# Przedmowa

W czasie, który upłynął od roku 1986, roku pierwszego wydania tej książki, świat projektowania kompilatorów znacząco się zmienił. Ewolucja języków programowania stworzyła nowe problemy. Architektury komputerów oferują dziś bogactwo zasobów, które projektant kompilatora powinien, a w zasadzie musi wykorzystać. Być może najbardziej interesujące jest to, że szanowane techniki optymalizowania kodu znalazły zastosowania poza kompilatorami. Są dziś używane w narzędziach wyszukujących błędy, a co najważniejsze, luki zabezpieczeń w już istniejącym oprogramowaniu. Zarazem większość technologii „przodowych" – gramatyki, wyrażenia regularne, parsery i translatory sterowane składnią – nadal jest w szerokim użyciu.

Tym samym nasza filozofia prezentowana w poprzednich wersjach tej książki się nie zmieniła. Zdajemy sobie sprawę, że bardzo nieliczni spośród czytelników będą tworzyć, lub choćby utrzymywać, kompilatory dla któregoś z głównych języków programowania. Jednak modele, teoria i algorytmy powiązane z kompilatorami mogą być stosowane w szerokim zakresie problemów projektowania i rozwijania oprogramowania. Dlatego szczególnie wyróżniamy kwestie do rozstrzygnięcia, które są najczęściej spotykane przy projektowaniu procesorów języków, niezależnie od języka źródłowego czy maszyny docelowej.

## Korzystanie z książki

Opanowanie całości czy choć większości materiału z tej książki wymaga co najmniej dwóch kwartałów, a może nawet dwóch semestrów. Typowe podejście polega na przedstawieniu pierwszej połowy w ramach wykładu podstawowego, drugą zaś połowę tematyki książki – optymalizowanie kodu – na poziomie dyplomowym lub pośrednim. Oto konspekt poszczególnych rozdziałów:

W **rozdziale 1** zawarto materiały motywujące, a ponadto przedstawiono w nim kilka podstawowych zagadnień architektury komputerów i zasady języków programowania.

W **rozdziale 2** pokazano projektowanie miniaturowego kompilatora i wprowadzono wiele ważnych koncepcji, które zostaną rozwinięte w kolejnych rozdziałach. Sam kompilator został w całości zamieszczony w dodatku na końcu książki.

W **rozdziale 3** zawarto omówienie analizy leksykalnej, wyrażeń regularnych, automatów skończonych i narzędzi generujących leksery. Materiał ten jest podstawą do przetwarzania tekstów dowolnego rodzaju.

W **rozdziale 4** zaprezentowano główne metody parsingu: zstępujące (metoda zejść rekurencyjnych LL) i wstępujące (LR i jej warianty).

W **rozdziale 5** wprowadzono podstawowe koncepcje definicji kierowanych składnią i translacji sterowanej składnią.

W **rozdziale 6** rozwinięto teorię z rozdziału 5 i pokazano, jak można ją wykorzystać do generowania kodu pośredniego dla typowego języka programowania.

W **rozdziale 7** skupiono się na środowiskach wykonawczych, ze szczególnym naciskiem na zarządzanie stosem w czasie wykonania i mechanizmy odśmiecania pamięci.

W **rozdziale 8** omówiono generowanie kodu wynikowego. Obejmuje ono konstruowanie bloków podstawowych, generowanie kodu dla wyrażeń i bloków podstawowych oraz techniki alokowania rejestrów.

W **rozdziale 9** wprowadzono technologie optymalizacji kodu, w tym grafy przepływu, problemy przepływu danych i iteracyjne algorytmy rozwiązywania tych problemów.

W **rozdziale 10** zaprezentowano optymalizacje na poziomie instrukcji. Główny nacisk został tu położony na możliwości wydobycia równoległości z małych sekwencji instrukcji i szeregowania ich na pojedynczych procesorach, które są w stanie wykonywać więcej niż jedną czynność naraz.

W **rozdziale 11** omówiono wykrywanie i wykorzystywanie równoległości w większej skali. W tym miejscu skupiono uwagę na programach numerycznych mogących zawierać wiele ciasnych pętli przebiegających wielowymiarowe tablice.

W **rozdziale 12** zajęto się analizami międzyproceduralnymi. Omówiono tu analizy wskaźników, aliasowania oraz przepływu danych, uwzględniające sekwencje wywołań procedur, które osiągają dany punkt w kodzie.

Wykłady oparte na materiale zawartym w tej książce były prowadzone na uniwersytetach Columbia, Harvard i Stanford. Na Uniwersytecie Columbia regularnie proponowany jest wykład dla pierwszego roku studiów wyższego poziomu na temat języków programowania i translatorów, wykorzystujący materiał z pierwszych ośmiu rozdziałów. Cechę wyróżniającą tego wykładu stanowi trwający semestr projekt, w którym studenci w małych zespołach pracują nad utworzeniem i implementacją prostego języka własnego projektu. Tworzone przez nich języki obejmują wielką rozmaitość zastosowań, w tym obliczenia kwantowe, syntezowanie muzyki, grafikę komputerową, gry, operacje na macierzach i wiele innych obszarów. Do zbudowania swoich własnych kompilatorów studenci wykorzystują generatory komponentów kompilatorów, takie jak ANTLR, Lex lub Yacc, oraz techniki translacji sterowanej składnią omówione w rozdziałach 2 i 5. Następujący później zaawansowany wykład skupia się na materiałach rozdziałów 9–12, z wyróżnieniem generowania i optymalizacji kodu dla nowoczesnych maszyn, w tym procesorów sieciowych i architektur wieloprocesorowych.

Na Uniwersytecie Stanforda kwartalny wykład wprowadzający obejmuje w przybliżeniu materiał z rozdziałów 1–8, choć znajduje się w nim wprowadzenie do globalnych technik optymalizacyjnych z rozdziału 9. Drugi wykład obejmuje treść rozdziałów 9–12 oraz bardziej zaawansowany materiał dotyczący odśmie-

cania pamięci z rozdziału 7. Studenci wykorzystują opracowany na miejscu, oparty na Javie, system o nazwie Joeq do implementowania algorytmów analizy przepływu danych.

## Wymagania wstępne

Czytelnik powinien dysponować pewnym „wyrafinowaniem informatycznym", obejmującym co najmniej zaawansowany wykład na temat programowania oraz wykłady ze struktur danych i matematyki dyskretnej. Przydatna będzie także znajomość kilku różnych języków programowania.

## Ćwiczenia

W książce zawarto rozbudowane ćwiczenia, po kilka dla niemal każdego podrozdziału. Trudniejsze ćwiczenia lub ich części zostały wyróżnione wykrzyknikiem. Najtrudniejsze ćwiczenia oznaczono podwójnym wykrzyknikiem.

## Podziękowania

Jon Bentley dostarczył wyczerpujących komentarzy o wielu rozdziałach wcześniejszego szkicu tej książki. Pomocne komentarze i poprawki dostarczyli również (w kolejności alfabetycznej): Domenico Bianculli, Peter Bosch, Marcio Buss, Marc Eaddy, Stephen Edwards, Vibhav Garg, Kim Hazelwood, Gaurav Kc, Wei Li, Mike Smith, Art Stamness, Krysta Svore, Olivier Tardieu oraz Jia Zeng. Jesteśmy bardzo wdzięczni za pomoc tych wszystkich osób. Wina za wszelkie pozostawione błędy oczywiście spoczywa na nas.

Dodatkowo Monica chciałaby podziękować swoim koleżankom i kolegom z zespołu kompilatora SUIF za 18-letnią naukę kompilowania, są to: Gerald Aigner, Dzintars Avots, Saman Amarasinghe, Jennifer Anderson, Michael Carbin, Gerald Cheong, Amer Diwan, Robert French, Anwar Ghuloum, Mary Hall, John Hennessy, David Heine, Shih-Wei Liao, Amy Lim, Benjamin Livshits, Michael Martin, Dror Maydan, Todd Mowry, Brian Murphy, Jerey Oplinger, Karen Pieper, Martin Rinard, Olatunji Ruwase, Constantine Sapuntzakis, Patrick Sathyanathan, Michael Smith, Steven Tjiang, Chau-Wen Tseng, Christopher Unkel, John Whaley, Robert Wilson, Christopher Wilson oraz Michael Wolf.

*A.V.A.*, Chatham NJ
*M.S.L.*, Menlo Park CA
*R.S.*, Far Hills NJ
*J.D.U.*, Stanford CA

Czerwiec 2006 r.