Objektorientiertes Programmieren - SWB2 & TIB2
Hausaufgabe 2

Aufgabe 1: Ein einfaches Pac-Man-Spiel - Teil 2

In Labor 1 haben Sie ein Programm geschrieben, um ein Labyrinth erstellen zu können. Unser Pac-Man-Spiel soll objektorientiert sein. Daher soll ein Labyrinth ein Objekt der Klasse `Labyrinth` sein, die Sie in dieser Aufgabe schreiben.

Schreiben Sie die Klassendeklaration in eine hpp-Datei und die Definitionen in eine cpp-Datei.

Die Klasse `Labyrinth` hat folgende Eigenschaften:

a) Die Instanzvariablen `labZeilen` und `labSpalten` geben die Größe des Labyrinths an.
b) Die Instanzvariable `labAnzGeister` merkt sich, wie viele Geister im Labyrinth unterwegs sind.
c) Die Instanzvariable `muenzen` gibt an, wie viele Münzen im Labyrinth ausgelegt sind. Auf jedes Wegstelle im Labyrinth (Leerzeichen) soll später eine Münze (repräsentiert durch das Zeichen ':') ausgelegt werden.
d) Die Instanzvariable `lab` ist wie zuvor ein 2-dimensionales **`char`**-Array. Da die Dimensionen des Arrays Konstanten sein müssen, benötigen wir auch weiterhin die globalen Konstanten `kZeilen` und `kSpalten`. Beachten Sie, dass wie zuvor das Array zwei zusätzliche Spalten hat, um das Zeilenendezeichen ('\n') und das NULL-Zeichen ('\0') zu speichern.
e) Die Funktionen `initialisieren()`, `drucken()` und `erzeugen()` aus Labor 1 werden zu Instanzmethoden.
f) Der Konstruktor gibt den Instanzvariablen `labZeilen`, `labSpalten` und `labAnzGeister` sinnvolle Werte und setzt `muenzen` auf 0. Dann ruft er die Methode `initialisieren()` auf.
g) Die Instanzmethoden `getZeilen()`, `getSpalten()`, `getAnzGeister` und `getMuenzen()` liefern die entsprechenden Instanzvariablen zurück.
h) Die Instanzmethode `legeMuenzen()` verteilt auf den Wegen im Labyrinth (d.h. auf allen freien Chars (' ')) Münzen (':') und speichert die Anzahl der gelegten Münzen in der Instanzvariable `muenzen`.
i) Die Instanzmethode `zeichneChar(char c, Position pos)` schreibt an der Position `pos` im Labyrinth einen Character `c`.
Die Klasse `Position` ist in cpp- und hpp-Dateien vorgegeben, die am Ende dieser Aufgabe zu finden sind. Ein Objekt der Klasse `Position` speichert eine Zeilenkoordinate (`posy`) und eine Spaltenkoordinate (`posx`). Zusätzlich speichert die Position auch noch eine Laufrichtung (`RECHTS`, `LINKS`, `OBEN`, `UNTEN`), die wir erst später benötigen.
j) Die Instanzmethode `zeichneChar(char c, Position posalt, Position posneu)` schreibt ein Zeichen in das Labyrinth an Position `posneu` und schreibt ein Leerzeichen an Position `posalt`. Mit dieser Methode können wir Zeichen durch das Labyrinth laufen lassen. Nutzen Sie die Methode `zeichneChar(char, Position)` von zuvor.
k) Die Instanzmethode **`char`** `getZeichenAnPos(Position pos)` gibt das Zeichen zurück, das an der Position `pos` im Labyrinth gespeichert ist.
l) Die Instanzmethode **`bool`** `istMuenzeAnPos(Position pos)` prüft, ob an der gegebenen Position eine Münze (':') liegt.
m) Die Instanzmethoden `exportDatei` und `importDatei` erlauben, Labyrinthe als Text-

dateien abzuspeichern und wieder zu lesen. Die Details zu den Funktionen besprechen wir in der Vorlesung zu Filestreams.

```
// Labyrinth als Textdatei speichern
void Labyrinth::exportDatei(char * dateiname) {
    ofstream datei(dateiname);
    if (!datei) {
        cerr << "Kann Datei nicht oeffnen" << endl;
    }
    for (int i = 0; i < kZeilen; i++) {
        datei << lab[i];
    }
    datei.close();
}

// Labyrinth als Textdatei einlesen
void Labyrinth::importDatei(char * dateiname) {
    ifstream datei(dateiname);
    if (!datei) {
        cerr << "Kann Datei nicht oeffnen" << endl;
    }
    for (int i = 0; i < kZeilen; i++) {
        datei.getline(lab[i], kSpalten + 2);
        lab[i][kSpalten] = '\n';
        lab[i][kSpalten + 1] = '\0';
    }
    datei.close();
}
```

Testen Sie Ihre Klasse mit dem folgenden Hauptprogramm.

```
// Main für Hausaufgabe 2
#include "Labyrinth.hpp"

int main() {
    Labyrinth lab;
    lab.drucken();
    lab.erzeugen();
    lab.drucken();
    lab.exportDatei("lab.txt");
    Labyrinth lab2;
    lab2.importDatei("lab.txt");
    lab2.drucken();
    lab2.legeMuenzen();
    lab2.drucken();
}
```

Die Dateien für die Klasse `Position`:

```cpp
// Datei Position.hpp
#pragma once

// Vorwärtsdeklaration der Klasse Labyrinth
class Labyrinth;

// Die Laufrichtung (Orientierung) als Aufzählungstypen
enum Richtung { RECHTS, LINKS, OBEN, UNTEN };

// Klasse Position
// Speichert die Position (x,y) und Laufrichtung eines Objektes
// im Labyrinth
// Als struct realisiert, damit später direkt auf posx und posy
// zugegriffen werden kann.
struct Position {
    // Position in den Spalten
    int posx;
    // Position in den Zeilen
    int posy;
    // Richtung, in die jemand orientiert ist
    Richtung r;
    // Konstruktor mit posx (Spalten) und posy (Zeilen)
    Position(int = 0, int = 0);
    // Schritt nach vorne in Richtung r um einen Schritt
    // *this wird entsprechend geändert
    // Der int = 0 gibt an, dass die Mauern berücksichtigt werden.
    // Wenn int != 0, dann kann auch durch Mauern gelaufen werden.
    // Dies ist für das Erstellen von Labyrinthen notwendig.
    Position & schritt(Labyrinth &, int = 0);
    // Zwei Positionen vergleichen,
    // liefert true, wenn x und y gleich
    bool istGleichZu(Position &);
};
```

```cpp
// Datei Position.cpp
#include "Position.hpp"
#include "Labyrinth.hpp"

// Hilfsfunktion max
int max(int x, int y) {
    return (x <= y) ? y : x;
}

// Hilfsfunktion min
int min(int x, int y) {
    return (x <= y) ? x : y;
}
```

```
// Konstruktor
Position::Position(int x, int y) {
    posx = x;
    posy = y;
}

// Schritt nach vorne in Richtung r um einen Schritt
// Es kann aber nicht in Mauern hineingelaufen werden
// *this wird entsprechend geändert
// Der int mode = 0 gibt an, dass die Mauern berücksichtigt werden.
// Wenn int != 0, dann kann auch durch Mauern gelaufen werden.
// Dies ist für das Erstellen von Labyrinthen notwendig.
Position & Position::schritt(Labyrinth & lab, int mode) {
    Position tmp = *this;
    switch (r) {
        // oben
    case Richtung::OBEN: tmp.posy = max(1, posy - 1); break;
        // links
    case Richtung::LINKS: tmp.posx = max(1, posx - 1); break;
        // rechts
    case Richtung::RECHTS: tmp.posx =
                               min(lab.getSpalten() - 2, posx + 1);
        break;
        // unten
    case Richtung::UNTEN: tmp.posy =
                               min(lab.getZeilen() - 2, posy + 1);
        break;
    }
    if (mode != 0 || lab.getZeichenAnPos(tmp) != MAUER) {
        *this = tmp;
    }
    return *this;
}

// Zwei Positionen vergleichen,
// liefert true, wenn x und y gleich
bool Position::istGleichZu(Position & p) {
    return (posx == p.posx && posy == p.posy);
}
```

Aufgabe 2: Eine eigene String-Klasse - Teil 1

Schreiben Sie eine Klasse `MyString`, um einen bequemen Ersatz für C-Strings zu haben. Deklarieren und definieren Sie die Klasse wie folgt. Teilen Sie dabei Deklaration und Definition in eigene Dateien auf.

a) Die Instanzvariable `strPtr` zeigt auf ein dynamisches **`char`**-Array, in dem der String als C-String gespeichert wird.
b) Die Instanzvariablen `strSize` und `strCapacity` vom Typ **`unsigned int`** geben an,

wie lang der String ist (Größe, ohne abschließendes Null-Zeichen) bzw. wie viele Zeichen derzeit insgesamt in das dynamische **char**-Array geschrieben werden können (Kapazität, ohne abschließendes Null-Zeichen).

c) Der Standardkonstruktor legt einen leeren String an.

d) Ein Konvertierkonstruktor konvertiert einen C-String in einen `MyString`.

e) Der Kopierkonstruktor legt eine tiefe Kopie an.

f) Der Destruktor sorgt dafür, dass kein Speicherleck entsteht, d.h. dass rechtzeitig mit **new** reservierter Speicher auch wieder freigegeben wird.

g) Die Instanzfunktion **void** `reserve(`**unsigned int** `c)` vergrößert den Speicherplatz des dynamischen Arrays auf den übergebenen Wert `c`. Wenn die aktuelle Kapazität größer als der Wert `c` ist, wird nichts getan.

h) Die Instanzfunktion `MyString & append(MyString & str)` hängt `str` an das this-Objekt hinten an (also jeweils den gespeicherten C-String).

i) Die Instanzfunktion `MyString & assign(MyString & str)` weist den String `str` dem this-Objekt zu. Ein evtl. vorhandener String wird überschrieben.

j) Die Instanzfunktion `c_str` liefert einen Zeiger auf den im this-Objekt gespeicherten C-String zurück. Sorgen Sie dafür, dass über den Zeiger keine Manipulation an Ihrem String vorgenommen werden kann.

k) Mit `size()` erhalten Sie die Größe und mit `capacity()` die Kapazität eines `MyString`.

l) Die Instanzfunktion `clear()` setzt den String auf einen leeren String.

m) Die Instanzfunktion `empty()` prüft, ob ein String leer ist.

n) Mit `at(i)` kann man auf das (i-1)-te Zeichen eines Strings zugreifen und dieses auch verändern. Sollte `at(i)` versuchen, außerhalb des Strings zuzugreifen, liefert `at(i)` das Element `at(0)` zurück.

Zum Kopieren von C-Strings können Sie die Funktion `strncpy` aus der Bibliothek `<cstring>` nutzen. In diesem Fall empfiehlt es sich aber bei Visual Studio die Präprozessordirektive `_CRT_SECURE_NO_WARNINGS` in die Konfiguration des Projektes mit aufzunehmen. Alternativ können Sie die sicherere Funktion `strncpy_s` nutzen. Diese ist jedoch spezifisch für Microsofts Compiler.

Testen Sie Ihre Klasse mit dem folgenden Hauptprogramm:

```
#include <iostream>
#include "MyString.hpp"
using namespace std;

int main() {
    MyString s1;
    MyString s2("Hochschule Esslingen");
    cout << s1.c_str() << endl;
    cout << s2.c_str() << endl;
    s1.assign(s2);
    s2.assign(MyString(" - University of Applied Sciences"));
    s1.append(s2);
    cout << s1.c_str() << endl;
    cout << s2.c_str() << endl;
    MyString s3(s1);
    s1.at(21) = 'X';
```

```
    s1.at(2000) = 'O';
    cout << s1.c_str() << endl;
    cout << s2.c_str() << endl;
    if (!s3.empty()) {
        cout << "Laenge von s3:     " << s3.size() << endl;
        cout << "Kapazitaet von s3: " << s3.capacity() << endl;
        cout << s3.c_str() << endl;
    }
    s3.clear();
    cout << "Laenge von s3:     " << s3.size() << endl;
    cout << "Kapazitaet von s3: " << s3.capacity() << endl;
    cout << s3.c_str() << endl;
    return 0;
}
```