Übungsaufgaben:

1. Objektorientierte Programmierung - Teil 1

1. Das Objekt „Bruch“ mit einem Standardkonstruktor (initialisieren mit 0), einem allgemeinen Konstruktor (Zähler und Nenner können beliebig vorgegeben werden) und einem Typumwandlungskonstruktor (der Bruch wird mit einer ganzen Zahl initialisiert). Zudem sollen die Methoden Addieren (der Bruch addiert zu seinem Wert den angegebenen Parameter), Subtrahieren (der Bruch subtrahiert von seinem Wert den angegebenen Parameter), Multiplizieren (der Bruch multipliziert seinen Wert mit dem angegebenen Parameter) und Dividieren (der Bruch teilt seinen Wert durch den angegebenen Parameter) realisiert werden. Um möglichst kleine Zähler und Nenner zu erhalten, soll eine private Hilfsmethode „Kürzen“ implementiert werden. Schließlich bietet sich noch die Methode Anzeigen an, um die Ergebnisse besser kontrollieren zu können.
2. Das Objekt „Komplexe Zahl“ mit den Attributen Real- und Imaginärteil und den Methoden wie in Aufgabe 1.
3. Das Objekt „Komplexe Zahl“ wie in Aufgabe 2, wobei Real- und Imaginärteil Atrribute vom Typ „Bruch“ sind (die Klasse aus Aufgabe 1).
4. Das Objekt „Datum“ mit einem Standardkonstruktor (initialisieren mit dem 1.1.1900) und einem allgemeinen Konstruktor (Tag, Monat und Jahr können beliebig vorgegeben werden, wobei die Gültigkeit der Daten geprüft werden soll). Als Methoden sollen „TageWeiter“ (zählt das Datum um die angegebene Anzahl von Tagen hoch), „TageZurück“ (zählt das Datum um die angegebene Anzahl von Tagen zurück) und „Anzeigen“ (zeigt das Datum in der Form TT.MM.JJJJ an) realisiert werden.
5. Das Objekt „Liste“ wie in der Vorlesung mit den Methoden Einfügen (neues Element am Ende ergänzen), Entfernen (das Element an einer vorgegebenen Stelle entfernen) und Suchen (den gewünschten Wert in der Liste suchen, und den entsprechenden Index zurückliefern). Für den Lesezugriff soll außerdem eine Zugriffsmethode „GetElement“ implementiert werden, die das Element mit dem angegebenen Index zurückliefert.

2. Objektorientierte Programmierung - Vererbung

1. Das Objekt „Liste“ wie in Aufgabe 1.5 sowie das davon abgeleitete Objekt „sortierte Liste“ (bei dem ein neues Element immer sortiert eingefügt wird). Angepaßt werden sollen hier also die Methoden Einfügen und Suchen.
2. Das Objekt „Fahrzeug“ mit den Attributen Geschwindigkeit und MaxGeschwindigkeit, sowie einem allgemeinen Konstruktor (Geschwindigkeit ist 0, MaxGeschwindigkeit kann beliebig vorgegeben werden). Das Objekt soll die Methoden „beschleunigen“, „bremsen“ und „Geschwindigkeit anzeigen“ besitzen, wobei alle drei Methoden abstrakt sind. Davon sollen die Objekte „Fahrrad“ und „Auto“ abgeleitet werden.

   Das Fahrrad besitzt die zusätzlichen Attribute „Gang“ und „MaxGang“. Beim Beschleunigen wird die Geschwindigkeit um 1 erhöht bis MaxGeschwindkeit erreicht ist. Dann wird der Gang um 1 erhöht (sofern noch möglich) und die Geschwindigkeit wieder auf 1 gesetzt. Beim Bremsen funktioniert es genau andersherum.

   Das Auto besitzt das zusätzliche Attribut „Beschleunigung“. Beim Beschleunigen wird die Geschwindigkeit immer um diesen Wert erhöht, solange die Maximalgeschwindigkeit noch nicht überschritten wird. Das Bremsen funktioniert ebenfalls wieder umgekehrt.
3. Die Objekte „Mitarbeiter“ (Atrribute Name und Personalnummer) und „Abteilungsleiter“ (von „Mitarbeiter“ abgeleitet mit den zusätzlichen Attributen Abteilung und Anzahl der Untergebenen). Die Objekte sollen einen allgemeinen Konstruktor und eine Methode „Visitenkarte“ besitzen (über die die Informationen angezeigt werden).

3. dynamische Speicherverwaltung

1. Dynamischer Stack: das Objekt „Stack“, dessen Konstruktor die gewünschte Maximalgröße mitgegeben werden kann, mit den Methoden Push und Pop sowie einem geeigneten Destruktor
2. Ein Feld mit 10 Elementen, wobei jedes Element eine Instanz der Klasse c_bruch (vgl. Übungsaufgabe 1.1). Verwendung der Methoden dieser Klasse, um die Instanzen des Feldes miteinander zu addieren, subtrahieren, ...
3. Die gleiche Aufgabe mit Elementen der Klasse c_komplex aus Aufgabe 1.2
4. Ein Objekt Liste, über dessen Konstruktor die gewünschte Anzahl an Elementen (vom Typ int) angegeben werden kann und für die per malloc Speicherplatz reserviert wird. Zudem wird ein Parameter „Delta“ übergeben. Falls nun beim Einfügen eines neuen Elements festgestellt wird, dass die Liste bereits voll ist, dann wird die Liste um Delta-viele Elemente vergrößert (per realloc). Dazu die Methoden „anfuegen“ und „loeschen“, sowie ein geeigneter Destruktor.

4. Listen

1. Eine einfach verkettete Liste, in der int-Werte gespeichert werden. Einlesen der Zahlen in einer Schleife (bis der Anwender nicht mehr will). Anschließend aufsummieren der Zahlen und Ausgabe der Summe. Speicherfreigabe am Programmende nicht vergessen!
2. Eine einfach verkettete Liste, in der Instanzen der Klasse c_bruch (vgl Übungsaufgabe 1.1) gespeichert werden. Einlesen von Zählern und Nennern und Speichern dieser Eingaben in einer Schleife (bis der Anwender nicht mehr will). Anschließend aufsummieren (unter Verwendung der Methode addieren) und Ausgabe der Summe. Speicherfreigabe am Programmende nicht vergessen!
3. Ein Objekt Stack, dessen Elemente (vom Typ int) als einfach verkettete Liste gespeichert werden
4. Ein Objekt Queue, dessen Elemente (vom Typ int) als einfach verkettete Liste gespeichert werden
5. Ein Objekt Liste, das beliebige Elemente in einer einfach verketteten Liste speichert (in der Liste wird lediglich die Speicheradresse des Elements gespeichert, die beim Anfügen des Elements als Parameter übergeben wird). Zur Abfrage eines Elements stehen 3 Methoden zur Verfügung: get_as_int, get_as_char und get_as_float, die die Daten entsprechend interpretieren und zurückliefern (der Aufrufer der Methode muß natürlich sicherstellen, dass die Daten korrekt interpretiert werden)
6. Ein Objekt Liste, bei dem die Element Instanzen einer Klasse c_element sind, die in einer einfach verketteten Liste gespeichert werden. Die Klasse c_element enthalte nur private Atrribute und Methoden und deklariere die Klasse c_liste als befreundet.

5. Sortierverfahren

7. Insertion-Sort mit einem int-Feld
8. Selection-Sort mit einer einfach verketteten Liste von Daten des Typs int
9. Bubble-Sort mit einer einfach verketteten Liste von Daten des Typs int
10. Ein Objekt „Liste“, in die die Daten unsortiert eingefügt werden und über die Methode „quicksort“ oder „heapsort“ sortiert werden können. Die Daten der Liste werden per int-Feld verwaltet.
11. Das gleiche wie in Aufgabe 4, nur dass nun Daten der Klasse c_bruch aus Aufgabe 1.1 verwendet werden. Dazu wir ein Vergleichsoperator für die Elemente dieser Klasse benötigt.

6. Bäume

1. Binärbaum mit int-Werten als Knoten (keine Duplikate).
2. Ausgabe eines Binärbaums auf dem Bildschirm.
3. Binärbaum mit einer Liste von int-Werten als Knoten (Duplikate werden in einer einfach-verketteten Liste abgelegt).
4. Gegeben ein Baum wie in Aufgabe 1. Umsortieren des Baums (zum Beispiel absteigende Sortierung) und dabei Aufbau eines neuen Baums.
5. Gegeben ein Baum wie in Aufgabe 1. Abspeichern der Werte im Baum in einer Liste. Dabei sollen die verschiedenen Möglichkeiten zum Durchlaufen umgesetzt werden (PreOrder, InOrder, PostOrder und LevelOrder).
6. Das gleiche wie in Aufgabe 5. Anschließend soll aber aus der Liste ein neuer Baum aufgebaut werden. Ausgabe der beiden Bäume wie in Aufgabe 2 und Vergleich.
7. Aufgabe 1-7 mit AVL-Bäumen.