C/C++ Programmierung
Übungsblatt 11

Florian Oetke

1 Konvertieren von Zahlensystemen

Implementieren Sie die folgenden drei Funktionen:

- `parse_binary`: Bekommt einen `std::string`, mit der Binärdarstellung einer Zahl, als Parameter und gibt die repräsentierte Zahl als `std::uint32_t` zurück.
- `to_binary`: Bekommt eine Zahl (`std::uint32_t`) als Parameter und gibt einen `std::string` mit der Binärdarstellung dieser zurück. Führende Nullen auf der linken Seite sollen ausgelassen werden.
- `to_hex`: Bekommt eine Zahl (`std::uint32_t`) als Parameter und gibt einen `std::string` mit der Hexadezimaldarstellung dieser zurück. Führende Nullen auf der linken Seite sollen ausgelassen werden.

Als Basis für die Implementierung können Sie den im Handout bereitgestellten Programmcode verwenden, der für die Beispieleingabe `"11100"` die folgende Ausgabe liefern sollte:

```
Dec : 28
Bin : 11100
Hex : 1c
```

2 Dynamisches Bitset

In dieser Aufgabe sollen Sie eine einfache Klasse implementieren, die eine beliebig große Menge von Bits verwaltet. Um die einzelnen Bits zu speichern, können Sie entsprechend nicht einfach nur einen einzelnen Integer mit einer festen Größe verwenden, sondern benötigen einen Container von Integern.

Schreiben Sie eine Klasse `Dynamic_bitset` mit einer Member-Variable `size_`, in der Sie sich die aktuell gespeicherte Anzahl Bits merken, und einer Member-Variable `bytes_`, die ein `std::vector` von `std::uint8_t` ist und in jedem Element jeweils 8 Bits speichert. Das heißt insbesondere, dass Sie weder `std::vector<bool>` noch `std::bitset` für Ihre Implementierung verwenden sollen.

Darüber hinaus sollte Ihre Klasse die folgenden Methoden implementieren:

- `size`: Gibt die Anzahl der gespeicherten Bits zurück
- `get`: Bekommt einen Index als Parameter und gibt den Wert des Bits an der entsprechenden Stelle zurück, d.h. `true` wenn das Bit gesetzt ist und `false` wenn nicht. Für Indices, die größer als die aktuelle Größe sind, soll immer `false` zurückgeliefert werden.
- `set`: Bekommt einen Index und einen `bool` als Parameter und ändert den Wert des entsprechenden Bits, d.h. wenn der zweite Parameter `true` ist, soll das Bit gesetzt werden und wenn er `false` ist, soll das Bit wieder auf 0 gesetzt werden. Wenn der Index größer als die aktuelle Größe ist, sollen alle neuen Bits dazwischen auf 0 gesetzt werden.

Als Basis für die Implementierung können Sie den im Handout bereitgestellten Programmcode verwenden, der bei einer korrekten Lösung die folgende Ausgabe erzeugen sollte:

```
bits.get(5) = 0

bits.get(0) = 0
bits.get(1) = 0
bits.get(2) = 0
bits.get(3) = 0
bits.get(4) = 1
bits.get(5) = 0
bits.get(6) = 1
bits.get(7) = 0
bits.get(8) = 1
bits.get(9) = 0
```

3 Binärer Suchbaum

In dieser Aufgabe wollen wir uns nochmals etwas mehr mit `std::unique_ptr` beschäftigen, indem wir mit seiner Hilfe einen binären Suchbaum implementieren.

Schreiben Sie ein Klassen-Template `Binary_search_tree<T>` mit folgenden Member-Funktionen:

- `insert`: Bekommt einen Parameter vom Typ `T` und fügt ein neues Element mit dem Wert in die Struktur hinzu, sofern der Wert noch nicht existiert
- `operator[]`: Bekommt einen zu suchenden Wert vom Typ `T` als Parameter und gibt entweder `nullptr` oder einen `const T*` auf das Element zurück, falls es sich in der Struktur befindet
- `print`: Gibt die gespeicherte Baumstruktur, wie in der Beispielausgabe unten, aus

Die Werte sollen als sortierter Binärbaum gespeichert werden. In Ihrer Klasse benötigen Sie also ein `struct` für die Knoten, das neben dem Wert jeweils einen `std::unique_ptr<Node>` auf seinen rechten und linken Kind-Knoten enthält. In Ihrem `Binary_search_tree` sollten sie nur eine Member-Variable vom Typ `std::unique_ptr<Node>` haben, welche die Wurzel des Baums enthält, sofern er nicht leer ist.

Durch die Sortierung wird sichergestellt, dass alle Elemente im linken Teilbaum eines Knotens kleiner und alle im rechten größer sind als der Knoten selbst.

Als Basis für die Implementierung können Sie den im Handout bereitgestellten Programmcode verwenden, der bei einer korrekten Lösung z.B. die folgende Ausgabe erzeugen sollte:

```
[C]
    [A]
        *empty*
        [B]
            *empty*
            *empty*
    [D]
        *empty*
        [F]
            *empty*
            *empty*
```

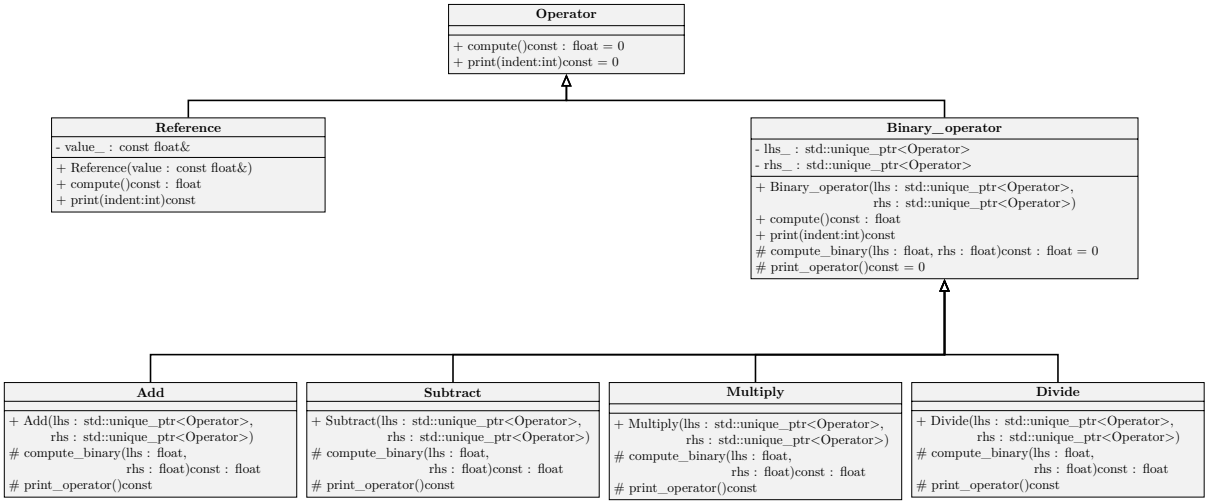
Das heißt für die eingefügten Werte `"C"`, `"D"`, `"A"`, `"F"`, `"B"`, `"A"` besteht der Baum aus `"C"` an der Wurzel mit `"A"` im linken und `"D"` im rechten Kind-Knoten, und so weiter.

Hinweis
Sowohl für `insert` als auch für `operator[]` müssen Sie den Baum rekursiv nach dem Knoten für einen bestimmten Wert durchsuchen. Hierzu bietet es sich also u.U. an, eine statische Hilfsmethode zu schreiben, die für einen bestimmten Teilbaum und Wert eine Referenz auf den `std::unique_ptr` zurückgibt, indem der Knoten gespeichert ist bzw. gespeichert werden sollte, wenn er noch nicht im Baum ist. Dies funktioniert z.B. nach dem folgenden Schema:

1. Wenn der `std::unique_ptr<Node>` Parameter, der den aktuellen Teilbaum angibt, leer ist, gib eine Referenz auf genau diesen `std::unique_ptr` zurück. Das ist die Position, an der der Wert stehen müsste, wenn er im Baum wäre.
2. Wenn der Wert des Knotens `==` dem gesuchten Wert ist, gibt auch eine Referenz auf genau diesen `std::unique_ptr` zurück.
3. Wenn der gesuchte Wert `<` als der Wert im Knoten ist, rufe die Funktion rekursiv mit dem linken Kind-Knoten auf.
4. Wenn der gesuchte Wert `>` als der Wert im Knoten ist, rufe die Funktion rekursiv mit dem rechten Kind-Knoten auf.

4 Klassenhierarchie für Operatoren

Implementieren Sie die Klassenhierarchie, die durch das folgende UML-Diagramm beschrieben wird:



In dem Diagramm sind die folgenden Aspekte absichtlich ausgelassen worden und sollen von Ihnen passend festgelegt/implementiert werden, wie es in der Vorlesung besprochen wurde:

- Special-Member-Functions (Destruktor, Kopierkonstruktor, Zuweisungs-Operatoren, ...)
- Welche Methoden als `virtual` markiert werden müssen
- Welche Klassen als `final` markiert werden sollten
- Welche virtuellen Methoden als `override` und ggf. `final` markiert werden sollten

Mit dieser Klassenhierarchie können einfache Berechnungen als Datenstruktur modelliert werden. Im Handout zu dieser Aufgabe finden Sie eine `main`-Funktion, die dies mit einer etwas komplexeren Formel demonstriert.

In der Hierarchie unterscheiden wir zwischen Operatoren ohne Parameter (`Reference`) und mit zwei Parametern (`Add`/`Subtract`/`Multiply`/`Divide`). `Reference` hat eine Member-Variable, um eine Referenz auf einen externen Wert zu speichern. Die anderen Operatoren zwei Operanden, die wiederum Operatoren sind und als `std::unique_ptr<Operator>` gespeichert werden. Wir bauen also eine Baumstruktur von Operationen auf, wie Sie sie in der Ausgabe unten sehen.

Um eine solche Berechnung auszuwerten, gibt es am `Operator` die Methode `compute()`, die die entsprechende Operation auswertet und das Ergebnis als `float` zurückgibt. Im Fall des `Binary_operator` müssen vorher die Operanden ausgewertet werden. Dies soll in der `compute()` Methode erfolgen, die mit den Ergebnissen der Operanden dann wiederum `compute_binary` aufrufen, welche dann in den abgeleiteten Klassen die eigentliche Berechnung durchführt.

Neben der Durchführung der Berechnung soll die Baumstruktur auch ausgegeben werden können. Hierzu ist die `print()` Methode vorgesehen, die zuerst die jeweilige Operation (bzw. bei Reference/Constant den Wert) ausgibt und dann ggf. die `print()` Methode ihrer Operanden aufruft.

Beachten Sie, dass `std::unique_ptr`, der zum Speichern der Operanden verwendet wird, nicht kopiert werden kann. Hierdurch können Sie die Klassen mit derartigen Member-Variablen ebenfalls nicht kopieren (was wir hier auch nicht benötigen) und Sie müssen bei Zuweisungen (z.B. im Konstruktor) `std::move` verwenden. Bei einem Konstruktor könnte dies z.B. so aussehen:

```
1 Binary_operator(std::unique_ptr<Operator> lhs, std::unique_ptr<Operator> rhs)
2   : lhs_(std::move(lhs)), rhs_(std::move(rhs)) {}
```

Die Ausgabe des fertigen Programms sollte so aussehen:

```
3
0.25

[/]
  [-]
    [+]
      [2]
      [/]
        [-9]
        [-9]
    [2]
  [*]
    [2]
    [2]
```