ulm university universität
uulm

Labor Softwareentwurf mit Multiparadigmen-Programmiersprachen
H. Falk, A. Luppold

Versuch 2
vom 3. November 2014

Bearbeitung bis: 9. November 2014

Prozedurale Programmierung II

Versuch 2-1: Theorie (10 Punkte)

a) Was ist der Unterschied zwischen der Deklaration und der Definition einer Funktion?

b) Wieso sollten Funktionen in Header-Dateien deklariert werden?

c) Gibt es Fälle, in denen es sinnvoll ist, eine Funktion auch in der Header-Datei zu definieren? Welche?

d) Was versteht man unter einem std::initializer_list-Parameter einer Funktion. Wozu dient er?

e) Was versteht man unter dem Überladen einer Funktion? Welche Regeln müssen dabei eingehalten werden?

Versuch 2-2: Bibliotheken und Namespaces (25 Punkte)

Hinweis: Benutzen Sie für diese Aufgabe **nicht** die von der Standardbibliothek bereitgestellten Listen.

a) Erstellen Sie eine Menge an Funktionen, mit denen Sie eine doppelt verkettete Liste erzeugen und verwalten können. Ihre Implementierung soll die folgenden Funktionen beinhalten:

```
1  // Dies ist die Struktur eines Knotens der doppelt verketteten
2  // Liste
3  struct node {
4    node* prev = nullptr;
5    node* next = nullptr;
6    int daten = 0;
7  };
8
9  // Sucht das Ende der Liste ausgehend von *node, und haengt
10 // einen neuen Knoten mit dem Inhalt von "daten" an das Ende
11 // an. Rueckgabe true, wenn der Knoten erfolgreich angefuegt
12 // werden konnte, false im Fehlerfall
13 bool push_back(node* knoten, int &daten);
14
```

```
15  // Fuegt einen neuen Knoten mit Inhalt "daten" hinter "knoten" ein.
16  bool insert_after(node* knoten, int &daten);
17
18  // Loescht den Knoten knoten aus der Liste
19  bool remove(node* knoten);
20
21  // Zaehlt die Anzahl an Elementen in der Liste
22  std::size_t count(node* knoten);
23
24  // Gibt das letzte Element der Liste zurueck.
25  node* last_element(node* knoten);
26
27  // Gibt das erste Element der Liste zurueck.
28  node* first_element(node* knoten);
```

Verwenden Sie zum Allokieren des Speichers neuer Knoten new und zum Löschen delete. Zur Kennzeichnung des ersten Elements soll dessen Zeiger *prev auf den Nullzeiger verweisen. Entsprechend soll *next des letzten Elements der Liste ebenfalls auf den Nullzeiger verweisen.

Testen Sie alle Möglichkeiten Ihrer Liste, indem Sie eine main()-Funktion hinzufügen, und von dort die Listenfunktionen aufrufen.

b) Sie werden nun die Listenfunktionen und die konkrete Implementierung trennen. Verschieben Sie hierzu Ihre main()-Funktion und alle eventuell vorhandenen weiteren Funktionen, die nicht für die Liste notwendig sind, in eine neue Datei. Benennen Sie Ihre Dateien so, dass deutlich wird in welcher Datei sich die Listenfunktionen, und in welcher die konkrete Implementierung befindet.

   Binden Sie in Ihre Implementierung den Header Ihrer Liste ein. Erstellen Sie mit dem Compiler-Parameter −c eine Bibliothek aus der Quelldatei mit denListenfunktionen.

   Wie unterscheidet sich die entstandene Binärdatei von einer ausführbaren Programmdatei? Wie müssen Sie bei der Übersetzung der Implementierung den Compiler-Aufruf anpassen, um die Bibliothek nutzen zu können? Was genau bewirkt ihr Parameter?

c) Betten Sie die Funktionen Ihrer Liste in einen eigenen namespace ein. Welche Auswirkungen, welche Vorteile und welche Nachteile hat dies?

Versuch 2-3: Funktionspointer (5 Punkte)

Fügen Sie Ihrer Liste aus Versuch 2-2 eine weitere Funktion zum aufsteigenden Sortieren der Liste hinzu. Nutzen Sie hierzu die Sortierfunktion std::qsort().

Erläutern Sie, wieso std::qsort() ein Funktionszeiger übergeben werden muss. Welche Nachteile erkennen Sie beim Einsatz von std::qsort()? Nennen Sie eine mögliche Alternative (Diese muss **nicht** implementiert werden!).

Versuch 2-4: Variable Anzahl an Parametern (5 Punkte)

Erweitern Sie Ihre Liste aus Versuch 2-2 um eine Funktion insert_multi(). Diese soll es ermöglichen, eine variable Anzahl an Knoten mit einem Befehlsaufruf zu erzeugen. Nutzen Sie hierzu std::initializer_list.

Versuch 2-5: Überladen von Funktionen (5 Punkte)

Übergeben Sie beim Anlegen eines Knotens in Ihrer Liste als Wert eine Gleitkommazahl anstatt eines Integers. In Ihrer jetzigen Implementierung wird der Gleitkommawert implizit auf einen ganzzahligen Wert konvertiert. Dies geschieht, indem die Nachkommastellen abgeschnitten werden.

Überladen Sie die Funktionen Ihrer Liste so, dass beim Übergeben einer Gleitkommazahl mathematisch korrekt auf- oder abgerundet wird,

Versuch 2-6: Nutzung der Standardbibliothek (10 Punkte)

Implementieren Sie die vorhergehenden Aufgaben mit std::list anstatt eigene Funktionen zu nutzen. Vergleichen Sie beide Lösungen hinsichtlich Laufzeiteffizienz und Code-Größe. Wie erklären Sie sich die Unterschiede? Hinweis: Zum Messen der Laufzeit innerhalb Ihres Programmcodes können Sie auf die Klassenbibliothek std::chrono zurückgreifen.