Metody kompilacji

Zbiorcza lista zadań ze wszystkich laboratoriów

Piotr Błaszyński

6 maja 2018

Zadania:

- pobrać wszystkie pliki z przykładami: `http://detox.wi.ps.pl/pb/tk/wszystkieZ.zip`
- przetestować kompilację i uruchomienie kodu z katalogów z1, z2, z3 (instrukcja na końcu),
- uruchomienie polega na podaniu jako argumentu pliku inX.txt, pliki znajdują się w odpowiednich katalogach,
- dodatkowo przetestować pracę w trybie interaktywnym,
- zmodyfikować pliki wejściowe w z2, tak aby się „kompilowały" obydwa, ew. można zmodyfikować plik z analizatorem (z2.l), jest to zadanie na zapoznanie się ze szkieletem budowy analizatora leksykalnego, należy sprawdzić jakie leksemy obsługuje zaprezentowany analizator leksykalny, i usunąć z pliku źródłowego nieobsługiwane elementy,
- zapoznać się z treścią Makefile w z3,
- pobrać emulator procesora MIPS - MARS `http://courses.missouristate.edu/KenVollmar/mars/download.htm` (alternatywnie QtSpim), uruchamianie emulatora poprzez maszynę wirtualną javy (przykład na końcu),
- uruchomić przykładowy program `http://courses.missouristate.edu/KenVollmar/mars/CCSC-CP%20material/row-major.asm`
- przygotować gramatykę dla pojedynczego wyrażenia składającego się ze zmiennych, liczb i operatorów (plik def.y - przykładowa gramatyka w katalogu z5),
- przekazać wartości semantyczne identyfikatorów i liczb (całkowitych i rzeczywistych) z analizatora leksykalnego do analizatora składniowego,
- zaimplementować funkcję main (tylko jedna wersja w def.y), z obsługą parametrów wywołania (argc, argv),
- zapisać do pliku wartości semantyczne poszczególnych identyfikatorów i liczb oraz operatory w kolejności dopasowywania,

- przetestować działanie dla wyrażeń składających się z kilku (8-10) elementów,
- przygotować liste leksemów w analizatorze leksykalnym (plik zX.l)
- przygotować listę tokenów w części początkowej analizatora składniowego (plik def.y),
- w analizatorze leksykalnym dopisać zwracanie tokenów - dla pojedynczych symboli jest to kod ASCII symbolu, dla słów kluczowych, liczb, identyfikatorów i symboli wieloznakowych jest kod tokenu,
- zmodyfikować pliki źródłowe i plik Makefile, tak, by analizator składniowy i cały kompilator był kompilowany przy pomocy kompilatora C++,
- zmodyfikować gramatykę tak, by można było kompilować program składający się z wielu wyrażeń (wielu linii),
- dopisać kod zapisujący trójki do pliku,
- dodać opcję kompilacji -std=c++11 do Makefile (w miejscach gdzie używamy g++),
- przygotować regułę dla symbolu nieterminalnego przedstawiającego przypisanie (jeżeli wcześniej tego nie zrobiliśmy),
- utworzyć tablicę symboli (zapisywać w niej wszystkie identyfikatory),
- generować zmienne tymczasowe do przechowywania wyników trójek,
- dla każdej trójki generować 4 linie kodu asemblera i przechowywać je w wektorze,
- dopisać kod zapisujący linie z vectora do pliku *yyout*, wywołać ten kod po *yyparse*,
- po *yyparse* zapisać tablicę symboli do pliku *symbols.txt*,
- zapisać symbole z tablicy symboli przed kodem w bloku danych,
- dodać kompilację instrukcji do wprowadzania i wypisywania wartości,
  - reguły gramatyki,
  - generowanie kodu wynikowego,
- dodać kompilację instrukcji warunkowych,
  - reguły gramatyki,
  - generowanie kodu wynikowego,
- dodać kompilację instrukcji pętli,
  - reguły gramatyki,
  - generowanie kodu wynikowego,
- dodać kompilację tworzenia i używania statycznych tablic jednowymiarowych,
  - reguły gramatyki,
  - generowanie kodu wynikowego,

- dodać obsługę liczb zmiennoprzecinkowych,
  - generowanie kodu obliczeń,
  - konwersja lub zgłoszenie błędu,
- dodać obsługę statycznych tablic wielowymiarowych,
  - reguły gramatyki,
  - tworzenie wpisu w tablicy symboli,
  - obliczanie indeksu,
- dodać kompilację deklaracji i wywołania prostych funkcji,
  - reguły gramatyki,
  - generowanie kodu wynikowego,
- dodać kompilację instrukcji tworzenia dynamicznych tablic jednowymiarowych,
  - reguły gramatyki,
  - generowanie kodu wynikowego,
- dodać kompilację instrukcji do obsługi struktur (ew. klas),
  - reguły gramatyki,
  - generowanie kodu wynikowego,