universität wien

Implementierung einer Vector Klasse

unter Verwendung der C++-Konstrukte Array und Pointer

Ziel

Implementierung einer Klasse Vector analog zu std::vector<double>

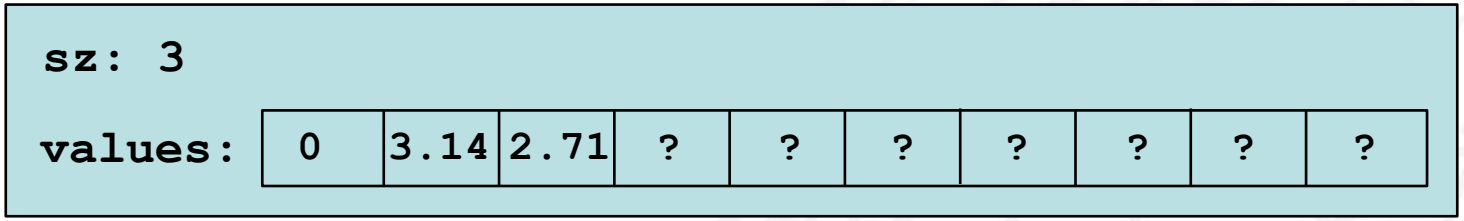
In einer ersten Version beschränken wir uns auf Vektoren mit einer fixen maximalen Größe. Diese wird in der Header-Datei als globale Konstante **`vector_max_size`** vom Type **`size_t`** definiert.

Für einen Vektor werden zwei Instanzvariablen benötigt:

- Ein double-Array **`values`** der Größe **`vector_max_size`**, um die Werte zu speichern.
- Ein Wert **`sz`** vom Typ **`size_t`**, der die aktuell gespeicherte Anzahl von Werten enthält.

Ein Vektor (schematisch)

`vector_max_size: 10`



Zu allen Zeitpunkten müssen folgende Integritätsbedingungen gelten:

- $0 \leq sz \leq vector\_max\_size$
- Die gespeicherten double Werte sind $values[0]$ ... $values[sz-1]$
- sz gibt den nächsten freien Index im Feld $values$ an.

Funktionalität

Methoden:

- Konstruktor: erstellt einen leeren Vektor
- **`size_t size() const`**: liefert die Anzahl der gespeicherten Elemente
- **`bool empty() const`**: liefert **`true`**, falls der Vektor leer ist, **`false`** sonst
- **`void push_back(double)`**: fügt einen Wert am Ende ein; Exception, falls der Vektor schon voll ist.
- **`void pop_back()`**: entfernt den Wert am Ende; Exception, falls der Vektor schon leer ist.
- **`double& operator[](size_t)`**: liefert eine Referenz auf den indizierten Wert, bzw. eine Exception, falls der Index ungültig ist.
- **`void clear()`**: entfernt alle Werte aus dem Vektor.
- **`ostream& print(ostream &) const`**: Gibt die Werte in Form einer Komma-separierten Liste (z.B.: [1, 2, 3]) aus.

globale Funktionen:

- **`bool operator==(const Vector& lop, const Vector& rop)`**: true, falls der Inhalt der beiden Vektoren exakt übereinstimmt
- **`bool operator>(const Vector& lop, const Vector& rop)`**: true, falls rop in der lexikographischen Anordnung vor lop kommt.
- Analog für **`operator!=`**, **`operator>=`**, **`operator<`** und **`operator<=`**.

Vorgehensweise

1. Erstellen Sie die Klassendefinition mit dem erforderlichen Interface in einer Datei vector.h.
2. Erstellen Sie die Implementierung der Klasse in einer Datei vector.cpp.
3. Erstellen Sie ein Hauptprogramm, in dem Sie die Funktionalität der Klasse testen.

Problem

Versuchen Sie folgende Funktion zu Ihrem Testprogramm hinzuzufügen:

```
void foo(const Vector& v) {
  cout<<v[0];
}
```

1. Welchen Fehler erhalten Sie?
2. Was ist die Ursache?
3. Wie kann man diesen Fehler beheben?
4. Passen Sie Ihre Klassendefinition (vector.h) und –implementierung (vector.cpp) entsprechend an.

Erweiterungen

Implementieren Sie Funktionen, um die Summe, das Minimum, das Maximum und den Mittelwert der gespeicherten Werte zu ermitteln (Warum gibt es derartige Funktionen nicht in **`std::vector`**?).

Überladen Sie **`operator<<`**, sodass die in einem Vektor gespeicherten Werte in einer Komma-separierten Liste ausgegeben werden, z.B.: [1, 2, 0.5]

Implementieren Sie Methoden **`insert(size_t, double)`** und **`erase(size_t)`**, die einen Wert an einer vorgegebenen Stelle einfügen, bzw. von einer vorgegebenen Stelle entfernen.

Anmerkung: Für diese Erweiterungen sind die Signaturen der Funktionen nicht mehr genau vorgegeben. Überlegen Sie sich selbst sinnvolle Returnwerte und Prototypen.

universität wien

Spezialaufgabe

Implementieren Sie einen Konstruktor, der es erlaubt, Vektoren mit einer Liste von Werten zu initialisieren, z.B.:

```
Vector v {1, 2, 3};
```

Dazu benötigen Sie einen Konstruktor, der eine initializer_list als Parameter akzeptiert. Das wurde in der Vorlesung nicht behandelt. Fragen Sie in der Übungsgruppe oder im Forum nach, bzw. googeln Sie, wie ein solcher Konstruktur aussieht.