Aufgabe Programmierung

Scheinkriterien: Für den Erhalt des ECG-Scheines ist es notwendig, insgesamt mindestens 37 Punkte zu erreichen. Zusätzlich gilt, dass pro Theorieblatt mindestens 1 Punkt und pro Praxisblatt mindestens 2 Punkte erreicht werden müssen!

1 Praxis

1.1 Bibliothek mit Baumdatenstruktur (5 Pt)

1.1.1 Grundgerüst der Baumdatenstruktur (2 Pt)

Erstellen Sie zwei neue Dateien `tree.h` und `tree.cxx` und implementieren Sie eine Baumdatenstruktur wie folgt.

Die Knotenklasse `node` soll einen Namen vom Typ `std::string` speichern können und folgende Methoden bereitstellen (Typen der Argumente und Rückgabewerte sind bewusst weggelassen und müssen erschlossen werden)

- Konstruktor mit einem Argument vom Typ `const std::string&`, das den Knotenname initialisiert
- Destruktor zum Löschen aller Kindknoten mit dem `delete`-Operator. Deklarieren Sie den Destruktor als virtuelle Methode.
- `get_name() const` ... gibt den Namen des Knotens zurück
- `set_name(new_name)` ... setzt den Namen des Knotens auf einen neuen Namen
- `get_nr_children() const` ... gibt die Anzahl der Kindknoten an
- `get_child(i) const` ... gibt einen Zeiger auf den i-ten Kindknoten zurück
- `add_child(node)` ... fügt am Ende einen neuen Kindnoten hinzu

Nutzen Sie zum Speichern der Kindknotenzeiger die Template Klasse `std::vector` der Standard Template Library, die im Header `<vector>` deklariert ist.

1.1.2 Programmstruktur (2 Pt)

Erstellen Sie eine dritte Datei `main.cxx`. Binden Sie `tree.h` mit einem entsprechenden `#include`-Befehl ein

- Implementieren Sie eine `main`-Funktion, die einen Baum mit einem Wurzelknoten namens `"root"` und zwei Kindern namens `"left child"` und `"right child"` erzeugt und danach den ganzen Baum mit dem `delete`-Operator angewendet auf den Wurzelknoten wieder löscht.

- Erstellen Sie zwei Visual Studio Projekte - ein Projekt namens `tree`, das aus `tree.cxx` eine statische Bibliothek erzeugt und ein Projekt namens `tree_test`, das aus `main.cxx` eine Anwendung erstellt. Setzen Sie in der Solution das Projekt `tree_test` als abhängig vom Projekt `tree`.

1.1.3 Debugging (1 Pt)

Setzen Sie im Destruktor der Knotenklasse einen Break-Point und starten Sie die Anwendung in der Debug-Konfiguration im Debug-Modus. Beobachten Sie, wie der Destruktor rekursiv aufgerufen wird. Erweitern Sie den Destruktor so, dass folgende Ausgabe entsteht (nicht vergessen, `<iostream>` zu inkludieren):

```
enter ~node() of "root"
enter ~node() of "left child"
leave ~node() of "left child"
enter ~node() of "right child"
leave ~node() of "right child"
leave ~node() of "root"
```

1.2 Rekursive Traversierung (5 Pt)

1.2.1 Globale Knotenzählung (1 Pt)

Erweitern Sie die Knotenklasse um eine statische Variable `node_id`, die eine globale Knotennummer mitzählt. Initialisieren Sie diese in `tree.cxx` auf `0` und zählen Sie sie im Knotenkonstruktor um eins hoch.

Geben Sie dem Namensparameter im Knotenkonstruktor einen leeren String als Defaultparameterwert und setzen Sie im Konstruktor den Knotennamen auf "`node_`<*node_id*>", falls kein Knotenname angegeben wurde. Dabei sollen die automatisch erzeugten Knotennamen mit `"node_1"` beginnen. Nutzen Sie zur Umwandlung der Knotennummer in einen String, die `std::stringstream`-Klasse aus dem Header `<sstream>`, die wie folgt verwendet wird:

```
std::stringstream str_sm;
str_sm << node_id;
std::string node_id_str = str_sm.str();
```

1.2.2 Rekursive Baumerstellung (2 Pt)

Implementieren Sie in `tree.cxx` eine Funktion `create_complete_tree(nr_child_nodes, tree_depth)`, die rekursiv einen Baum erstellt, bei dem alle Knoten bis auf die Blattknoten genau `nr_child_nodes` Kindknoten haben und bei dem die Pfade von der Wurzel bis zu den Blättern genau `tree_depth` Knoten enthalten (dabei ist der Wurzelknoten mitzuzählen). Deklarieren Sie die Methode in `tree.h` als extern und rufen Sie die Methode mit den Parameterwerten `(2,4)` in der `main`-Funktion auf.

1.2.3 Stream-Ausgabe (2 Pt)

Überladen Sie den <<-Operator für die Knotenklasse so, dass folgende Ausgabe erzeugt wird, wenn man den Wurzelknoten des von `create_complete_tree(2,4)` erzeugten Baumes ausgibt:

```
node_1
   node_2
      node_3
         node_4
         node_5
      node_6
         node_7
         node_8
   node_9
      node_10
         node_11
         node_12
      node_13
         node_14
         node_15
```

Nutzen Sie dazu eine weitere statische Variable, die die Anzahl der Einrückungen für die Textausgabe speichert. Eine mit `extern` versehene Deklaration des überladenen Operators soll wieder in `tree.h` erscheinen und in `tree.cxx` die Implementierung.

1.3 Zusatzaufgaben (maximal 5 Pt)

- Erstellen Sie einen Baum, der einen Zyklus enthält und implementieren Sie eine neue Traversierungsmethode zur Ausgabe ähnlich zum <<-Operator, die Zyklen detektiert, markiert ausgibt und eine unendliche Rekursion vermeidet. Erweitern Sie dazu die Knotenklasse um ein boolsches Flag, das speichert, ob ein Knoten schon traversiert wurde. Optional können Sie alle besuchten Knoten beim Traversieren in einem `std::vector` mitführen. (max +5 Pt)

- Erstellen Sie ein Visual Studio Projekt namens `tree_dll`, das aus `tree.cxx` eine dynamische Bibliothek erstellt. Dabei ist zu beachten, dass beim Erstellen der Dll die Deklarationen in `tree.h` mit `__declspec(dllexport)` gekennzeichnet werden müssen, beim Nutzen der Dll im Project `tree_test` jedoch stattdessen mit `__declspec(dllimport)`. Lesen Sie die Dokumentation zu `__declspec` und ändern Sie Ihre Deklarationen in `tree.h` so ab, dass der Header sowohl im statischen Bibliotheksprojekt `tree` wie auch im dynamischen Fall in `tree_dll` verwendet werden kann. Beispiellösungen hierfür finden sich in freien Bibliotheken wie `glew` oder `fltk`. (max +3 Pt)

- Erweitern Sie das `glut`-Projekt des in der Vorlesung besprochenen Tutorials um die Darstellung eines in der Übung entwickelten Baums. Dazu ist der `geometry_node` so zu erweitern, dass ein Baum aus der Übung als Geometrie gezeichnet werden kann. Um eine Namenskollision der Klasse `node` zu verhindern, betten Sie die in der Übung entwickelte Klasse in den Namensraum `tree` ein. Überlegen Sie sich ein geeignetes Layout zum graphischen Darstellen des Baumes. Nutzen Sie die Funktion `glutBitmapCharacter` zum Zeichnen von Text. (max +5 Pt)

2 Abgabe

2.1 Was

Die Lösung muss im Rechnerpool bzw. in den beiden Laborräumen E67, E69 lauffähig sein. Für die Abnahme setzen wir ein MS Visual Studio (2008) Projekt voraus. Stellen Sie also sicher, dass der Quellcode unter diesen Voraussetzungen kompiliert. Es wird ausschließlich Quellcode akzeptiert, der zuvor wie unter „Wie“ beschrieben abgegeben wurde.
Wenn Sie die entsprechenden Software auch zu Hause verwenden wollen, besteht für Informatikstudenten der TU Dresden die Möglichkeit diese kostenlos via MSDNAA zu beziehen:

`http://www.inf.tu-dresden.de/index.php?node_id=2129&ln=de`

.

2.2 Wie

Archivieren Sie alle Dateien Ihrer Lösung in eine ZIP-Datei mit dem Namen

`übung1_lab.zip` .

Die Abgabe erfolgt via OPAL. Dazu nutzen Sie die Praktikumsseite der entsprechenden Übung.
Über den Punkt „Abgabeordner“ laden Sie Ihre Ergebnisse als ZIP-Datei auf den OPAL Server. Dies können Sie bis zum Abgabetermin (siehe Seitenkopf) mehrfach vornehmen z.B. um Korrekturen an Ihrer Lösung durchzuführen.
Jeder Student muss die Lösung (seines 2er-Teams) via *seines* OPAL-Accounts hochladen !