ulm university universität
uulm

Institut für Numerische Mathematik

Dr. Andreas F. Borchert

26. April 2018
Blatt 2

Objektorientierte Programmierung mit C++ (SS 2018)

Abgabe bis zum 3. Mai 2018, 16:00 Uhr

Lernziele:

- Erste Erfahrungen mit Klassen in C++.

Aufgabe 1: Nim-Spiel

Nim ist ein Spiel für zwei Personen, bei dem auf n Haufen jeweils s_i Stäbchen verteilt sind. Die beiden Spieler ziehen alternierend und ein Spielzug besteht darin, zwischen einem bis maximal m Stäbchen von genau einem Haufen zu entfernen. Findet ein Spieler keine Stäbchen mehr zum Entfernen vor, hat er verloren.

Laden Sie für diese Aufgabe *Nim.tar.gz* von der Vorlesungswebseite herunter und packen Sie das Archiv mit Hilfe von *tar* aus. So könnte das funktionieren:

```
theon$ mkdir nim
theon$ cd nim
theon$ wget -O- -q \
> http://www.mathematik.uni-ulm.de/numerik/cpp/ss18/uebungen/01/Nim.tar.gz |
> gunzip | tar xf -
theon$ ls
Makefile  Nim.cpp  NimGame.cpp  NimGame.hpp  NimMove.cpp  NimMove.hpp
theon$
```

Sie finden dann eine halbfertige Umsetzung des Spiels vor. Die Klasse *NimMove* repräsentiert einen Spielzug, *NimGame* enthält die Datenstruktur und alle Methoden, um ein Nim-Spiel durchzuführen, und in *Nim.cpp* finden Sie ein darauf basierendes Hauptprogramm, das eine Partie zwischen einem menschlichen und einem Computer-Spieler durchführt. Die Klasse *UniformIntDistribution* aus *UniformIntDistribution.hpp* stellt einen Pseudo-Zufallszahlengenerator für ganze Zahlen zur Verfügung. Überall, wo Sie in den Quellen die Zeichenkette *FIXME* vorfinden, ist etwas zu ergänzen. Das betrifft die Klasse *NimGame*, bei der die

Mehrheit der Methoden umzusetzen ist, und den Teil des Hauptprogramms, bei der der Computer einen Spielzug aussucht, wenn er die Chance hat, zu gewinnen.

Für Nim-Spiele gibt es eine auf Patrick M. Grundy und Roland P. Sprague zurückgehende Theorie, die bei Nim-Spielen jeder Spielsituation eine nicht-negative sogenannte Nim-Zahl zuordnet.[1] Wenn der Wert 0 ist, dann hat der am Zug befindliche Spieler keine sichere Gewinnstrategie mehr. Bei einem positiven Nim-Wert gibt es mindestens einen Spielzug, der zum Gewinn führt. Bei dem vorgestellten Spiel werden zunächst die Nim-Werte für die einzelnen Haufen bestimmt und dann per Nim-Addition zu einem Wert aggregiert. Der Nim-Wert für einen einzelnen Haufen ist die Zahl der verbliebenen Stäbchen modulo der um eins erhöhten maximalen Zahl von Stäbchen, die in einem Zug entfernt werden dürfen (also $m + 1$). Die Nim-Addition entspricht dem bitweisen Exklusiv-Oder (also dem Operator „^“ in C++).

Beispiel: Angenommen, wir haben drei Haufen mit 9, 11 und 7 Stäbchen und es können maximal drei Stäbchen entfernt werden. Dann sind die Nim-Werte für die einzelnen Haufen 1, 3 und 3 und „1 ^ 3 ^ 3“ ergibt 1. In dieser Situation ist die Entfernung eines Stäbchens vom ersten Haufen ein guter Zug, da sich dann bei 8, 11 und 7 ein Nim-Wert von 0 ergibt.

Es ist empfehlenswert, mit der Fertigstellung von *NimGame.cpp* zu beginnen und dabei jeweils auf die Kommentare und die genannten Vorbedingungen in der zugehörigen Header-Datei *NimGame.hpp* Rücksicht zu nehmen. Mit **#include** <cassert> steht bereits *assert* zur Verfügung und davon sollte auch entsprechend Gebrauch gemacht werden. Bei der bereits implementierten Methode *set_heap_size* sehen Sie, wie auch ein übergebener Index mit *assert* überprüft werden kann. Bei der Methode *get_heap_size* können Sie das analog umsetzen.

Zur Datenstruktur eines *NimGame*-Objekts gehört die Spielkonfiguration *number_of_heaps* (Zahl der Haufen) und *maxtake* (maximale Zahl der Stäbchen, die in einem Zug entfernt werden dürfen. 0 bedeutet hier, dass beliebig viele von einem Haufen genommen werden dürfen). In dem Vektor *heap_size* wird die Zahl der verbliebenen Stäbchen in jedem der Haufen verwaltet. Es wird hierzu die STL-Container-Klasse *vector* benutzt, die ähnlich wie ein Array benutzbar ist. Beim bereits vorhandenen Konstruktor wird der Vektor passend dimensioniert. Die Variable *next_player* gibt jeweils an, wer als nächster Spieler am Zug ist. Am Ende eines Spieles handelt es sich dabei um den Verlierer.

Abgesehen von *NimGame.cpp* ist nur noch beim Hauptprogramm *Nim.cpp* der Teil zu ergänzen, der einen gewinnenden Zug ermittelt, wenn es einen solchen gibt. Hierzu können Sie systematisch alle zulässigen Züge auswerten, bis sie einen finden, der zu einem Nim-Wert von 0 führt. Den Test können Sie durchführen, indem Sie einfach das Spielobjekt kopieren und dann den zu testenden Zug auf der Kopie durchführen:

```
NimGame test = game; // make a copy of the current game
NimMove testmove(heap_index, count); // create a move
test.execute_move(testmove); // execute it
if (test.nim_value() == 0) {
   // winning move found ...
}
```

[1] Siehe Elwyn R. Berlekamp, John H. Conway und Richard K. Guy (2001): *Winning Ways for Your Mathematical Plays*, Volume 1, Second Edition, ISBN 1-56881-130-6, S. 81 ff.

Beim Eingeben des Spielzugs ist zuerst der Index des Haufens anzugeben (ab 0 beginnend) und dann die Zahl der Stäbchen. So könnte eine erfolgreiche Partie aussehen:

```
theon$ Nim
*** Game of Nim ***
Number of heaps: 3
Maximal number of sticks that can be taken in one move: 3
Heaps: 10 11 11
Your move: 0 2
Heaps: 8 11 11
Taking 1 from heap 0
Heaps: 7 11 11
Your move: 0 3
Heaps: 4 11 11
Taking 1 from heap 0
Heaps: 3 11 11
Your move: 0 3
Heaps: 0 11 11
Taking 1 from heap 1
Heaps: 0 10 11
Your move: 1 3
Heaps: 0 7 11
Taking 1 from heap 1
Heaps: 0 6 11
Your move: 1 3
Heaps: 0 3 11
Taking 1 from heap 1
Heaps: 0 2 11
Your move: 2 1
Heaps: 0 2 10
Taking 1 from heap 1
Heaps: 0 1 10
Your move: 2 1
Heaps: 0 1 9
Taking 1 from heap 1
Heaps: 0 0 9
Your move: 2 1
Heaps: 0 0 8
Taking 1 from heap 2
Heaps: 0 0 7
Your move: 2 3
Heaps: 0 0 4
Taking 1 from heap 2
Heaps: 0 0 3
Your move: 2 3
Congratulations!
theon$
```

Sie können Ihre Lösung mit *submit* auf der Theon einreichen:

```
theon$ submit cpp 1 Nim.cpp NimGame.cpp
```

Es steht Ihnen aber auch frei, die anderen Teile zu verändern und in aktualisierter Form mit einzureichen.

Aufgabe 2: Kleine Verbesserung des Nim-Spiels

Bei Ihrer in der ersten Aufgabe erarbeiteten Lösung wird für den Computer die erstbeste Zugmöglichkeit ausgesucht. Hier wäre es reizvoll, wenn aus allen geeigneten Spielzügen ein Spielzug gleichverteilt zufällig ausgewählt wird.

Entwerfen Sie hierzu eine Klasse *NimMoveSelector* mit den Methoden *add*, *get* und *get_count*. Mit *add* können beliebig viele Spielzüge hinzugefügt werden, mit *get_count* lässt sich die Zahl der mit *add* hinzugefügten Spielzüge abrufen und mit *get* wird genau ein Zug aus den zuvor hinzugefügten Zügen zufällig bestimmt und zurückgegeben. Die Methode *get* wird normalerweise nur ein einziges Mal aufgerufen und mehrfache Aufrufe von *get* dürfen entsprechend den gleichen Spielzug liefern.

Umsetzen lässt sich *NimMoveSelector* mit einer Trivialfassung des sogenannten *Reservoir-Sampling*-Algorithmus, der auf Alan G. Waterman zurückgeht. Im folgenden wird der Algorithmus beschrieben:[2]

Wenn n Einträge von einer Datei unbekannter Länge zufällig auszuwählen sind, werden die folgenden Variablen für den Algorithmus benötigt:

- $R[n]$, ein Array mit n Elementen, in dem die ausgewählten Einträge gespeichert werden (das Reservoir), ab 0 indiziert.
- t, die Anzahl der bereits von der Datei gelesenen Einträge, zu Beginn 0.

Reservoir-Sampling-Algorithmus:

1. Öffnen Sie die Datei und lesen Sie n Einträge ein und speichern Sie sie in R ab. Wenn es weniger Einträge gibt, brechen Sie an der Stelle ab. Ansonsten setzen Sie t auf n.
2. Lesen Sie den nächsten Eintrag aus der Datei ein. Wenn es keine weiteren Einträge mehr gibt, dann sind Sie fertig.
3. Erhöhen sie t um 1 und erzeugen Sie eine Zufallszahl M zwischen 0 und $t - 1$. Wenn $M \geq n$ ist, geht es mit Punkt 2 weiter.
4. Kopieren Sie den aktuellen Eintrag nach $R[M]$ (dabei wird ein früherer Eintrag überschrieben). Weiter geht es mit Punkt 2.

Dieses Verfahren vereinfacht sich natürlich bei $n = 1$, so dass statt einem Array eine einfache Variable genügt. Und wir lesen auch keine Datei aus – stattdessen werden die Elemente mit der *add*-Methode hinzugefügt, die dann die oben genannten Schritte 2 bis 4 durchführt.
Sie können Ihre Lösung wieder mit *submit* auf der Theon einreichen:

[2] Nach Donald E. Knuth: *The Art of Computer Programming*, Volume 2, S. 144: *Algorithm R*

```
theon$ submit cpp 2 Nim.cpp NimGame.cpp NimMoveSelector.hpp
```

Und wie zuvor steht es Ihnen frei, die sonst angepassten Teile einzureichen. Wenn Ihre Klasse *NimMoveSelector* nicht *header-only* ist, dann kann auch gerne *NimMoveSelector.cpp* hinzugefügt werden.

Viel Erfolg!