5. Übung des Programmierpraktikums

Abgabetermin: 12. Juni 2023, 23:59 Uhr

Die Übungen sind grundsätzlich selbst zu lösen.
Zweiergruppen sind erlaubt, aber nur unter Angabe des Partners im gut sichtbaren Kommentar.

Abzugeben sind jeweils die sinnvoll dokumentierten Programmfiles (`*.cpp`, `*.h`) in einem separaten Ordner dessen Name `bsp_`***`nummer`*** zu sein hat, d.h., der Ordner `bsp_2` gehört zu Aufgabe 2. **Speichern Sie die Funktionsdeklarationen jeweils in einem separaten Headerfile und die Funktionsdefinitionen in einem separaten Sourcefile (**.cpp* oder **.tcc*), d.h., jedes Projekt enthält mindestens drei Files mit Suffixen **.cpp*, **.h*, **.tcc*. Alle eigenen Funktionen und Klassen sind so im Headerfile zu dokumentierten, daß die Funktionalitäten für Dritte ausreichend beschrieben werden und daß `doxygen` damit arbeiten kann.**

---

20. Template-Funktionen, analog zu Bsp. 4:
    Schreiben Sie zwei Template-Funktionen zur Berechnung des arithmetischen, des geometrischen und des harmonischen Mittels. (2 Pkt.)

    (a) Funktion mit 4 Zahlen als übergebene Inputparameter und den 3 Mitteln als Outputparameter oder als Rückgabe.

    (b) Funktion mit Vektor als Inputparameter und den 3 Mitteln als Outputparameter oder als Rückgabe.

    (c) Rufen Sie die Template-Funktionen von Ihrem Hauptprogramm mit Template-Parameter `float`, `double`, `long double` auf und geben Sie im Hauptprogramm die Resultate aus.

Hier geht es weiter!
⇓

21. Entwerfen (und implementieren) Sie eine Klasse FIGUR welche als Klassenmember die Position der Figur auf einer Fläche $[-b, b] \times [-b, b]$, und deren verfügbare Leben enthält. Folgende Methoden der Klasse sind zu implementieren

    - Parameterkonstruktor welcher neben der Position (x, y) standardmäßig die Anzahl der Leben auf 3 und den Boxparameter (b) auf 8.0 setzt. [Zeile 3]
    - Methoden zur Abfrage der Koordinaten $(x,\ y)$, der verfügbaren Leben und der Boxgrenze. [Zeilen 11-12]
    - Die Methoden zur Positionsänderung der Figur: `moveby(dx,dy)`, `rotate(alpha)` (Rotation[1] um α Grad, math. positiv), `mirrorX` (Spiegelung an x-Achse).
    - Falls die Figur bei einer Positionsänderung die Box verläßt, dann verliert diese ein Leben und die ensprechende Koordinate wird auf die Boxgrenze gesetzt.
    - Positionsänderungen sind nur für lebende Figuren möglich.
    - Schreiben Sie einen Output–Operator (`<<`) für Ihre Klasse (Funktion! keine Methode) welcher Position, verfügbare Leben und Status (lebt/tot) ausgibt.
    - Schreiben Sie eine Funktion `distance(a,b)` welche den Euklidischen Abstand zwischen Figuren `a` und `b` zurückgibt. [Zeile 13]
    - Tip: Deaktivieren (Auskommentieren) Sie zu Beginn die Zeilen 4-15 und implementieren Sie die Klasse soweit, daß der Code mit Zeile 3 ausgeführt wird. Danach schrittweise die folgenden Zeilen aktivieren und die Klasse ergänzen.
    - Tip: Nutzen Sie die Rule of Five[2] (/Three/Zero).

    Ihr Hauptprogramm sollte ähnlich dem nachstehenden Vorschlag[3] aussehen. (4 Pkt.)

```
1  int main()
2  {   //     a(x=3, y=4)
3      Figure a(3,4);     cout << a << endl;
4      a.moveby(3.5,-8.5);cout << a << endl;
5      a.moveby(2,1);     cout << a << endl;
6      a.rotate(45.2);    cout << a << endl;
7
8      Figure b(a);
9      b.mirrorX();       cout << b << endl;
10     b.moveby(3,-3);    cout << b << endl;
11     cout << " b has " << b.lives() << " lives at " << b.getX() << " " << b. getY();
12     cout << " and box limit of " << b.getBoxLimit() << endl;
13     cout << " Distance from a : " << distance(a,b) << endl;
14     b.rotate(-20);     cout << b << endl;
15     b.moveby(-5,8);    cout << b << endl;
16
17     return 0;
18 }
```

[1] https://de.wikipedia.org/wiki/Drehmatrix
[2] https://en.cppreference.com/w/cpp/language/rule_of_three
[3] https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/templ_21.cpp

22. Entwerfen und implementieren Sie eine Klasse der 2D-Vektoren (hier `Vek2` genannt) sodaß der folgende Code[4] damit **unverändert** ausführbar ist. (5 Pkt.)

```
1   int main()
2   {
3       Vec2 const a(3,4);           cout << a << endl;
4       Vec2 const b(-0.75f,0.5f);   cout << b << endl;
5
6       Vec2 c(a+b);                 cout << c << endl;
7       Vec2 d = c-a;                cout << d << endl;
8       if (d!=b) {cout << " Falsch: d muss gleich b sein." << endl;}
9
10      Vec2 e = -2.0f*a-3.0f*(b-c); cout << e << endl;
11      if (e!=a) {cout << " Falsch: e muss gleich a sein." << endl;}
12
13      e *= -4.0f;                  cout << e << endl;
14      if (a.isParallel(e))       {cout << "Korrekt: e muss parallel zu a sein." << endl;}
15
16      if (a.isPerpendicular(e))  {cout << "  Falsch: e nicht senkr. zu a." << endl;}
17      if (!a.isPerpendicular(b)) {cout << "Korrekt: b nicht senkr. zu a." << endl;}
18
19      return 0;
20  }
```

Impementieren Sie **alle** notwendigen Operatoren und Methoden, nutzen Sie dabei die automatische Generierung des Kopierkonstruktors und Zuweisungsoperators, siehe Skript §9.5-9.8.

- Unterscheiden Sie zwischen Klassendeklaration und -implementierung (also Header- und Sourcefile).
- Deaktivieren Sie in obigem Code zuerst Zeilen 6–17 und starten Sie mit der Deklaration/Implementierung des Parameterkonstruktors und des Standardkonstruktors (ohne Parameter). Dazu gleich den Ausgabeoperator (keine Methode!) implementieren.
- Zeile 6: (Kopierkonstruktor und) Additionsoperator `operator+` [Methode oder Funktion].
- Zeile 7: (Zuweisungs- und) Additionsoperator `operator-` [Methode oder Funktion].
- Zeile 8: Vergleichsoperatoren `operator==` und `operator!=` [Methoden oder Funktionen].
- Zeile 13: Skalierung des Vektors mit einem Skalar via `operator*=` [Methode].
- Zeile 10: Skalierung der Kopie des Vektors mit einem Skalar via der Funktion `operator*(float, Vek2)` [Funktion].
- Zeile 14: Test auf Parallelität zweier Vektoren [Methode, auch als Funktion möglich].
- Zeile 16: Test auf Orthogonalität zweier Vektoren [Methode, auch als Funktion möglich].

23. Schreiben Sie Ihre Klasse `Vek2` aus Aufg. 22 als Templateklasse. Testen Sie Ihre Klasse mit den Templateparametern `float`, `double`, `int` analog zu Ihrem Hauptprogramm aus Aufg. 22. Die Skalierungsparameter in Zeile 10 obigen Codes sollte dann denselben Typ haben wie Ihr Templateparameter. (3 Pkt.)

[4] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/templ_22.cpp`

24. Legen Sie einen STL–Vektor der Klasse `Vek2` aus Aufg. 22 oder aus Aufg. 23 an dessen Elemente mit den Daten aus dem Inputfile input_24.txt[5] initialisiert werden. Die Daten im File sind so angeordnet, daß nach x- und y-Komponente der 2D-Vektoren $x_0 \;\; y_0$, dasselbe für die zweite Zahl $x_1 \;\; y_1$, usw. eingelesen wird.

    Benutzen Sie nunmehr **Algorithmen der STL** bzw. Methoden[6] des Containers `vector` um folgende Aufgaben durchzuführen (natürlich stets mit Kontrollausgabe danach): (6 Pkt.)

    (a) Ausgabe des (eingelesenen) Vektors a über eine eigene Funktion `operator<<` .

    (b) Zählen Sie die Anzahl der Elemente in a welche gleich $(1, 4)$ sind [STL-Algorithmus].

    (c) Kopieren des Vektors a auf einen zweiten Vektor b [STL-Algorithmus].

    (d) Sortieren Sie den Vektor b aufsteigend [STL-Algorithmus]. Nutzen Sie zum Sortieren die Länge (den Betrag) eines 2D-Vektors als Quasiordnung.

    (e) Entfernen aller mehrfach vorkommenden Elemente aus b [STL-Algorithmus + Methode].

    (f) Kopieren aller Elemente mit mind. einer negativen Koordinate aus b in einen Vektor c [STL-Algorithmus + Lambda-Fkt.].

    (g) Kopieren aller 2D-Vektoren aus a welche orthogonal zu $(1.0, -0.25)$ sind [STL-Algorithmus + Lambda-Fkt.].

    (h) Entfernen aller zu (0.25,1) parallelen 2D-Vektoren aus b [STL-Algorithmus + Lambda-Fkt. + Methode].

    +1 Pkt Entfernen aller parallelen 2D-Vektoren aus a, d.h. jede Richtung darf nur einmal vorkommen.

    Hinweise, siehe auch Vorlesungsbsp. zur STL[7] und §11.3 des Skriptes[8]:

    - Es ist leichter wenn Sie obige Aufgaben a)-f) zuerst mit einem entsprechend initialisierten `vector<int>` probieren. Wenn das funktioniert, dann nacheinander bei jeder Teilaufgabe die Instanz vom `vector<int>` mit einer Instanz von `vector<Vek2>` ersetzen.
    - Schauen Sie sich Funktion und die Beispiele(!!) der folgenden Algorithmen/Methoden im www[9] an, verwenden Sie diese wenn möglich: `sort`, `unique`, `copy`, `copy_if`, `count`, `count_if`, `find`, `remove`, `remove_if`, `sort`, `max`, `max_element`, `vector<T>::erase` .
    - Fügen Sie Ihrer Klasse `Vek2` die notwendigen Vergleichsoperatoren ($<$, $==$) als Methoden hinzu und nutzen Sie Lambda-Funktionen.

    (*) Für Interessierte: Probieren Sie sie Aufgaben auch für eine Liste Ihrer Klasse aus. Achtung, der Algorithmus `sort` ist eine Methode des Containers `list`.

Hinweise:
Die Schlüsselwörter `continue, goto` dürfen nicht benutzt werden, `break` nur in Verbindung mit der `switch`-Anweisung. Desgleichen darf ein `return` nur als letzte Anweisung einer Funktion vorkommen.

Die Abgabe der Lösungen (`*.cpp`-Files, `*.tcc`, `*.h`, (`Makefile*`,) ...,) erfolgt an der KFU-Moodle, siehe dazu die Hinweise auf der LV-Homepage[10].

[5] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/input_24.txt`
[6] `http://www.cplusplus.com/reference/vector/vector/?kw=vector`
[7] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/SS23/v_9a.zip`
[8] `https://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Script/html/script_programmieren.pdf`
[9] `http://www.cplusplus.com`
[10] `http://imsc.uni-graz.at/haasegu/Lectures/Kurs-C/Download/kfu.html`

Die Verzeichnisnamen mit den Files für die jeweilige Aufgabe **müssen** dem Schema `bsp_nummer` folgen, z.B. enthält das Verzeichnis (der Ordner) `bsp_1` alle Files für Beispiel 1. Andere Verzeichnisnamen zählen als nicht abgegeben.
Keine Lehrzeichen, Sonderzeichen oder Umlaute in File- und Verzeichnisnamen benutzen (Portabilität).