char digits[] {
        "0123456789abcdef"
    };
    std::cout << '%' <<
        digits[(b >> 4) & 0xf] <<
        digits[b & 0xf];
@end(write escaped)
```

- andere Bytes werden mit dem Präfix `%` als zwei hexadezimale Ziffern ausgegeben
- auch das Zeichen `%` muss so kodiert werden

- gibt den Schlüssel aus
- eigenes Fragment, da sich der Typ des Schlüssels später ändert

```
@Def(write key)
    write_byte(e.first);
@End(write key)
```