Programmieren 2 **Übungszettel 4** **SS2010**

Aufgabe 1) Klassen

Übernehmen Sie Ihre Implementierung für Bit-Ketten (Übungszettel 2, Aufgabe 2) und wandeln Sie den dort definierten abstrakten Datentyp in eine Klasse um. Passen Sie Ihre Testroutinen aus Übungszettel 2, Aufgabe 2 an die Implementierung als Klasse an. Erstellen Sie dabei bitte Deklaration und Implementierung der Klasse in getrennten Dateien und führen Sie den Test von einer dritten Datei (z.B. main.cpp) aus. Überlegen Sie sich, welche Funktionen Sie als **friend** deklarieren möchten und welche Member der Klasse bleiben. Denken Sie bitte auch über Konstruktoren nach: Welche hätten Sie gerne, um mit der Klasse gut arbeiten zu können? Implementieren Sie diese und bauen Sie sie in den Test ein.
ACHTUNG: Die von Ihnen hier erstellte Klasse benötigen wir auch in den nachfolgenden Aufgaben. Sie sollten daher die Klasse so bauen, dass sie mit möglichst wenig Aufwand zu benutzen ist (ob dies so ist, kann man gut am Test-Code für die Klasse sehen).

Aufgabe 2) OOA/OOD

Für diese Aufgabe gilt ganz besonders: Lesen Sie bitte erst den Aufgabentext und fangen danach an daran zu arbeiten.
In dieser Aufgabe soll eine Textanalyse durchgeführt werden, wie wir es in der VL am Beispiel Projektmanagement kennengelernt haben. Bilden Sie hierzu bitte eine Zweiergruppe. Im ersten Schritt soll jede Person alleine die Textanalyse durchführen und ein Klassendiagramm entwerfen. Erzeugen Sie hier bitte noch keinen Code! Im zweiten Schritt setzen Sie sich bitte mit Ihrem Teampartner zusammen und erklären jeweils Ihrem Gegenüber Ihren Entwurf anhand des Diagrammes und der dafür verwendeten Textstellen. Diskutieren Sie beide Entwürfe kritisch und konsolidieren Sie danach die Entwürfe zu einem gemeinsamen Entwurf. Beachten Sie dabei bitte die folgenden Hinweise (Sie müssen das hinterher implementieren, daher ist es in Ihrem Interesse):

- Gestalten Sie die Klassen übersichtlich und kompakt, d.h. für ihre jeweilige Aufgabe zugeschnitten und keine Alleskönner-Klassen. Dies vereinfacht Veränderungen in späteren Übungen
- Verwenden Sie wenn möglich nur private Attribute in einer Klasse
- Überlegen Sie sich ggf. spezialisierte Konstruktoren, führen aber nur solche spezialisierten Konstruktoren auch im Klassendiagramm ein
- Vermeiden Sie eine zu hohe Anzahl an Klassen
- Vermeiden Sie möglichst zyklische Abhängigkeiten zwischen Klassen, die navigierbar sein müssen (wenn nicht, werden Sie spätestens in der Implementierung erleben, was Sie sich da angetan haben; im schlimmsten Fall bedeutet es ein komplettes Re-Design)
- Benutzen Sie Assoziationen anstelle von Attributen, wenn es um Klassenverweise geht. Bestimmen Sie auch Multiplizitäten und Richtungen.

Implementieren Sie nun das Code-Gerüst für Ihre Anwendung. Sie dürfen hier gerne Tool-Unterstützung (wie beispielsweise StarUML) benutzen, bedenken Sie aber, dass Sie den Code in jedem Fall von grässlichen Artefakten bereinigen müssen. Wenn Sie sich noch unsicher fühlen, schreiben Sie den Code besser per Hand; das übt. Führen Sie bitte – wie es guter Stil ist – getrennte Dateien für Deklaration und Implementierung einer Klasse ein.
Im Fall von Multiplizitäten mit * verzichten Sie bitte zunächst auf eine Ausprogrammierung, sondern vermerken in einem Kommentar, dass dort noch etwas getan werden muss.
Und hier ist nun die Beschreibung der zu entwerfenden Anwendung:

Anwendung: Digitalsimulator

Es soll ein einfacher Digitalsimulator entworfen werden, mit dem es möglich ist, für eine anzugebene Zeitspanne die Veränderungen von Signalen in einem Netzwerk von digitalen Funktionseinheiten im Zusammenspiel mit Eingaben auf offenen Signalen zu beobachten. Hierzu sind die folgenden Anforderungen zu berücksichtigen (ein Glossar am Ende erklärt Begrifflichkeiten, den Rest müssen Sie zwischen den Zeilen lesen oder nachfragen):

A1: Der Simulator besitzt eine Schnittstelle, über die die Simulation eine anzugebene Zeitspanne gestartet werden soll. Während der Simulation wird für jeden Zeitschritt der Simulationszyklus ausgeführt. Die Simulation ist zeit-diskret durchzuführen.

A2: Der Simulator verwaltet die Struktur der zu simulierenden Schaltung. Er bietet eine Schnittstelle, mit der Funktionseinheiten und deren Vernetzung aufgebaut werden können.

A3: Die Vernetzung ist so ausgelegt, dass beliebig viele Teilvektoren eines Ausgabe-Vektors einer Funktionseinheit an beliebig viele Teilvektoren eines Eingabe-Vektors einer Funktionseinheit (auch dieselbe Funktionseinheit ist möglich; Rückführungen sind erlaubt) angeschlossen werden können. Bedingung ist jedoch, dass

a) Die Breite des Teilvektors des Eingabe-Vektors ausreicht, um den Teilvektor eines Ausgabe-Vektors aufzunehmen
b) Immer komplette Teilvektoren anzuschließen sind und keine Stellen im Eingabe-Vektor auszuschließen sind: Vektoren werden grundsätzlich durch aufeinanderfolgende Bits verschaltet und erlauben keine Lücken

A4: Es ist erlaubt, Teilvektoren eines Ausgabe-Vektors offen zu lassen, d.h. mit keinem Eingabe-Vektor einer Funktionseinheit zu verschalten

A5: Es ist erlaubt, Teilvektoren eines Eingabe-Vektors offen zu lassen, d.h. mit keinem Ausgabe-Vektor einer Funktionseinheit zu verschalten. Offen gelassene Eingabe-Teilvektoren müssen bei der Initialisierung einen konkreten Wert bekommen.

A6: Es ist nicht erlaubt, mehrere Ausgabebits an einem einzelnen Eingabebit anzuschließen; dies muss bereits beim Aufbau der Vernetzung unter Fehlerausgabe verhindert werden.

A7: Offene Eingabe-Vektoren können zu bestimmten Zeiten mit Input-Stimuli versorgt werden. Die Simulation erwartet eine Menge von Ereignissen, die die Sequenz von Input-Stimuli über die Zeit auf den offenen Eingabe-Vektoren definiert. Die Menge der Ereignisse ist vom Benutzer mit anzugeben.

A8: Offene Eingabe-Vektoren halten ihre Werte solange, bis ein neues Ereignis ihn verändert. Danach behalten sie den durch das Ereignis neu induzierten Wert bis zum nächsten Ereignis.

A9: Zwecks besserer Handhabung soll es dem Benutzer möglich sein, Werte von Ereignissen auch in der Form einer Zeichenkette aus beliebig vielen ‚0'en und ‚1'en in die Menge der Input-Stimuli hinzuzufügen.

A10: Der Simulator soll nach jedem Simulationszyklus für jede Funktionseinheit die Eingabe-Vektoren und Ausgabe-Vektoren auf den Bildschirm drucken.

Glossar:

Funktionseinheiten sind Bausteine, die auf einem Eingabevektor Berechnungen durchführen und das Ergebnis an ihrem Ausgabevektor zur Verfügung stellen. Ein- und Ausgabevektor haben jeweils eine feste, zur Erstellzeit der Funktionseinheit definierte, Anzahl an Bits. Die Anzahl der Bits darf für Eingabe-Vektor und Ausgabe-Vektor unterschiedlich sein.

Bits sind Werte, die entweder 0 oder 1 sein können. Zusammenfassungen von Bits werden Vektor genannt.

Berechnungen sind Wertetransformationen, die in den Funktionseinheiten durchgeführt werden. Jede Transformation dauert exakt eine Zeiteinheit. Dieser Zeitverbrauch wird im Simulationszyklus dadurch emuliert, dass nur die zu Beginn des Zyklus vorliegenden Eingabe-Werte für die Berechnung benutzt werden.

Ereignisse sind ein Vektor von Bits mit einer konkreten Belegung der Bits, sowie ein dazugehöriger Zeitstempel, der den Zeitpunkt angibt, ab dem die Belegung des Vektors gültig wird.

Zeit-diskrete Simulationen führen die Simulation in einem festen Zeitraster mit konstantem Abstand der Zeitpunkte durch (beispielsweise jede Millisekunde).

Simulationszyklus nennt sich die sequentielle Abarbeitung der einzelnen Aktionen innerhalb eines Zeitschrittes der Simulation. Dieser ist wie folgt definiert:

1. Update auf den Eingabevektoren aller Funktionseinheiten zum Zeitpunkt t durch Ereignisse mit dem Zeitpunkt t aus den Input-Stimuli
2. Berechnungen der Ergebnisse in den Funktionseinheiten zum Zeitpunkt t mit den oben erzeugten aktuellen Eingabewerten
3. Update aller Ausgabe-Vektoren der Funktionseinheiten zum Zeitpunkt t
4. Propagation der Werte der Ausgabe-Vektoren an die angeschlossenen Eingabe-Vektoren der Funktionseinheiten
5. Beenden des Zeitschrittes t und fortschreiten bei $t+\Delta t$ mit Schritt 1

Input-Stimuli sind Mengen von Ereignissen, die zu konkreten, durch das Ereignis bestimmten, Zeitpunkten von außen an das zu simulierende System angelegt werden und offene Eingänge mit neuen Werten belegen.