Programmierung 1 (PR1)	Abschlussprüfung		Name/Mnr:	1

Listen von Paaren

Implementieren Sie die Klassen **`Paar`** und **`Liste`**.

Ein Paar besteht aus zwei ganzzahligen Werten (int, z.B.: (0,0), (-2,1), etc.). In der Klasse **`Paar`** sind folgende Operatoren und Methoden zu implementieren:

- Konstruktor(en): Ein Konstruktoraufruf soll mit zwei, einem oder keinem int-Parameter möglich sein. Defaultwert für nicht übergebene Parameterwerte soll 0 sein. Somit ergibt Paar{-2,1} das Paar (-2,1), Paar{5} das Paar (5,0) und Paar{} das Paar (0,0).
- Operator +: Zwei Paare können mittels + addiert werden. Das Ergebnis wird durch die komponentenweise Addition definiert. Also $(i_1, j_1) + (i_2, j_2) = (i_1 + i_2, j_1 + j_2)$, z.B. (-2,1) + (4,-3) = (2, -2). Die Operanden sollen durch die Ausführung der Operation nicht verändert werden.
- Operator *: Ein Paar kann (von links oder von rechts) mit einer ganzen Zahl multipliziert werden. Dabei werden einfach die beiden Komponenten mit der ganzen Zahl multipliziert. Also $(i_1, j_1) * k = k * (i_1, j_1) = (ki_1, kj_1)$, z.B. -2 * (2, -2) = (-4, 4). Die Operanden sollen durch die Ausführung der Operation nicht verändert werden.
- Operator <<: Ein Paar wird ausgegeben, indem die beiden Komponenten in runden Klammern und durch Komma getrennt ausgegeben werden, z.B.: (-1, 7)

Eine Liste kann eine beliebige Anzahl von Paaren (auch 0) enthalten. In der Klasse **`Liste`** sind folgende Operatoren und Methoden zu implementieren:

- Konstruktor(en): Ein Konstruktoraufruf soll mit der Angabe von vier int-Werten möglich sein: a**`nzahl_paare`** (default 1, nicht negativ), **`startwert_i`** (default 0), **`startwert_j`** (default 0) und **`schrittweite`** (default 0). Die Liste soll dann so viele Paare enthalten, wie **`anzahl_paare`** vorgibt. Das erste Paar ist (**`startwert_i`**, **`startwert_j`**) und die weiteren Paare werden durch Addition der **`schrittweite`** zu jeweils beiden Komponenten des vorhergehenden Paares gewonnen, z.B. Paar{3, 4, -1, 2} ergibt die Liste {(4, -1), (6, 1), (8, 3)}.
- Operator *: Eine Liste wird mit einer ganzen Zahl (von links oder von rechts) multipliziert, indem jedes Paar in der Liste mit der ganzen Zahl multipliziert wird, z.B.: 2*{(4, -1), (6, 1), (8, 3)} ergibt die Liste {(8, -2), (12, 2), (16, 6)}. Die Operanden sollen durch die Ausführung der Operation nicht verändert werden.
- Operator <<: Eine Liste wird ausgegeben, indem alle Paare durch Komma getrennt in geschwungenen Klammern ausgegeben werden, z.B.: {(1, 2), (5, -3), (0, 0)}. Die leere Liste wird in der Form {} ausgegeben.
- Zusatz für 15 Punkte: Operator +: Zwei Listen werden miteinander addiert, indem eine neue Liste erstellt wird, die abwechselnd jeweils ein Paar aus der ersten Liste (linker Operand) und dann ein Paar aus der zweiten Liste (rechter Operand) enthält. Enthält ein Operand kein weiteres Paar, so sind die übrigen Paare des anderen Operanden (falls vorhanden) in das Ergebnis zu übernehmen, z.B.: {(1, 2), (5, -3) } + {(4, -1), (6, 1), (8, 3), (0, 0)} = {(1, 2), (4, -1), (5, -3), (6,1), (8, 3), (0, 0)}. Die Operanden sollen durch die Ausführung der Operation nicht verändert werden.
- Zusatz für 10 Punkte : Operator +=: Wie Operator +, allerdings übernimmt der linke Operand den Wert des Ergebnisses. Der rechte Operand soll durch die Ausführung der Operation nicht verändert werden.

Implementieren Sie die Klassen **`Paar`** und **`Liste`** mit den notwendigen Konstruktoren, Methoden und Operatoren, sodass jedenfalls das Rahmenprogramm (paar.h, paar.cpp, liste.h, liste.cpp, und testliste.cpp verfügbar unter /home/Xchange/ue11) kompiliert und ausgeführt werden kann und die gewünschten Ergebnisse liefert. Achten Sie in Ihren Konstruktoren darauf, dass nur gültige Objekte erstellt werden können. Werfen Sie gegebenenfalls eine Exception vom Typ **`runtime_error`**.

Für Ihr Programm dürfen Sie **nur** die im vorgegebenen Rahmenprogramm angeführten include-Dateien verwenden!

Instanzvariablen sind **`private`** zu definieren und die Verwendung globaler Variablen ist (abgesehen von im Rahmenprogramm eventuell bereits definierten) nicht erlaubt! Interpretationsspielraum in der Angabe können Sie zu Ihren Gunsten nutzen.

Die Teilaufgaben, bei denen keine Punkteanzahl angegeben ist, gelten als Basisfunktionalität. Für eine positive Beurteilung ist zumindest die Basisfunktionalität zu implementieren. Diese wird mit 30 Punkten bewertet.

Die übrigen Teilaufgaben müssen nicht unbedingt implementiert werden, führen aber im Falle einer korrekten Implementierung zu einer entsprechenden Erhöhung der Punkteanzahl.