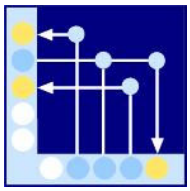
Hochschule Aalen
Fakultät Elektronik und Informatik
Studienbereich Informatik



Compilerbau-Praktikum

Lehrveranstaltung im Sommersemester 2021
Prof. Dr. habil. Christian Heinlein

1. Aufgabenblatt (22. März 2021)

Aufgabe 1: C++-Bibliothek *lib*$_{\text{CH}}$: Sequenzen

Arbeiten Sie sich anhand der Dokumentation zur Bibliothek *lib*$_{\text{CH}}$ in das Thema Sequenzen ein! Wichtig sind insbesondere die Teilabschnitte 5.1.1 bis 5.1.7 sowie 5.1.12 und 5.1.13.

✎

```
#include <iostream>
using namespace std;

#include "seq.ch"
using namespace CH;
```

✐

Implementieren Sie mit dem erworbenen Wissen u. a. folgendes:

a) Für eine Sequenz `s` mit einem beliebigen Elementtyp `T` gibt `print(s)` die Elemente von `s` nacheinander auf dem Standardausgabestrom `std::cout` aus. Zwischen den Elementen wird jeweils ein Leerzeichen und nach dem letzten Element ein Zeilentrenner ausgegeben.

✎

```
template <typename T>
void print (const seq<T>& s) {
    const char* sep = "";
    for (T x : s) {
        cout << sep << x;
        sep = " ";
    }
    cout << endl;
}
```

✐

b) Für eine ganze Zahl `n` liefert `nat(n)` eine Sequenz mit den natürlichen Zahlen von 1 bis `n`.

✎

```
seq<int> nat (int n) {
    seq<int> r;
    for (int i = 1; i <= n; i++) r += i;
    return r;
}
```

✐

c) Für eine Sequenz `s` mit einem beliebigen Elementtyp `T` und eine ganze Zahl `n` liefert `getA(s, n)` bzw. `getZ(s, n)` eine neue Sequenz mit den ersten bzw. letzten `n` Elementen von `s`; `cutA(s, n)` bzw. `cutZ(s, n)` liefert eine neue Sequenz mit allen Elementen von `s` außer den ersten bzw. letzten `n`.
Was liefern diese Funktionen, wenn `n` kleiner als 0 oder größer als die Länge von `s` ist?

✎

```
template <typename T>
seq<T> getA (const seq<T>& s, int n) {
    return s(A|n);
}

template <typename T>
seq<T> getZ (const seq<T>& s, int n) {
    return s(n|Z);
}

template <typename T>
seq<T> cutA (const seq<T>& s, int n) {
    return s(A+n|Z);
}

template <typename T>
seq<T> cutZ (const seq<T>& s, int n) {
    return s(A|Z+n);
}
```

- Für `n` kleiner als 0 liefern `getA(s, n)` und `getZ(s, n)` jeweils eine leere Sequenz, während `cutA(s, n)` und `cutZ(s, n)` jeweils eine Sequenz mit allen Elementen von `s` liefern.
- Für `n` größer als die Länge von `s` liefern `getA(s, n)` und `getZ(s, n)` jeweils eine Sequenz mit allen Elementen von `s`, während `cutA(s, n)` und `cutZ(s, n)` jeweils eine leere Sequenz liefern.

✐

d) Für eine Sequenz `s` mit einem beliebigen Elementtyp `T` und einen Wert `x` mit Typ `T` fügt `pushA(s, x)` bzw. `pushZ(s, x)` den Wert `x` am Anfang bzw. am Ende der Sequenz `s` hinzu; `popA(s)` bzw. `popZ(s)` entfernt den ersten bzw. letzten Wert der Sequenz `s` und liefert ihn zurück.
Die Sequenz(variable) `s` wird von allen vier Funktionen direkt verändert, d. h. sie muss als Referenz mit Typ `seq<T>&` übergeben werden.
Mit diesen Funktionen kann eine Sequenz entweder als Stack (mit `pushZ` und `popZ` oder mit `pushA` und `popA`) oder als Queue (mit `pushZ` und `popA` oder mit `pushA` und `popZ`) oder als Double-Ended-Queue (mit allen vier Funktionen) verwendet werden.
Was machen `popA` und `popZ`, wenn sie auf eine leere Sequenz angewandt werden?

✎

```
template <typename T>
void pushA (seq<T>& s, const T& x) {
    s = s(A, x);
}

template <typename T>
void pushZ (seq<T>& s, const T& x) {
    s = s(Z, x);
}

template <typename T>
T popA (seq<T>& s) {
    T x = s[A];
    s = s(A+1|Z);
    return x;
}

template <typename T>
T popZ (seq<T>& s) {
    T x = s[Z];
    s = s(A|Z+1);
    return x;
}
```

- Für eine leere Sequenz liefern `popA` und `popZ` den nil-Wert des Typs `T` und lassen die Sequenz unverändert leer.

✎

e) Für Zeichenketten `s` und `t` mit Typ `str` liefert `split(s, t)` eine Sequenz von Zeichenketten, die nacheinander die maximal langen Teilketten von `s` enthält, die keine Zeichen aus `t` enthalten, d. h. die Zeichenkette `s` wird an Folgen von Zeichen aus `t` aufgeteilt.
Zum Beispiel liefert `split("XabcYZdeXXXfghi", "XYZ")` eine Sequenz mit den Zeichenketten `"abc"`, `"de"` und `"fghi"`.

✎

```
seq<str> split (const str& s, const str& t) {
    // Resultatsequenz.
    seq<str> r;

    // Anfangs- und Endposition der nächsten Teilkette,
    // die zur Resultatsequenz r gehört.
    posA p = A, q = A;

    // Wert von flag (siehe unten) aus dem vorigen Schleifendurchlauf.
    bool prev = true;

    // Sequenz s zeichenweise durchlaufen.
    for (char c : s) {
        // Überprüfen, ob das aktuelle Zeichen c in t enthalten ist.
        bool flag = t(searchA(eq(c)));
```

```
            // Wenn c in t enthalten ist und das vorige Zeichen nicht
            // oder umgekehrt:
            if (flag != prev) {
                // Wenn c in t enthalten ist,
                // gehört die Teilkette von p bis q zur Resultatsequenz r.
                // Andernfalls beginnt bei q die nächste derartige Teilkette.
                if (flag) r += s(p|q);
                else p = q;

                // Wert von flag für den nächsten Durchlauf speichern.
                prev = flag;
            }

            // Aktuelle Position q weitersetzen.
            q++;
        }

        // Ggf. gehört noch die aktuelle Teilkette von p bis q
        // zur Resultatsequenz r.
        if (!prev) r += s(p|q);

        return r;
    }
```

✎

f) Für Zeichenketten `s`, `p` und `r` mit Typ `str` und einen Booleschen Wert `a` liefert `subst(s, p, r, a)` eine Kopie der Zeichenkette `s`, in der das erste (wenn `a` gleich `false` ist) bzw. alle (wenn `a` gleich `true` ist) Vorkommen der Zeichenkette `p` durch die Zeichenkette `r` ersetzt sind.
Der Parameter `a` ist optional; wenn er nicht angegeben wird, ist er gleich `false`.

✎

```
    str subst (str s, const str& p, const str& r, bool a = false) {
        // Länge der Zeichenkette p.
        int n = *p;

        // Für jede mögliche Anfangsposition i eines Vorkommens von p in s:
        for (posA i = A, e = A + *s - n; i < e; i++) {
            // Wenn p tatsächlich an dieser Position i in s vorkommt:
            if (s(i|n) == p) {
                // Dieses Vorkommen von p in s durch r ersetzen.
                s = s(i|n, r);

                // Schleife beenden,
                // wenn nur das erste Vorkommen ersetzt werden soll.
                if (!a) break;
```

```
                // Position i auf das Ende von r in s weitersetzen
                // (minus 1 wegen i++ im Schleifenkopf).
                i += *r - 1;
            }
        }

        return s;
    }
```

Einfaches Testprogramm

```
int main () {
    print(nat(5));                                  // 1 2 3 4 5

    print(getA(nat(10), 5));                        // 1 2 3 4 5
    print(getZ(nat(10), 5));                        // 6 7 8 9 10
    print(cutA(nat(10), 5));                        // 6 7 8 9 10
    print(cutZ(nat(10), 5));                        // 1 2 3 4 5

    seq<int> s1;
    pushA(s1, 1);
    pushZ(s1, 2);
    int x = popA(s1);
    int y = popZ(s1);
    cout << x << " " << y << endl;                  // 1 2

    str s2 = "XabcYZdeXXXfghi";
    print(split(s2, "XYZ"));                        // abc de fghi

    cout << subst(s2, "X", "<>") << endl;           // <>abcYZdeXXXfghi
    cout << subst(s2, "X", "<>", true) << endl; // <>abcYZde<><><>fghi
    cout << subst(s2, "X", "XX", true) << endl; // XXabcYZdeXXXXXXfghi
}
```