Universität Karlsruhe (TH)

Institut für Innovatives Rechnen und Programmstrukturen (IPD)

Real Life Programming (Praktikum im SS 2005) http://www.info.uni-karlsruhe.de/
Dipl.-Inf. Michael Beck beck@ipd.info.uni-karlsruhe.de
Dipl.-Inform. Rubino Geiß rubino@ipd.info.uni-karlsruhe.de
Dipl.-Inform. Sebastian Hack hack@ipd.info.uni-karlsruhe.de

Übungsblatt 1 | Ausgabe: 13.04.2005 | Besprechung: 20.04.2005

Aufgabe 1: *Spielregeln*

- Für jede der sieben Aufgaben werden maximal 10 Punkte vergeben.
- Es herrscht Anwesenheitspflicht. Pro unentschuldigtem Fehlen werden 5 Punkte abgezogen. Nur ärztliche Atteste gelten als Entschuldigung.
- Die Studenten sollen sich zu Gruppen zusammenschließen. Die Gruppengröße ist 3.
- Jede Gruppe, die einen Vortrag hält, kann bis zu 10 Punkte erhalten.
- Zum Erlangen des Scheins sind 50% der erreichbaren Punkte sowie das Bearbeiten von mindestens 80% der Aufgaben notwendig.
- Da wir sehr viele Anmeldungen haben, ist es unfair einen Praktikumsplatz zu belegen, wenn man nicht zur aktiven Teilnahme bereit ist und ausreichend Zeit (>8h pro Woche) eingeplant hat.

Aufgabe 2: *Vorträge*

Im Laufe des Praktikums sollen Vorträge zu den einzelnen Schwerpunkten gehalten werden. Bis auf den ersten Vortrag sollen diese von den Studenten (jeweils eine Gruppe) bestritten werden. Die Vorträge sollen ca. 45 bis 60 min dauern. Die Wahl der Mittel ist Euch überlassen, eine Zusammenfassung (max. 2 A4-Seiten) oder geignete Literaturverweise sind notwendig. Nachstehend sind die geplanten Vorträge in Form von Stichpunkten gelistet. Ihr könnt selbstverständlich auch weitere Punkte besprechen, allerdings sind die angegebenen Stichpunkte als Mindestumfang anzusehen. Die Vortragsthemen:

2.1 *C für (Java-)Programmierer (Referenten: Beck, Geiß, Hack)*

- Pointer, dangling Pointer
- Strings
- Structs/Unions
- malloc/alloca
- Stack-Variable
- Header-Files
- Typsystem
- Modifier/Storage Classes
- Dialekte
- Compileraufruf/make

2.2 *Speicherverwaltung mit C*

- Wie funktionieren malloc/free/realloc?

- Wie funktioniert alloca?
- Wie funktionieren obstacks?
- Wie funktioniert brk?
- Wie schreibt man eine eigene Speicherverwaltung?
- Wie geht Garbage Collection mit C?
- Speicherfehlersuche? Löcher, Seitenfehler/Segmentation Fault

2.3 *Die C-Stdlib*

- Was ist Posix?
- Was gibt es noch?
- Braucht C eine stdlib?
- Wie sehen Programme „ohne“ stdlib aus? Braucht man sowas?
- stdio, wie funktioniert das?
- Strings in C

2.4 *C nach Maschine*

- Wie sieht eine allgemeine RISC-Architektur aus?
- Was ist ein Stack-Frame?
- Was ist ein Code-, Daten-, BSS-Segment?
- Wie funktioniert Parameterübergabe/Rückgabewert?
- Was ist die ABI?
- Wie werden Konstrukte wie if/for/while/switch abgebildet?
- Wie sehen Pointer/Pointerarithmetik aus?
- Was ist Zweierkomplement, IEEE754, andere?
- Kann man C auf einer VM ausführen? Auf der JVM?

2.5 *Handwerkszeug*

- Wie funktioniert der Präprozessor?
  - Wozu kann man ihn außer #ifdef benutzen?
  - Was sollte man vermeiden?
- cvs, subversion
- diff, patch
- Make: Das build-System?
- automake, autoconf
- doxygen, javadoc

2.6 *Profiler*

- Was ist ein Profiler?
  - Wie funktioniert invasives Profiling? Warum ist das schlecht?
  - Wie funktioniert nicht-invasives Profiling? Was ist oprofile / vtune?
- Compilerflags beim Profilen
- valgrind, kcachegrind, und callgrind/calltree
- Prozessorunterstützung beim Profiling

2.7 *Debugging*

- Was ist ein Debugger?
- Wie funktioniert Debugging?
  - Was sind Breakpoints/Watchpoints?
  - Was ist Debuginfo?
  - Was ist -O2 -g?
  - Warum „sieht“ man manchmal falsche Werte?
  - Was ist Debug-Mode/Release-Mode?
- ddd, insight
- valgrind
- Wie funktioniert assert()? Wie darf man assert NICHT verwenden?
- Debug-Output: Wie macht man das „richtig“?

2.8 *Buffer-Overflow*

- Was ist ein Buffer-Overflow?
- Wie kann er entstehen, typische Anwendungsfälle.
- Wie kann man Buffer-Overflows detektieren?
- Passive Programmierung: Wie vermeidet man BO?
- Gefahren: Wie kann man BO „ausnutzen“?
- Wie funktionieren „Anti-BO-Tricks“?

2.9 *OO und C*

- Wie werden OO-Strukturen in Maschine dargestellt?
  - vtable/Polymorphe Rufe
  - Vererbung
  - Overloading
  - Name-Mangling
  - Runtime-Type-Info
- Wie stellt man diese Konzepte in C dar?

2.10 *API-Bindings*

- Wie kann man C-funktionen an andere Sprachen anbinden?
  - JNI
  - Python
- Warum geht das überhaupt?
- Warum gibt es kein CNI?

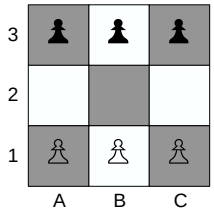
2.11 *C++*

- (Mehrfach-)Vererbung
- Operatorvereinfachung
- Templates, STL
- C++ = Java ohne Pointer ?

Aufgabe 3: *Bauernschach (10+X Punkte)*

Als kleine Anwendung wollen wir zunächst ein einfaches Spiel programmieren:

Unter Bauernschach versteht man ein Schachbrett, auf dem nur Bauern stehen. Gewonnen hat, wer als erster eine Dame bekommt oder wer den letzten Zug machen kann, ein Patt gibt es also nicht. Wir wollen dieses Spiel auf einem 3x3 Brett spielen, die Ausgangsstellung ist also



Der Computer soll eine Seite übernehmen und durch „Lernen“ immer besser spielen. Dies läßt sich wie folgt implementieren:

Die möglichen Spielstellungen sowie Züge bilden einen azyklischen Graphen. Der Wurzelknoten des Graphen ist die Ausgangsstellung. Die Endstellungen sind Knoten, von dehnen keine Kanten ausgehen. Der Computer navigiert in diesem Graphen. Hat ein Knoten nur einen möglichen Nachfolger, wird dieser genommen, anderenfalls wird aus der Menge der Nachfolger per Zufall einer ausgewählt. Wenn der Computer verliert, streicht er einfach die letzte Kante, die per Zufall gewählt wurde, diese Auswahl war offenbar „schlecht“. Nach nur wenigen Spielen bleiben so nur noch Kanten übrig, die zu potentiellen Gewinnsituationen führen.

Der initiale Graph kann entweder statisch eingegeben werden (bei 3x3 Feldern gibt es nicht soviele Knoten wenn man Spiegelung ignoriert) oder auch zur Laufzeit anhand von gültigen Zügen berechnet werden. Wenn der Computer sowohl weiß, als auch schwarz spielen soll, benötigt man eigentlich zwei Graphen, man kann diese aber auch verschmelzen (Vorsicht! Kanten dürfen dann nicht einfach gestrichen werden).

Die Ausgabe soll einfach als ASCII-Text erfolgen, die Eingabe in Form einer einfachen Schachnotation, z.B. `A1-A2`.

Extra Punkte gibt es für eine besonders „C-artige“ Implementation.

Hinweis: „C-artig“ meint NICHT unleserlich!

Aufgabe 4: *Infrastruktur*

Mache Dich mit der Infrastruktur bekannt. Zu klären sind hier Deine persönlichen Fragen, Probleme und Vorlieben im Umgang mit den Rechnern und Einrichtungen es IPD. Die nachfolgende Liste ist ein nicht immer ganz ernst gemeintes Beispiel.

- Wie funktioniert das mit dem login? Wie kommt man von zu Hause an die Rechner ran?
- Wir müssen ein Passwort für unser Gruppe wählen *und* setzen. Wie geht das nochmal... `passwd` ist es nicht!
- Wie druckt man?
- Wie ist das mit dem Plattenplatz? Passen die mp3 noch drauf?
- Rechtevergabe? Warum tut chmod nicht?
- Was ist nochmal AFS?
- Ich möchte um 07:30 Uhr in den Poolraum!

Folgende Dokumente sollten Dir dabei helfen:

- Erste Schritte am IPD: `https://admin.info.uni-karlsruhe.de/user.pdf`
- Eine Einführung in die AFS Benutzung: `https://admin.info.uni-karlsruhe.de/afs-einfuehrung.ps`

Falls du persönliche Hilfe brauchst, wende Dich entweder an das Praktikumsteam oder bei Fragen zur technischen Infrastruktur an Bernd Traub (Zimmer 205).