WI

Metody kompilacji

Wykład 1, Wprowadzenie

Włodzimierz Bielecki, Piotr Błaszyński

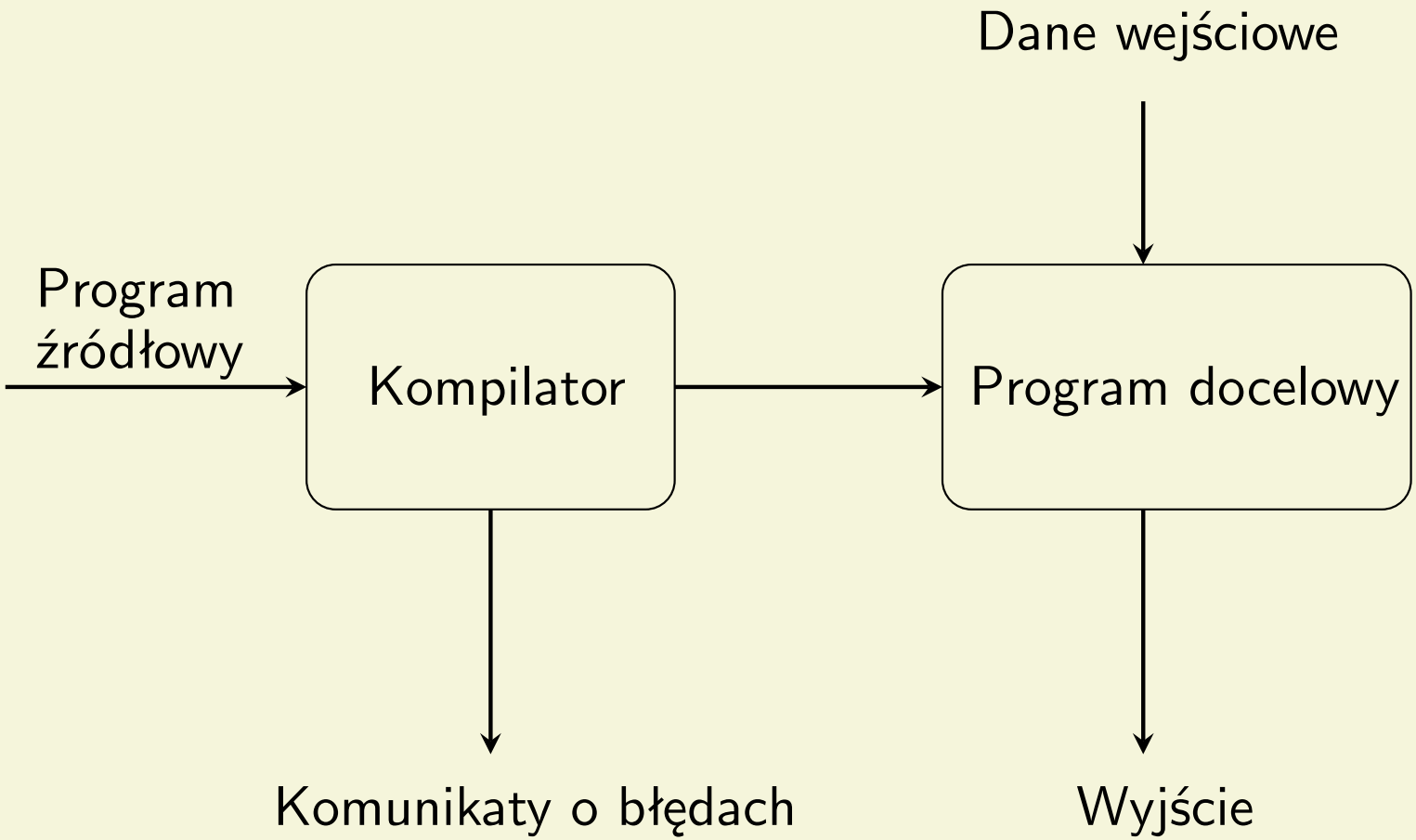
Wydział Informatyki Zachodniopomorskiego Uniwersytetu Technologicznego

2 marca 2020

Kompilator

Kompilator jest to program, który odczytuje program w języku źródłowym i tłumaczy go na program równoważny w języku docelowym. Interpreter, zamiast produkowania programu docelowego, bezpośrednio wykonuje czynności określone w programie źródłowym. Program docelowy, produkowany przez kompilator, jest zwykle znacznie szybszy niż proces produkowania wyniku przez interpreter. Rysunek na nastepnym slajdzie pokazuje schemat działania kompilatora.

Schemat działania kompilatora

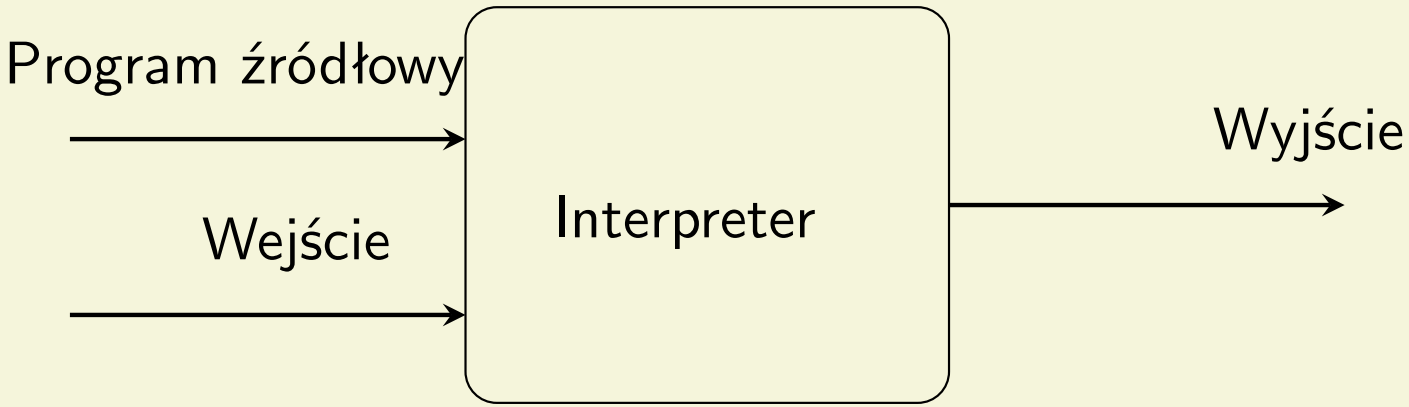


Rysunek: Schemat działania kompilatora

Interpreter

Interpreter jednak zazwyczaj daje lepszą diagnostykę błędów niż kompilator, ponieważ wykonuje instrukcję programu źródłowego instrukcja po instrukcji. Schemat działania interpretera pokazuje rysunek na kolejnym slajdzie.

Schemat działania interpretera



Rysunek: Schemat działania interpretera

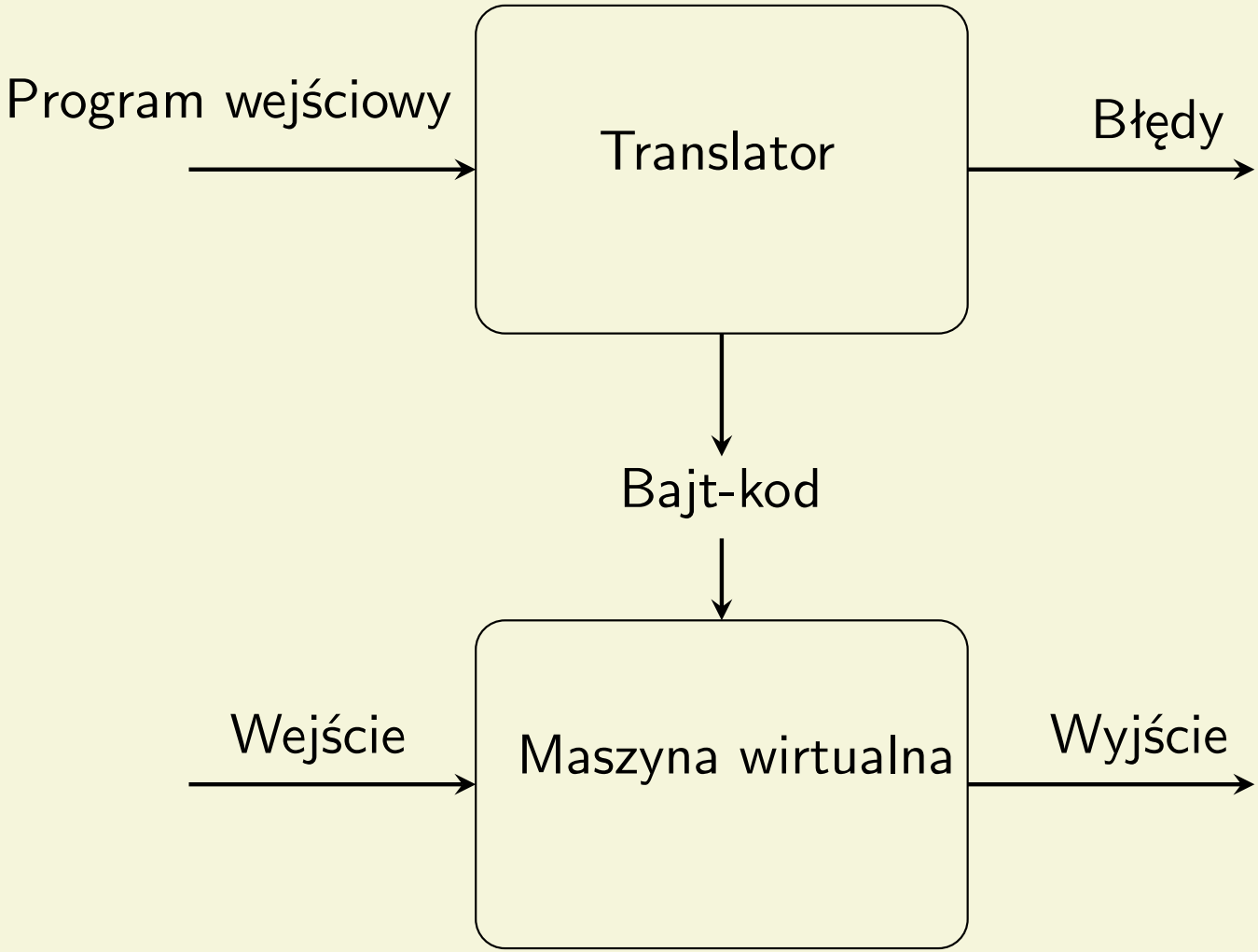
Zgłaszanie błędów

Ważną rolą kompilatora jest zgłaszanie wszelkich błędów w programie źródłowym, które są wykrywane podczas procesu tłumaczenia.

Kompilacja hybrydowa

Wirtualne procesory języka Java łączą kompilację i interpretację. Program źródłowy w języku Java jest najpierw kompilowany do postaci pośredniej zwanej bajtkodami (*bytecodes*). Bajtkody następnie są interpretowane przez maszynę wirtualną. Na następnym slajdzie pokazany jest schemat działania kompilatora hybrydowego.

Schemat działania kompilatora hybrydowego

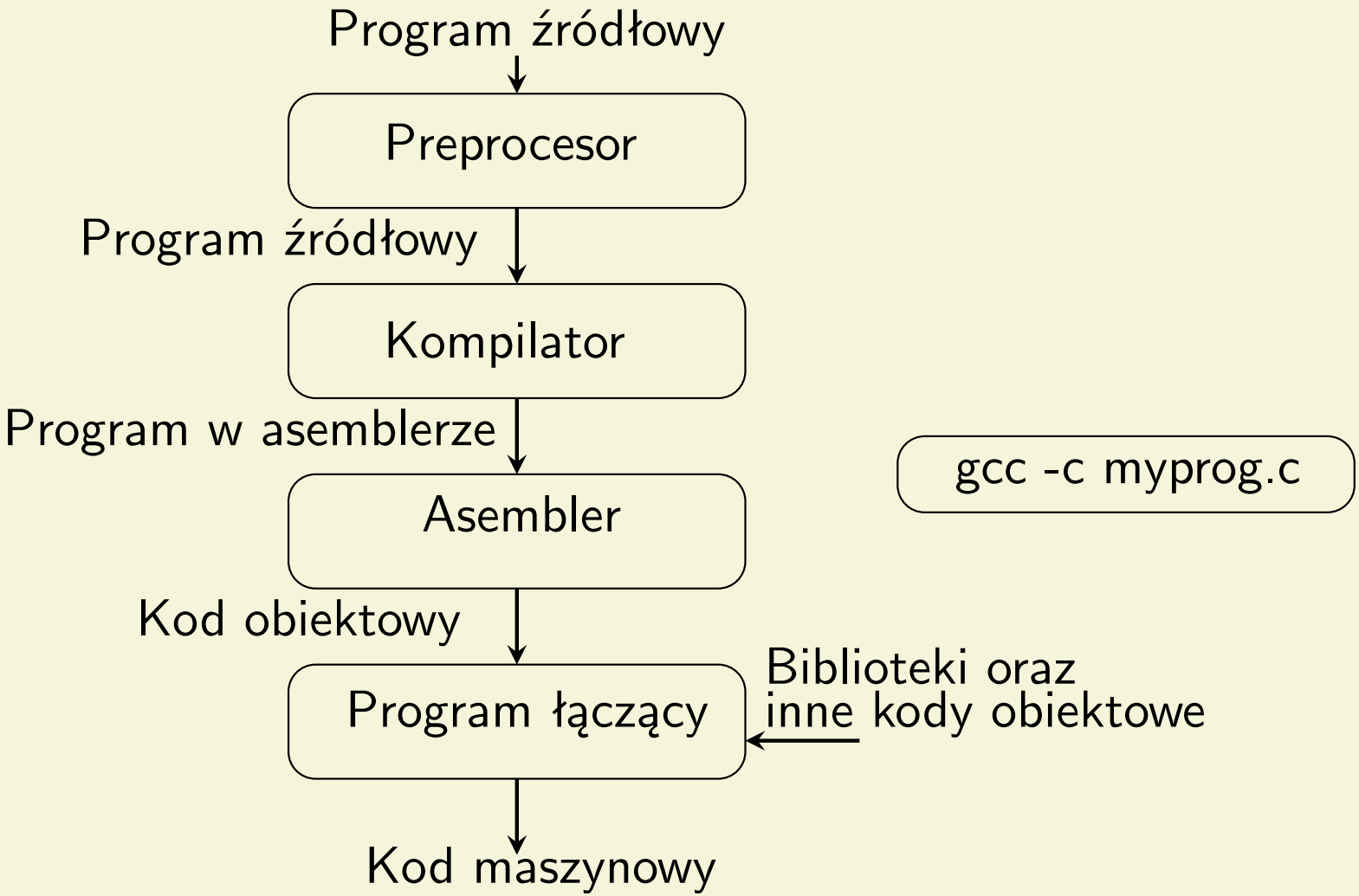


Rysunek: Schemat działania kompilatora hybrydowego

Schematy działania kompilatora

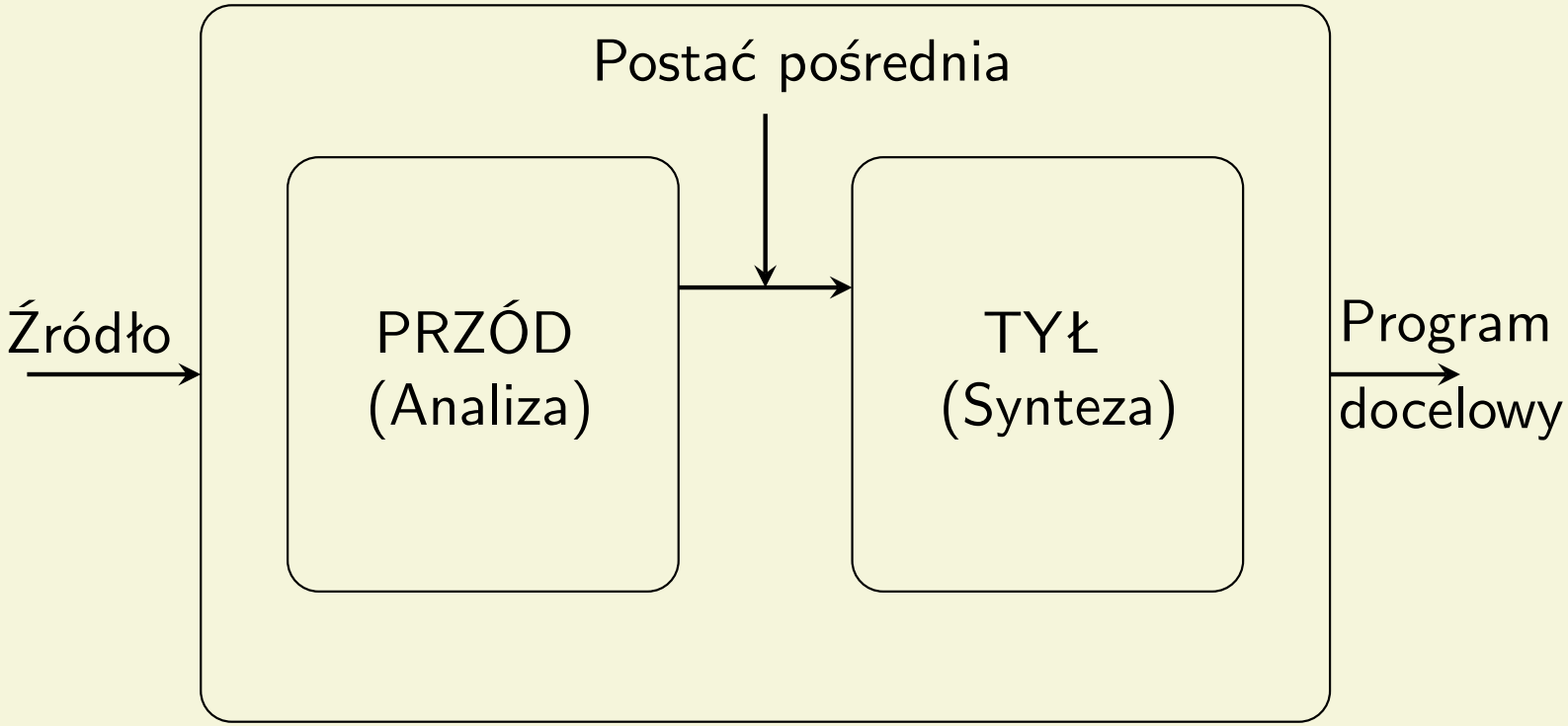
Oprócz kompilatora może być potrzebnych kilka innych programów, aby utworzyć wykonywalny program docelowy. Kolejność przetwarzania programu źródłowego w kod maszynowy pokazuje rysunek na następnym slajdzie. Ogólna architektura kompilatora (podział na przód i tył) pokazana jest na kolejnym rysunku. Natomiast szczegóły budowy kompilatora, z podziałem na poszczególne elementy, pokazano na trzecim z prezentowanych rysunków.

Kolejność przetwarzania kodu źródłowego



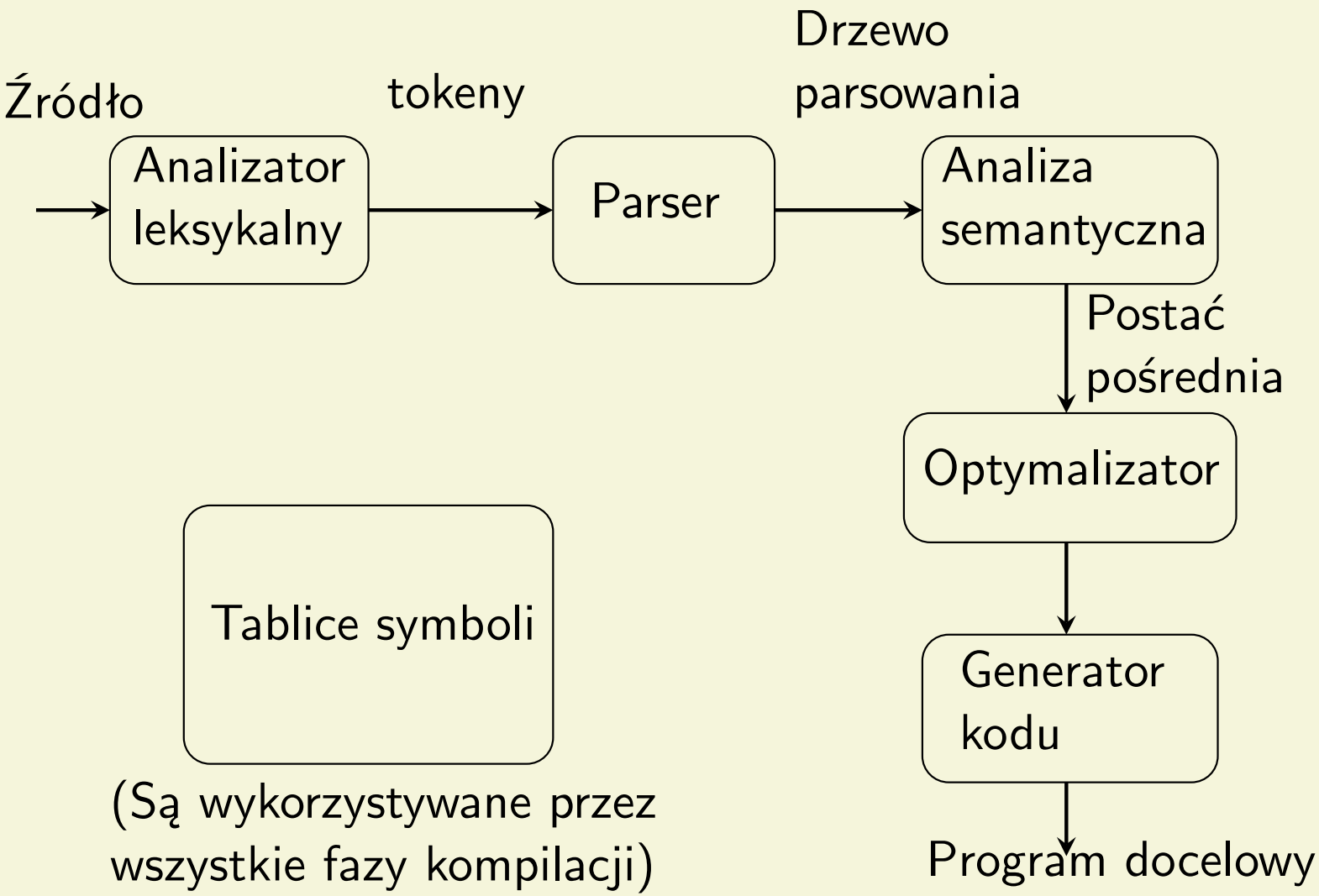
Rysunek: Schemat działania kompilatora – kolejność przetwarzania kodu źródłowego

Architektura kompilatora



Rysunek: Architektura kompilatora

Architektura kompilatora – szczegóły



Analizator leksykalny

Analiza leksykalna. Analizator leksykalny czyta znaki z programu źródłowego i grupuje je w sekwencje, które reprezentują leksemy. Dla każdego leksemu analizator leksykalny produkuje token o postaci:
$< token - name, attribute - value >$. Na przykład załóżmy, że program źródłowy zawiera instrukcję przypisania:
$position := initial + rate * 60$.

Analizator leksykalny

Analizator leksykalny produkuje następujący wynik:

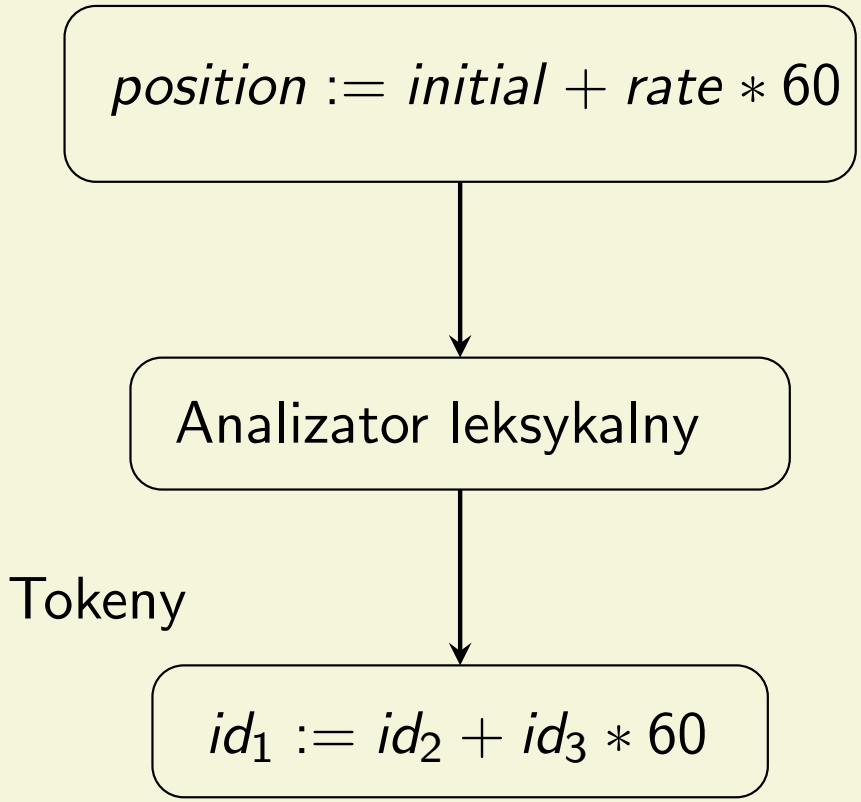
1. *position* jest leksemem, dla którego jest tworzony token: $< id, 1 >$, gdzie *id* jest to symbol abstrakcyjny oznaczający identyfikator; 1 jest adresem, pod którym tablica symboli przechowuje leksem *position* oraz dodatkowe atrybuty, na przykład typ danych.
2. Symbol przypisania := jest leksemem, dla którego jest produkowany token $<:=>$. Ponieważ ten token nie wymaga atrybutu, drugi składnik jest pominięty.
3. *initial* jest leksemem, dla którego jest tworzony token $< id, 2 >$; 2 jest adresem, pod którym tablica symboli przechowuje leksem *initial*.

Analizator leksykalny

Analizator leksykalny produkuje następujący wynik:

4. $+$ jest leksemem, dla którego jest produkowany token $< + >$.
5. *rate* jest leksemem, dla którego jest tworzony token $< id, 3 >$; 3 jest adresem, pod którym tablica symboli przechowuje leksem *rate*.
6. $*$ jest leksemem odwzorowywanym na token $< * >$.
7. 60 jest leksemem odwzorowywanym na token $< 60 >$.

Schemat działania analizatora leksykalnego

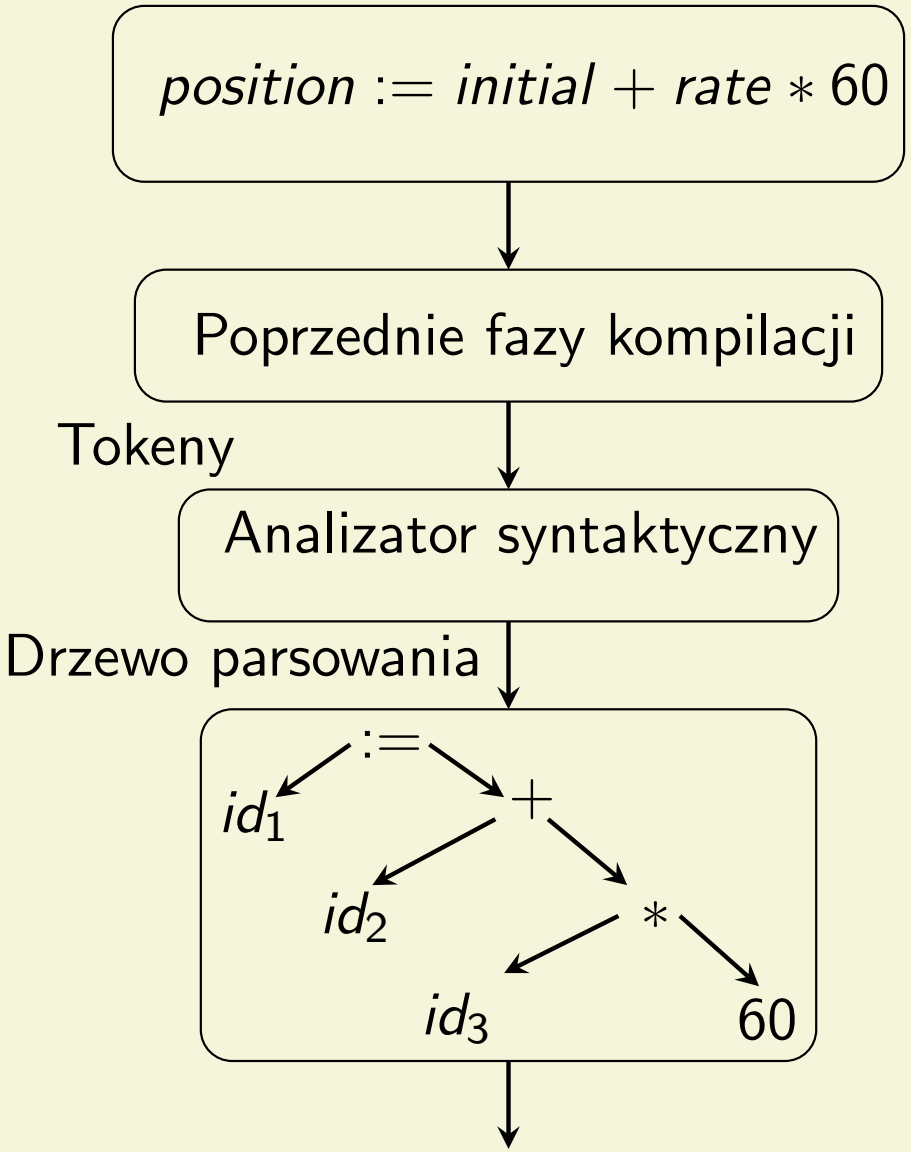


Rysunek: Schemat działania analizatora leksykalnego

Analizator syntaktyczny

Analiza syntaktyczna. Parser wykorzystuje pierwsze składniki tokenów, produkowane przez analizator leksykalny, aby utworzyć reprezentację pośrednią, która przedstawia strukturę gramatyczną strumienia tokenów. Typową reprezentacją składni jest drzewo syntaktyczne, w którym każdy węzeł wewnętrzny reprezentuje operację, natomiast „dzieci" tego węzła stanowią argumenty operacji. Na podstawie utworzonego drzewa parser decyduje, czy składnia programu jest poprawna. Na wyjściu parsera uzyskano wynik w postaci drzewa parsowania pokazany na rysunku na następnym slajdzie.

Schemat działania analizatora składniowego



Analizator semantyczny

Analiza semantyczna. Analizator semantyczny korzysta z drzewa parsowania, wykorzystuje informacje przechowywane w tablicy symboli i sprawdza program źródłowy pod względem spójności semantycznej, zdefiniowanej przez język programowania. Ponadto gromadzi informacje o typach zmiennych i zapisuje je w drzewie parsowania lub w tablicy symboli do wykorzystania podczas kolejnych etapów generacji kodu pośredniego.

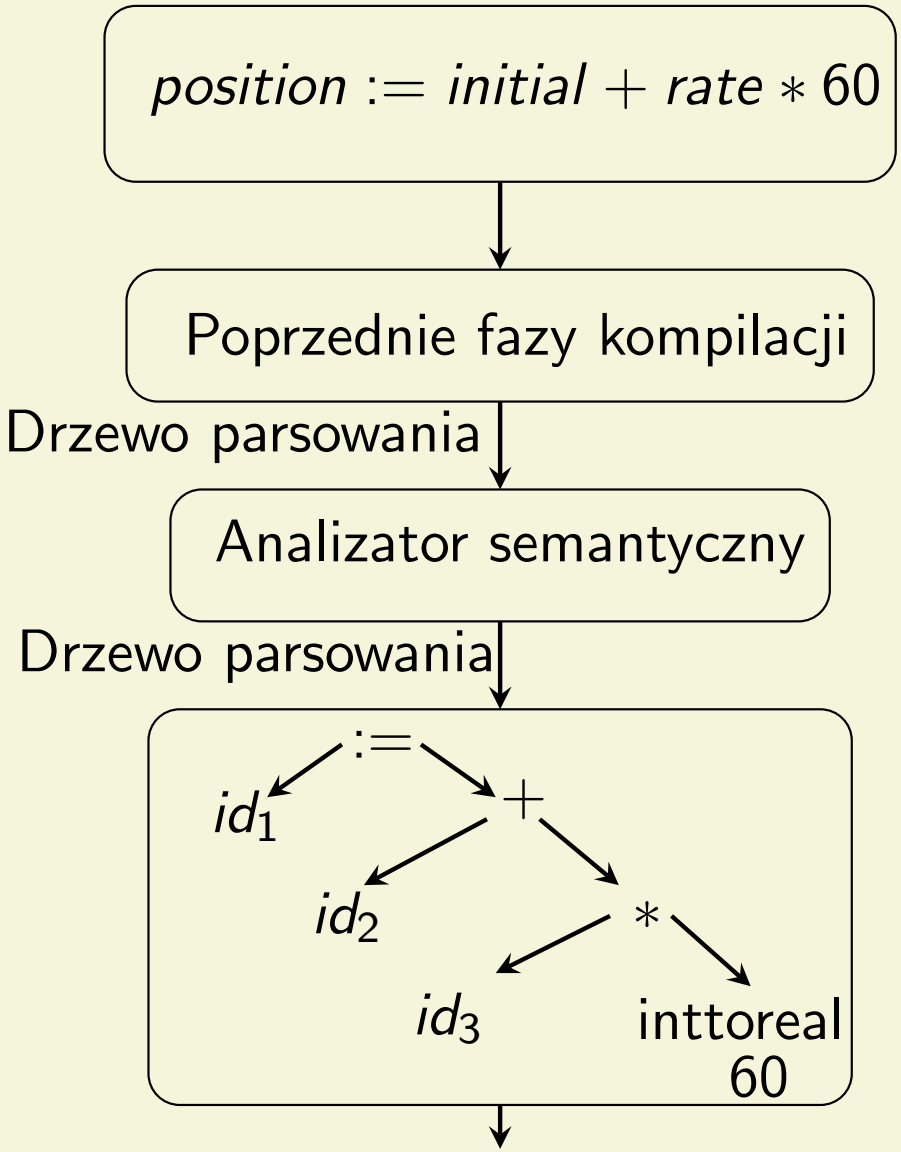
Analizator semantyczny

Ważną częścią analizy semantycznej jest kontrola typów – kompilator sprawdza, czy każdy operator ma dopasowane argumenty. Na przykład wiele języków programowania wymaga, żeby indeksy tablicy były liczbami całkowitymi; kompilator musi zgłosić błąd, jeśli do reprezentacji indeksu tablicy jest używana liczba zmiennoprzecinkowa. Specyfikacja języka może pozwalać na konwersję typów, znaną jako wymuszenie (*coercion*). Na przykład operator arytmetyczny może być zastosowany do pary liczb całkowitych lub pary liczb zmiennoprzecinkowych. Jeśli operandy nie należą do tego samego typu danych, to jeden z nich może być konwertowany do typu drugiego operandu.

Analizator semantyczny

Na rysunku pokazującym schemat działania analizatora semantycznego, operator *inttoreal* konwertuje liczbę całkowitą na liczbę zmiennoprzecinkową.

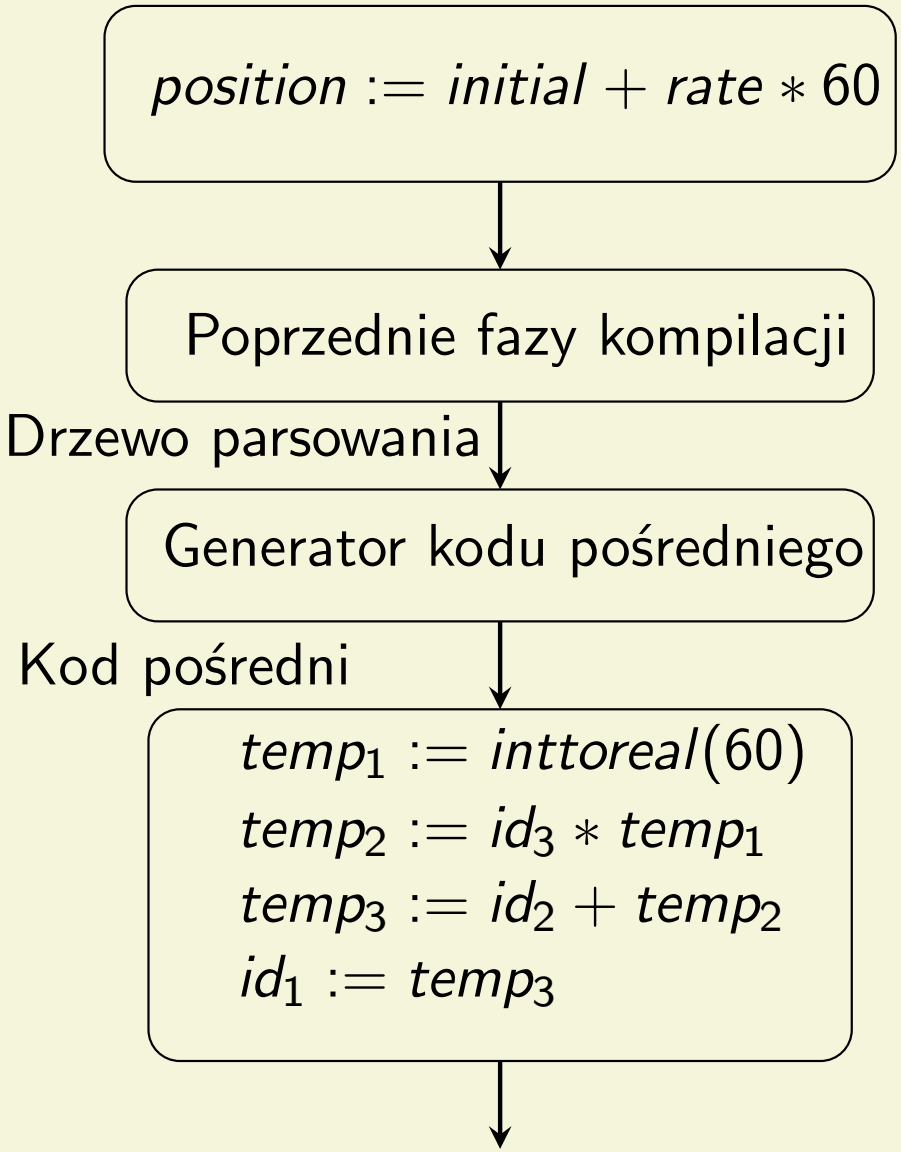
Schemat działania analizatora semantycznego



Generacja kodu pośredniego

W procesie tłumaczenia programu źródłowego na kod docelowy kompilator może utworzyć jedną reprezentację lub kilka reprezentacji pośrednich, które mogą mieć różne formy. Na przykład drzewa składniowe są popularną formą reprezentacji pośredniej. Schemat działania generatora kodu pośredniego pokazany jest na kolejnym slajdzie.

Generacja kodu pośredniego



Generacja kodu pośredniego

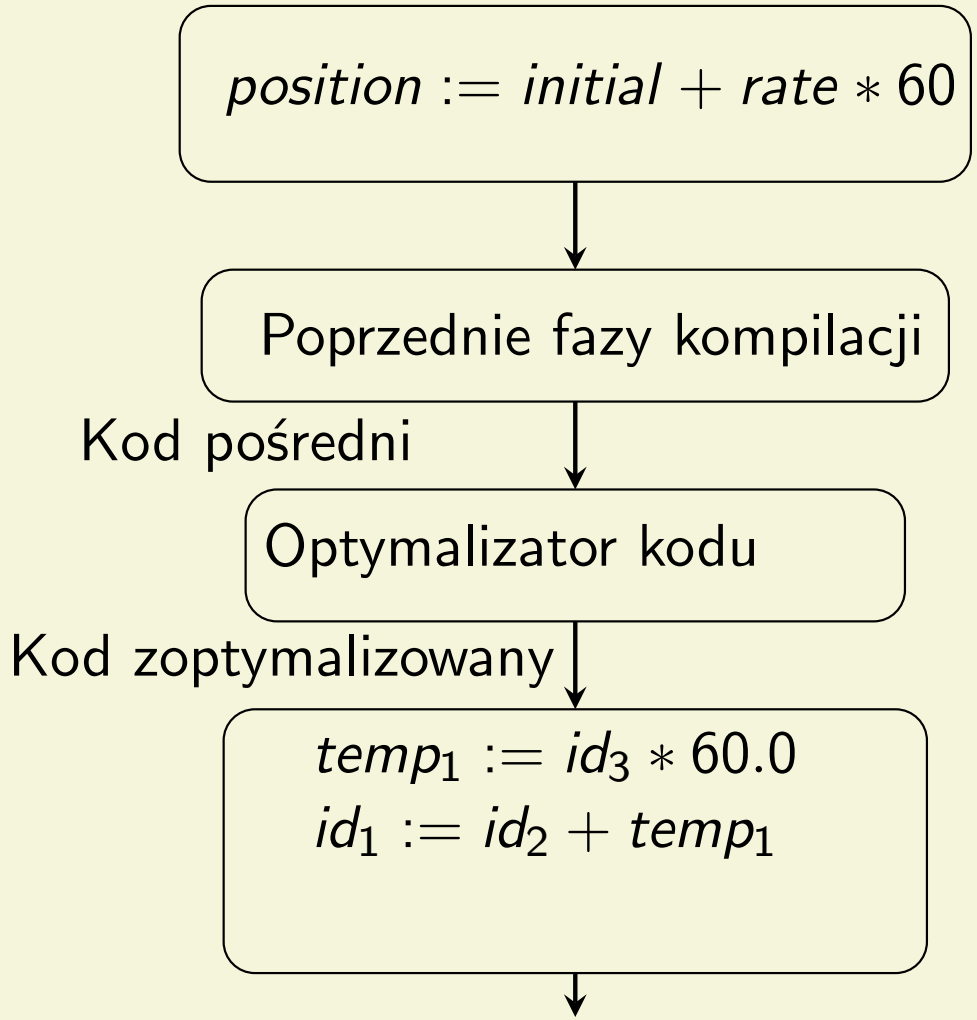
Drugą popularną formą jest kod trójadresowy:

```
t1 = inttoreal(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

Optymalizacja kodu

Optymalizacja kodu. Polega ona na redukcji liczby instrukcji i/lub zmniejszeniu zapotrzebowania na pamięć (zmniejszenie liczby zmiennych tymczasowych). Optymalizacja kodu może być fazą opcjonalną; schemat działania tej fazy kompilatora pokazany jest na kolejnym slajdzie.

Schemat z optymalizacją kodu

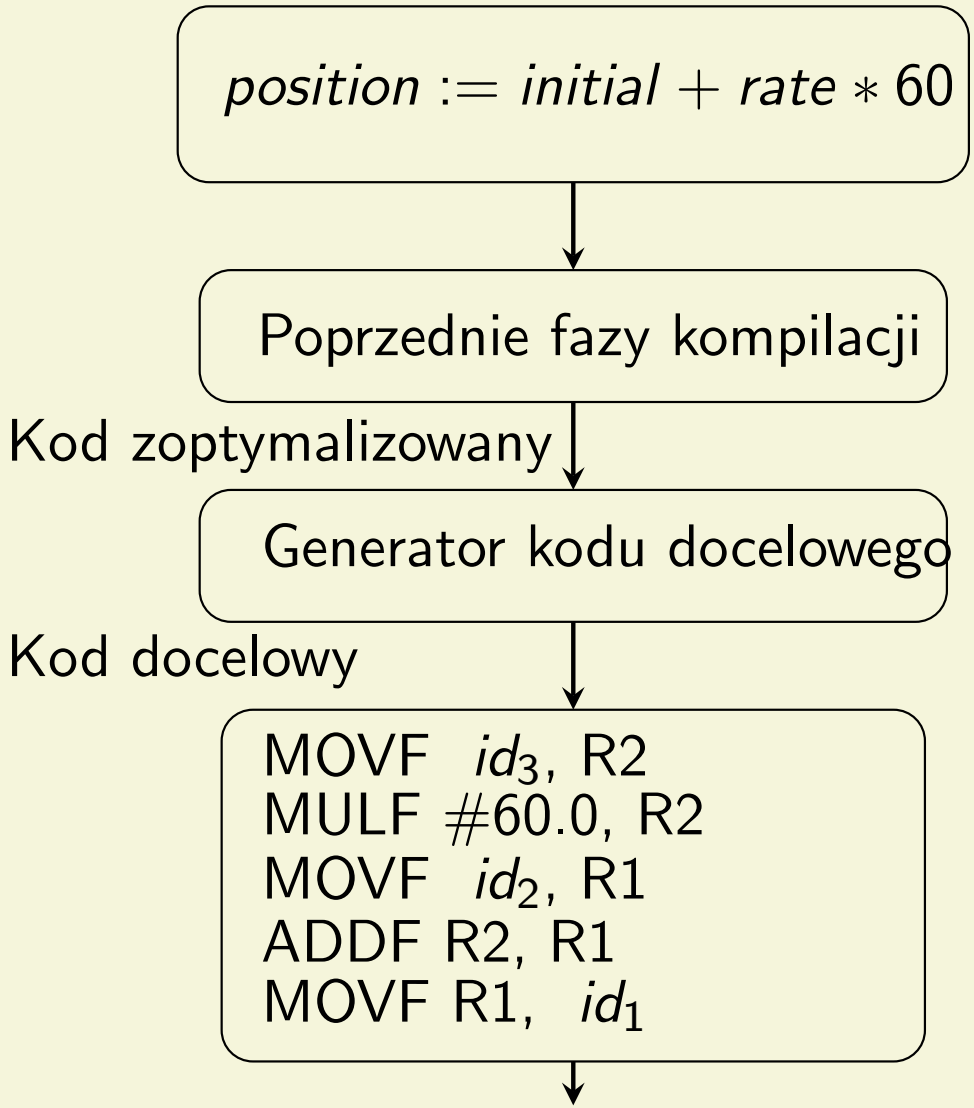


Rysunek: Schemat z optymalizacją kodu

Generacja kodu

Generacja kodu. Generator kodu tłumaczy reprezentację pośrednią programu na program w języku docelowym. Jeśli programem docelowym ma być kod maszynowy, to generator kodu musi przydzielić pamięć (rejestry, pamięć operacyjną) dla każdej zmiennej zdeklarowanej w programie źródłowym. Następnie każda instrukcja postaci pośredniej jest tłumaczona na sekwencję instrukcji maszynowych. Aspektem kluczowym generowania kodu maszynowego jest optymalny przydział rejestrów do przechowywania zmiennych. Na następnym slajdzie pokazany jest schemat budowy ostatniej części kompilatora.

Schemat z optymalizacją kodu



Tablica symboli

Tablica symboli. Jest ona strukturą danych zawierającą rekord dla każdej nazwy zmiennej, wraz z polami do przechowywania atrybutów tej zmiennej. Tablica symboli powinna być zaprojektowana tak, aby kompilator mógł szybko znaleźć rekordy dla każdej nazwy oraz szybko zapisać lub pobrać dane z tego rekordu.

Generacja kodu

Grupowanie faz kompilacji. Przedstawione fazy kompilatora pokazują jego logiczną organizację. W implementacji kompilatora fazy te mogą być grupowane w jedną większą fazę. Na przykład analiza leksykalna, analiza syntaktyczna i analiza semantyczna mogą być połączone w jedną fazę; w tej fazie czynności wszystkich analiz są wykonywane jednocześnie pod kontrolą analizatora syntaktycznego (kompilacja sterowana składnią).