Programmiertechnik II

Sommersemester 2022 – Aufgabenblatt 1

Hasso-Plattner-Institut
Fachgebiet Computergrafische Systeme

Anmerkungen zu Aufgabenblatt 1

Der zu verwendende Programmrahmen steht als Zip-Archiv `uebung1.zip` im Moodle unter der Adresse `https://moodle.hpi.de/course/view.php?id=297` zum Download zur Verfügung.

Allgemeine Hinweise
Versuchen Sie als erstes, den Programmrahmen zu verstehen. Ermitteln Sie dann im Programmrahmen die Stellen, die Sie selbst implementieren sollen, um die Aufgaben korrekt zu lösen; diese Stellen sind besonders gekennzeichnet. Kommentieren Sie bei Bedarf bitte Ihren Quellcode.

Bevor Sie mit der Lösung beginnen, lesen Sie das gesamte Aufgabenblatt und überprüfen Sie, ob der Programmrahmen auf Ihrem System fehlerfrei kompiliert und alle Programme ausführbar sind. Nehmen Sie die Übungstermine wahr.

Zielsetzung
Ziel dieses Aufgabenblattes ist es, sich mit der elementaren C++-Programmierung sowie einem C++-Compiler Ihrer Wahl grundlegend vertraut zu machen und die Funktion von formalen Sprachen, bzw. Grammatiken nachzuvollziehen.

Viel Spaß und Erfolg bei der Bearbeitung der Aufgaben!

Aufgabe 1.1: Berechnung von Fibonacci-Zahlen (10 Punkte 3+2+2+3)

In der Fibonacci-Folge ergibt die Summe zweier benachbarter Zahlen die unmittelbar folgende Zahl. Jede Zahl der Folge kann mit einem rekursiven Bildungsgesetz berechnet werden:

$$\mathrm{fib}\,(n) = \begin{cases} 1, & n \leq 2 \\ \mathrm{fib}(n-1) + \mathrm{fib}(n-2), & n > 2 \end{cases}, n \in \mathbb{N}^+ \tag{1}$$

a) Implementieren Sie die Funktion `fibonacci()` in `fibonacci_recursive.cpp` zur Berechnung der n-ten Fibonacci-Zahl. Verwenden Sie dazu die *rekursive* Formel 1. Achten Sie darauf, dass die Eingabeparameter für Anfragen von ungültigen oder mittels `signed int` nicht ausdrückbaren Fibonacci-Zahlen abgefangen werden, indem Ihre Berechnungen frühzeitig 0 zurückgeben.

b) Implementieren Sie die Funktion `fibonacci()` auf Basis eines iterativen Berechnungsschemas in `fibonacci_iterative.cpp`. Fangen Sie auch hier ungültige Eingaben ab.

c) Ergänzen Sie die Ausgabe beider Programme um die Anzahl der Berechnungsschritte (Summierungen) im folgenden Format: `<n> : <fib(n)> : #<number_of_summations>`. Die Ausgabe für $n = 8$ wäre beispielsweise in der rekursiven Variante `8 : 21 : #41`.

d) Gegeben sei eine Treppe mit n Stufen. Um diese Treppe hinauf (oder herab) zu steigen, können je Schritt eine oder zwei Stufen genommen werden. Implementieren Sie aufbauend auf Ihren Erkenntnissen über die Fibonacci-Folge die Funktion `combinations()` in `walking_stairs.cpp` zur Berechnung der Anzahl aller möglichen Schrittkombinationen, um eine Treppe mit n Stufen hinauf oder herab zu steigen. Wählen Sie eine Implementierung, die in der Lage ist, das Ergebnis für 92 Stufen *zügig* zu berechnen.
Hinweis: Machen Sie sich dazu mit den Grunddatentypen von C++ vertraut.

Aufgabe 1.2: Berechnung von Polynomen (10 Punkte 4+2+4)

Eine Summe der Form

$$P(x) = \sum_{i=0}^{n} a_i x^i; n \in \mathbb{N}; a_i, x \in \mathbb{R}; a_n \neq 0 \tag{2}$$

heißt ein Polynom in der Variable x vom Grad n.

a) Zunächst soll das Programm *polynom* die notwendigen Eingabeparameter $x, n, a_0, \ldots, a_n$ einlesen. Erweitern Sie dazu die Funktion `main()` in `polynom.cpp`. Achten Sie dabei auf eine Überprüfung auf gültige Eingabewerte und nutzen Sie `std::vector` für die Verwaltung der Koeffizienten.

b) Erweitern Sie das Programm *polynom* um die Ausgabe von $P(x)$. Vervollständigen Sie dazu die Funktion `polynom()` in `polynom.cpp` und geben Sie die Summe auf der Konsole via `std::cout` in `main()` aus.

c) Verbessern Sie die Lesbarkeit der Ausgabe von $P(x)$, indem Sie deren Formatierung eigenständig um Tausendertrennzeichen erweitern. Beispielausgabe für fixierte Koeffizienten $a_0 = 3, a_1 = 4, a_2 = 5, a_3 = 6$ und $x = 10$ ist `6.543`. Implementieren und nutzen Sie dazu die Funktion `prettyPrint()`, welche ihren Eingabeparameter entsprechend formatiert auf der Konsole ausgeben soll.

Aufgabe 1.3: Theoretische Aufgabe: Grammatik (10 Punkte 7+2+1)

Übernehmen Sie die Aufgabe eines Taschenrechners, Eingaben auf *syntaktische* Korrektheit zu überprüfen.

a) Geben Sie eine Grammatik in erweiterter Backus-Naur-Form an, die nachfolgende Anforderungen für eine gegebene `Eingabe` erfüllt.

- Es können Ganzzahlwerte oder Kommazahlen eingegeben werden, die positiv oder negativ sein können.
- Es werden die Operationen Addition (`+`), Subtraktion (`-`), Multiplikation (`*`), Division (`/`) und Potenz (`^`) unterstützt.
- Um Ausdrücke können Klammern gesetzt werden.
- Es können die Modulooperation (`%`) und Rundungsfunktion (`round()`, kaufmännisches Runden auf nächste Ganzzahl) verwendet werden.
- Die Zahlen π (`pi`) und e (`e`) stehen zur Verfügung.
- Werte können in einem von 10 Speicherplätzen (`M0` bis `M9`) abgelegt werden (`-> M_`). Dies ist auch mit Zwischenergebnissen möglich, z. B. würde die Eingabe `13 * 27 -> M2 + 5` den Wert `356` zurückgeben und `351` im Speicherplatz `M2` ablegen.
- Werte können aus Speicherplätzen wieder abgerufen werden (`M_`). So ergäbe nach obiger Belegung die Eingabe `M2 / 9` das Ergebnis `39`.

b) Geben Sie ein Beispiel für eine Eingabe an, das von Ihrer Grammatik akzeptiert und in dem ein Element jeder Anforderung verwendet wird.

c) Erklären Sie für zwei weitere Beispiele, warum diese im Rahmen Ihrer Grammatik ungültig sind.

Zusatzaufgabe: Wechselgeldrückgabe (3 Bonuspunkte)

Für die Wechselgeldrückgabe eines automatischen Kassensystems soll die Differenz zwischen dem gezahlten und dem zu zahlenden Betrag immer über die kleinstmögliche Anzahl von Münzen und Scheinen erfolgen. Für die Aufgabe wird vereinfachend angenommen, dass stets genügend Münzen und Scheine im System enthalten sind.

Implementieren Sie die Methode `change` in `change.cpp`, welche für die Eingabeparameter `due` (zu zahlender Betrag) und `paid` (gezahlter Betrag) ermittelt. Die Ausgabe gibt dabei den Cent-Betrag der Münze bzw. des Scheins und die ermittelte Anzahl an. Dabei sollen ausschließlich Münzen und Scheine bis 50 Euro unterstützt werden.

Zur Vereinfachung sollen alle Euro-Beträge in Cent kodiert werden, d. h. ein zu zahlender Betrag von 7,21 Euro entspricht 721 bzw. ein 10-Euro-Schein entspricht 1000.

Die Konsolenausgabe soll im CSV-Format (Comma Separated Values) erfolgen. Für einen zu zahlenden Betrag von 0,89 Euro und einem gezahlten Betrag von 10,00 Euro (`change 89 1000`) wird beispielsweise ausgegeben:

```
coin,num
500,1
200,2
10,1
1,1
```

Die Ausgabe soll mit der Option `-o <Filename>` in eine Textdatei geschrieben werden. Sorgen Sie dafür, dass Ihr Programm alle ungültigen Eingaben ohne Ausgabe abfängt.

Allgemeine Hinweise zur Bearbeitung und Abgabe

- Die Aufgaben können allein oder zu zweit bearbeitet werden. Je Gruppe ist nur eine Abgabe notwendig.
- Bonuspunkte können innerhalb des Übungsblattes verlorene Punkte ausgleichen. Sie sind nicht auf andere Übungsblätter übertragbar und werden nicht über die reguläre Punktzahl hinaus angerechnet.
- Bitte reichen Sie Ihre Lösungen bis spätestens **Dienstag, den 10. Mai um 17:00 Uhr** ein.
- Die Implementierung kann auf einer üblichen Plattform (Windows, Linux, OS X) erfolgen, darf aber keine plattformspezifischen Elemente enthalten, d. h. die Implementierung soll plattformunabhängig entwickelt werden.
- Die Lösungen von theoretischen Aufgaben können als Textdatei oder PDF abgegeben werden. Benennen Sie die Dateien im Format `<Aufgabenblatt>_<Aufgabe>_<Teilaufgabe>`, bspw. „`1_3_a.txt`“.
- Bestehen weitere Fragen und Probleme, kontaktieren Sie den Übungsleiter oder nutzen Sie das Forum im Moodle.
- Archivieren Sie zur Abgabe Ihren bearbeiteten Programmrahmen und die Lösungen der theoretischen Aufgaben als Zip-Archiv und ergänzen Sie Ihre Namen im Bezeichner des Zip-Archivs im folgenden Format: **`uebung1_Grp_MatNr1_MatNr2.zip`**,
  bspw. `uebung1_grp1_123456_654321.zip`. Beachten Sie, dass dabei nur die vollständigen Lösungen sowie eventuelle Zusatzdaten gepackt werden (alle Dateien, die im gegebenen Programmrahmen vorhanden waren); vermeiden Sie bitte nicht notwendige Unterordner. Laden Sie keine Kompilate und temporären Dateien (*.obj, *.pdb, *.ilk, *.ncb, *.exe, etc.) hoch. Testen Sie vor dem Hochladen, ob die Abgabe fehlerfrei kompiliert und ausgeführt werden kann.
- Reichen Sie Ihr Zip-Archiv im Moodle ein:
  `https://moodle.hpi.de/course/view.php?id=297`.