Fachhochschule
Dortmund
University of Applied Sciences and Arts

Prof. Dr. Dirk Wiesmann
Programmierkurs 2
Praktikum 3

Implementieren Sie ein *Dictionary* A unter Verwendung von *Hashing*. Ein *Dictionary* ist eine Menge A, die nur die folgenden Operationen unterstützt:

- `insert(a)`: $A \cup \{a\}$
- `delete(a)`: $A \setminus \{a\}$
- `member(a)`: `true` genau dann, wenn $a \in A$

Das *Dictionary* soll nur ganze Zahlen (`int`) verwalten. Verwenden Sie für die Implementierung das *geschlossene Hashing* (*open adressing*) auf der Basis eines Arrays. Das geschlossene Hashing muss mit m Behältern auskommen. Für eine Eingabe a berechnen Sie mit der Hashfunktion $h(a) = a \bmod m$ eine Behälternummer (den Arrayindex). Für die Operation `insert` speichern Sie dann die Zahl an der berechneten Arrayposition. Sie müssen keine Dubletten verhindern (d.h. eine Zahl darf mehrfach eingetragen werden). Falls eine Kollision auftritt (die berechnete Arrayposition ist bereits belegt), wird unter den übrigen Behältern nach einem freien Platz gesucht. Es gibt verschiedene Verfahren, die sich in der Reihenfolge der besuchten Behälter unterscheiden. Sie verwenden bitte das lineare Sondieren. Beim *linearen Sondieren* (*linear probing*) werden die übrigen Behälter der Reihe nach auf einen freien Platz untersucht. Die Betrachtung eines Behälters wird auch als „Probe“ bezeichnet. Bei der Suche nach einem Schlüssel müssen wir dann die folgenden Fälle unterscheiden:

1. Falls sich der Schlüssel im aufgesuchten Behälter befindet, ist die Suche erfolgreich.
2. Falls der aufgesuchte Behälter leer ist, befindet sich der Schlüssel nicht in der Hash-Tabelle.
3. Falls sich ein anderer Schlüssel in dem Behälter befindet, muss weitergesucht werden. Dabei muss am Ende der Tabelle die Suche am Anfang der Tabelle fortgeführt werden.

Das bedeutet aber auch, dass wir bei der *Delete*-Operation besondere Vorkehrungen treffen müssen. Wir können nicht einfach ein Element löschen, da Elemente, die später eingefügt wurden, auf der Suche nach einem freien Platz das zu löschende Element evtl. übersprungen haben. Da die Suche an einer freien Position stoppt, könnten wir diese Elemente dann später nicht mehr finden. Bitte kennzeichnen Sie die Behälter daher mit einem *Flag*. Dieses *Flag* soll den Status eines Behälters (z.B. frei oder gelöscht) repräsentieren. Die Suche überspringt dann einfach Behälter mit einem gelöschten Inhalt. Diese Behälter können später natürlich wieder gefüllt werden. Für die Darstellung des Status verwenden Sie bitte eine Aufzählung. Für die Arrayelemente definieren Sie bitte eine geeignete Struktur. Verwenden Sie zudem an geeigneten Stellen die Schlüsselwörter `typedef` und `static`. Wenn nicht genügend Speicherplatz zur Verfügung steht, dann liefert die Funktion `insert` den Wert 0, ansonsten den Wert 1. Die Funktion `delete`

gibt den Wert 1 zurück, wenn der Schlüssel a gelöscht werden konnte. Ansonsten wird der Wert 0 geliefert.
Für ein Array der Länge 2 sollte die folgende Sequenz z.B. fünfmal eine 1 ausgeben:

```
printf("%d\n", insert(1));
printf("%d\n", insert(3));
printf("%d\n", delete(3));
printf("%d\n", insert(5));
printf("%d\n", member(5));
```