אוניברסיטת בן-גוריון בנגב
הפקולטה למדעי הטבע - המחלקה למדעי המחשב
סמסטר א' תשפ"ו

סילבוס קורס

שם קורס:	עקרונות הקומפילציה
שם קורס באנגלית:	Compiler Principles
מספר קורס:	202-1-3021
סוג קורס:	בחירת חובה
נק"ז:	5
דרישות קדם:	עקרונות שפות תכנות, מעבדה בתכנות מערכות

מבוא לבניית קומפיילרים

התורה של ניתוח תחבירי של תוכניות מחשב
פרקי השלמה בתורת שפות תכנות
מודל קומפילציה מבוסס continuations
אנליזה סמנטית
יצירת קוד (code generation)
סביבת הריצה
אופטימיזציות

סילבוס באנגלית

אוניברסיטת בן-גוריון בנגב
הפקולטה למדעי הטבע - המחלקה למדעי המחשב
סמסטר א' תשפ"ו

Introduction to Compiler Construction

- The algebraic relationship between compilation & interpretation.
- Cross-compilation, boot strapping a compiler, de-compilation.
- The stages of the compiler: What work is done in each, what kinds of errors can and cannot be detected at each, the basic algorithms that are implemented at each stage.
- Dynamic vs statically-typed languages. Early binding vs late binding. The information available to the compiler for translation, error detection, and optimizations.

Scanning & Parsing Theory

- Scanner: DFA, NDFA, NDFA with epsilon-transitions
- Parsing: Top-down, recursive descent parsers, parsing combinators, bottom-up parsers
- Hand-coding various parsers
- Using parser-generation tools in C & Java
- Macro expansion: Syntactic transformations, reduction to core forms in the language, variables, meta-variables and syntactic hygiene.

Programming Languagues

- Functional vs Imperative programming: How change & side-effects are understood & modeled in the functional view of programming.
- Parameter-passing mechanisms: Call-by-value, call-by-reference, call-by-sharing/object, call-by-name, call-by-need. Features of the various parameter-passing mechanisms, their motivation, history & implementation. - Scope & its implementation: Dynamic scope (deep binding, shallow binding), lexical scope. Dynamic scope and the implementation of exception handling.
- The structure of the lexical environment, and the implications for data sharing & side effects.
- Object-oriented vs functional programming Languagues. The structure of the closure compared to that of the object. Mapping of lambda-expressions to objects. The virtual method table.
- Monads & monadic programming.

Continuation-Passing Style (CPS)

- CPS as a programming technique (multiple return values, multiple continuations, co-routines, implementation of threads.

- CPS as an approach to writing a compiler: CPS, defunctionalization of the continuation, stack machine.
- CPS as an intermediate language for the compiler: Optimizations that are simpler in CPS.

Semantic Analysis

- Lexical addressing, deBruijn numbering
- Identification of tail calls
- Boxing, data indirection, and motion from the stack to the heap: A comparison between quasi-functional programming languages (Scheme, LISP) and object oriented programming languages (Java).

Code Generation

- Layout of Scheme objects in memory. Run-time type information. Comparison with the situation in object-oriented programming languages
- An overview of the proof of correctness of the compiler, and how it is constructed along with the code generator.
- Optimization of tail calls
- Code generation to native x86 instructions for the various expressions in our language
- The primitive procedures & support code that are provided with the compiler

The Run-Time Environment

- The top level: n-LISP -- value cells, function cells, property cells, etc.
Dynamic memory management
  - Reference counting
  - Garbage collection: mark & sweep, stop & copy, generational garbage collection
- Namespaces, modules, and their implementation

Compiler Optimizations

- The tail-recursion & tail-call optimizations
- Loop optimizations & transformations
- Array optimizations
- Strength reduction optimizations
- Dead-code removal, write-after-write optimizations
- Common Sub-expression Elimination, both as a high-level and low-level optimization
- Optimizations for super-pipelined and parallel architectures

אוניברסיטת בן-גוריון בנגב
הפקולטה למדעי הטבע - המחלקה למדעי המחשב
סמסטר א' תשפ"ו

דרישות ומרכיבי ציון הקורס

3 מטלות בית (3% סה"כ), שני בחנים (18% סה"כ), פרוייקט סופי (רכיב חובה, 19%), ומבחן סיום (רכיב חובה, 60%). עבודה שתתבטל תעלה באופן פרופורציוני את ערך העבודות באותו רכיב. רכיב שיבוטל יעלה באופן פרופורציוני את ערך הרכיבים הנותרים. יש לעבור את רכיבי החובה בקורס בציון 56 ומעלה על מנת לעבור את הקורס.

הערכה חלופית למילואימניקים

תלמידים המשרתים בכוחות הביטחון (מילואימניקים, עתודאים, חיילים בקבע, וכו', כולל תלמידים שמתמודדים עם פציעות וקשיים אחרים בעקבות השירות שלהם) זכאים להערכה חלופית בקורס, בהתאם לתנאי השירות שלהם ולמצבם. ההערכה כוללת מועדים חלופיים להגשת עבודות בית, ובחינות חלופיות במועדים שיקבעו לפי תנאי השירות שלהם.

ספרות הקורס

1. Modern Compiler Design, by D. Grune, H. Bal, C. Jacobs, K. Langendoen
2. LISP in Small Pieces, by Christian Queinnec
3. The Anatomy of LISP, by John Allen
4. The Structure & Interpretation of Computer Programs, by Harold Abelson, et al.
5. Essentials of Programming Languages, by Daniel P. Friedman, Mitchell Wand, Christopher T. Haynes