Dokumente generieren

Dokumente generieren

- erzeugt Dokumente mit vorgegebener Sequenz-Häufigkeit
- liest die Häufigkeitsverteilung über Standard-Eingabe

- `main` Funktion liest Häufigkeit
- und generiert passende Zeichen

```
@Def(file: gen.cpp)
    @put(main prereqs);
    int main() {
        @put(read receipt);
        @put(loop);
    }
@End(file: gen.cpp)
```

Zufällige Zeichen generieren

Zufällige Zeichen generieren

- generiert zufällige Zeichen

- `next` generiert das nächste Zeichen
- wirft Exception, wenn kein Zeichen generiert werden konnte

```
@def(main prereqs)
    @put(next prereqs);
    class No_Next { };
    inline char next() {
        @put(next);
        throw No_Next { };
    }
@end(main prereqs)
```

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

- benötigt `std::cin`, `std::cout`, etc.

- initialisiere den Zustand
- und gib die generierten Zeichen aus
- wenn kein Zeichen generiert werden kann, initialisiere den Zustand neu

```
@def(loop)
    @mul(initialise);
    for (;;) {
        try {
            std::cout << next();
        } catch (const No_Next &) {
            @mul(initialise);
        }
    }
@end(loop)
```

```
@inc(prefix.md)
```

- definiert Klasse für Byte-Arrays einheitlicher Länge
- die als Schlüssel von `std::map` verwendet werden

- integriert die Definition von `Prefix` in das Programm
- wird ebenfalls im erweiterten Analysator verwendet

```
@def(collection prereqs)
    @Mul(prefix)
@end(collection prereqs)
```

```
@add(collection prereqs)
    #include <map>
@end(collection prereqs)
```

- benötigt `std::map`

- `Entry` zählt wie häufig ein Zeichen `ch` nach einem Präfix vorkommt

```
@def(list prereqs)
    struct Entry {
        const char ch;
        const int count;
        Entry (char c, int v):
            ch { c }, count { v }
        { }
    };
@end(list prereqs)
```

```
@add(collection prereqs)
    @put(list prereqs);
    #include <vector>
    class List {
        private:
            std::vector<Entry> entries_;
            int sum_ { 0 };
        public:
            @put(list publics);
    };
@end(collection prereqs)
```

- Liste von `Entry`s
- zusätzlich wird die Gesamtsumme vorgehalten

- der Generator verwendet eine Abbildung von Präfixen auf Listen

```
@def(next prereqs)
    @put(collection prereqs);
    using Collection =
        std::map<Prefix, List>;
    Collection collection;
@end(next prereqs)
```

```
@def(list publics)
    void add(char ch, int count) {
        entries_.emplace_back(
            ch, count
        );
        sum_ += count;
    }
@end(list publics)
```

- fügt ein neues Zeichen zur Liste hinzu
- und passt die Gesamtsumme an

- liefert ein zufälliges Zeichen
- wenn keine Einträge hinterlegt sind, wird eine Exception generiert

```
@add(list publics)
    class No_Entries { };
    char next() const {
        if (sum_ > 0) {
            @put(next ch);
        }
        throw No_Entries { };


    }
```

```
@end(list publics)
```

```
@add(list prereqs)
    #include <random>
    std::mt19937 rng_ {
        std::random_device{ }()
    };
@end(list prereqs)
```

- initialisiert einen Zufallsgenerator (Mersenne-Twister)

- ermittelt eine Zufallszahl zwischen `0` und `sum_ - 1`

```
@def(next ch)
    auto dist {
        std::uniform_int_distribution<
            std::mt19937::result_type
        >(
            0, sum_ - 1
        ) };
    int result = dist(rng_);
@end(next ch)
```

```
@add(next ch)
    for (const auto &i : entries_) {
        if (result < i.count) {
            return i.ch;
        }
        result -= i.count;
    }
@end(next ch)
```

- wählt Zeichen anhand der Zufallszahl

- der Zustand ist ein Präfix mit den letzten ausgegebenen Zeichen

```
@add(next prereqs)
    Prefix state;
@end(next prereqs)
```

```
@def(initialise)
    init(state);
@end(initialise)
```

- initialsiert den Zustand auf Null-Bytes

- ermittelt das nächste zufällige Zeichen

```
@def(next)
    try {
        char ch {
            collection[state].next()
        };
        push(state, ch);
        return ch;
    } catch (const List::No_Entries &) {
    }
@end(next)
```

Rezept einlesen

Rezept einlesen

- liest Häufigkeitsverteilung von Standard-Eingabe

- wandelt Escape-Sequenzen in Schlüsseln in die passenden Bytes um

```
@add(main prereqs)
    @put(normalize prereqs);
    std::string normalize(
        const std::string &key
    ) {
        std::string result;
        unsigned i { 0 };
        for (; i < key.size(); ++i) {
            @put(normalize char);
        }
        return result;
    }
@end(main prereqs)
```

```
@def(normalize char)
    if (key[i] == '%') {
        @put(unescape);
        i += 2;
    } else {
        result += key[i];
    }
@end(normalize char)
```

- Escape-Sequenzen beginnen mit %
- alles andere wird direkt kopiert

- wandelt hexadezimale Ziffer in numerischen Wert um

```
@def(normalize prereqs)
    int hex_digit(char ch) {
        if (ch >= '0' && ch <= '9') {
            return ch - '0';
        } else if (
            ch >= 'a' && ch <= 'f'
        ) {
            return ch - 'a' + 10;
        }
        std::cerr << "invalid digit\n";
        return 0;
    }
@end(normalize prereqs)
```

- Escape-Sequenzen bestehen aus

```
@def(unescape)
    result += static_cast<char>(
        (hex_digit(key[i + 1]) << 4) +
            hex_digit(key[i + 2])
    );
@end(unescape)
```

zwei hexadezimalen Ziffern

- Rezepte bestehen aus einer Liste von Schlüssel/Anzahl Paaren
- der erste Schlüssel bestimmt wie lang die Präfixe sind

```
@def(read receipt)
    bool first { true };
    Prefix k;
    for (;;) {
        @put(read key);
        @put(read count);
        if (first) {
            @put(setup length);
            first = false;
        }
        @put(add entry);
    }
@end(read receipt)
```

```
@def(read key)
    std::string key;
    std::cin >> key;
    if (! std::cin) { break; }
    key = normalize(key);
@end(read key)
```

- liest Schlüssel
- und expandiert Escape-Sequenzen

- liest die Anzahl

```
@def(read count)
    int count;
    std::cin >> count;
    if (! std::cin) { break; }
@end(read count)
```

```
@def(setup length)
    prefix_length = key.size() - 1;
    init(k);
@end(setup length)
```

- Länge ist eins weniger als die Schlüssel-Länge

- initialisiert k mit dem Schlüssel ohne dem letzten Byte
- fügt Anzahl für das letzte Byte in die Abbildung ein

```
@def(add entry)
    for (unsigned i { 0 };
        i + 1 < key.size(); ++i
    ) {
        push(k, key[i]);
    }
    collection[k].add(
        key.back(), count
    );
@end(add entry)
```