’Dover Town’ – Teil 1

Operatoren

- Implementieren Sie eine Klasse ’states’, die eine Menge von Zuständen enthalten kann, wovon genau einer aktiv ist. Beispiele von Zuständen: { ’Locked’, ’Unlocked’ } oder auch { ’Connected’, ’Disconnected’, ’Unknown’ }.
- Nutzen Sie intern einen ’vector’ von ’string’ zum Speichern der States, und einen Index für den gerade aktuellen Zustand.
- Grundsätzlich soll ein bestimmter Zustand abgefragt und gesetzt werden können. Weiter kann man den aktuellen Zustand um eins weiter oder um eins zurück setzen. Natürlich sollen auch Zustände hinzugefügt werden und alle gelöscht werden können.

Fortsetzung…

’Dover Town’ – Teil 2

- Implementieren Sie geeignete Operatoren bzw. Funktionen, so dass folgende Codeschnipsel für eine ’states’-Instanz ’s’ funktionieren:
  - ’s+=“home”’ fügt Zustand ’home’ zu ’s’ hinzu.
  - ’s=1;’ setzt aktuellen Zustand auf den zweiten Zustand.
  - ’++s;’ setzt den aktuellen Zustand um eins weiter.
  - ’--s;’ setzt den aktuellen Zustand um eins zurück.
    Beim Über- oder Unterlauf vorne bzw. hinten anfangen.
  - ’s[s()]’ gibt den aktuellen Zustand (string) zurück.
  - ’s.clear();’ löscht alle Zustände.
  - ’cout « s;’ gibt einen Vektor mit allen Zuständen aus.

Testen Sie Ihren Code aussagekräftig.

’Heart Land’ – Teil 1

Operatoren

- Implementieren Sie eine generische Klasse ’fastvector’, die einen mathematischen Vektor von Koeffizienten einer festen Dimension modelliert, mit dem man rechnen kann.
- Nutzen Sie intern ein ’array’ fester Länge. Die Dimension und der Typ werden durch die Template-Parameter bestimmt.
- Grundsätzlich soll addiert, subtrahiert und mit einem Skalar multipliziert werden können. Natürlich kann man auch einzelne Koeffizienten lesen und setzen.

Fortsetzung…

’Heart Land’ – Teil 2

- Implementieren Sie geeignete Operatoren bzw. Funktionen, so dass folgende Codeschnipsel für eine ’fastvector’-Instanz ’v’, bzw. ’v1’ und ’v2’, funktionieren:
  - ’v=1;’ setzt alle Koeffizienten auf einen Wert, hier 1.
  - ’v1+v2’ addiert ’v1’ und ’v2’.
  - ’v1-v2’ subtrahiert ’v2’ von ’v1’.
  - ’v*3’ bzw. ’3*v’ multiplizieren v mit einem Skalar, hier 3.
  - ’-v’ negiert alle Koeffizienten.
  - ’v[i]’ liest bzw. schreibt den i’ten Koeffizienten.
  - ’cout « v;’ gibt die Koeffizienten von ’v’ aus.

Testen Sie Ihren Code aussagekräftig.

’Deerwoods’ – Teil 1

Operatoren

- Implementieren Sie eine generische Klasse ’nullable’, die einen mathematischen Wert modelliert, der auch ’null’ sein kann.
- Nutzen Sie intern ein Attribut vom Datentyp, der durch den Template-Parameter bestimmt wird, sowie einen ’bool’, welcher angibt, ob der Wert ’null’ ist oder nicht.
- Grundsätzlich soll addiert werden können. Natürlich kann man auch auf ’null’ testen und den Wert ’null’ setzen.

Fortsetzung…

’Deerwoods’ – Teil 2

- Implementieren Sie geeignete Operatoren bzw. Funktionen, so dass folgende Codeschnipsel für eine ’nullable’-Instanz ’n’, bzw. ’n1’ und ’n2’, funktionieren:
  - ’n=1;’ setzt einen Wert, hier 1.
  - ’n.reset();’ bedeutet, ’n’ auf ’null’ zu setzen.
  - ’n+=3;’ addiert einen Wert zu ’n’, hier 3.
  - ’n1+n2’ addiert ’n1’ und ’n2’.
  - ’!n’ testet auf ’null’ und ist wahr, wenn nicht.
  - ’(int)n;’ wandelt in den konkreten Datentyp um, hier ’int’ .
  - ’cout « n;’ gibt den Wert von ’n’ aus.

Testen Sie Ihren Code aussagekräftig.

0x07 – Übungen

Hausübung

- Gucken Sie sich an, welche Operatoren in C++ grundsätzlich überladen werden können. Eine Übersicht findet sich beispielsweise bei cppreference

Selbstkontrolle

- Ich habe alle Codes und Übungsthemen verstanden.
- Ich kenne ’operator overloading’ und kann unäre und binäre Operatoren überladen.
- Ich kann erklären, warum manche Operatoren Member sind und manche nicht.
- Ich kenne Designregeln, die zu beachten sind, wenn man Operatoren überlädt.
- Ich weiß, was 'friend' bedeutet.
- Ich kenne die unterschiedlichen Bedeutungen von 'static'.