Kompilacja i kompilatory

Wykład #10

wersja:
18.01.2023

Kompilacja

- W rozumieniu ogólnym:
  Kompilacja – proces tłumaczenia kodu programu z pewnego języka programowania na inny

- W powszechnym rozumieniu:
  Kompilacja – proces tłumaczenia kodu programu z pewnego języka programowania wysokiego poziomu do kodu maszynowego

Kompilacja

- ogólny schemat kompilacji

Kompilacja

- **Kod źródłowy**:
  - kod programu zapisany w danym języku programowania, dostarczony przez programistę
  - kod programu odczytany z bibliotek systemu kompilacji
  - kod wygenerowany automatycznie

- **Kod wynikowy**:
  - kod maszynowy procesora docelowego
  - kod assemblerowy
  - kod w innym języku programowania (najczęściej niższego poziomu np. kompilacja z C++ do C)

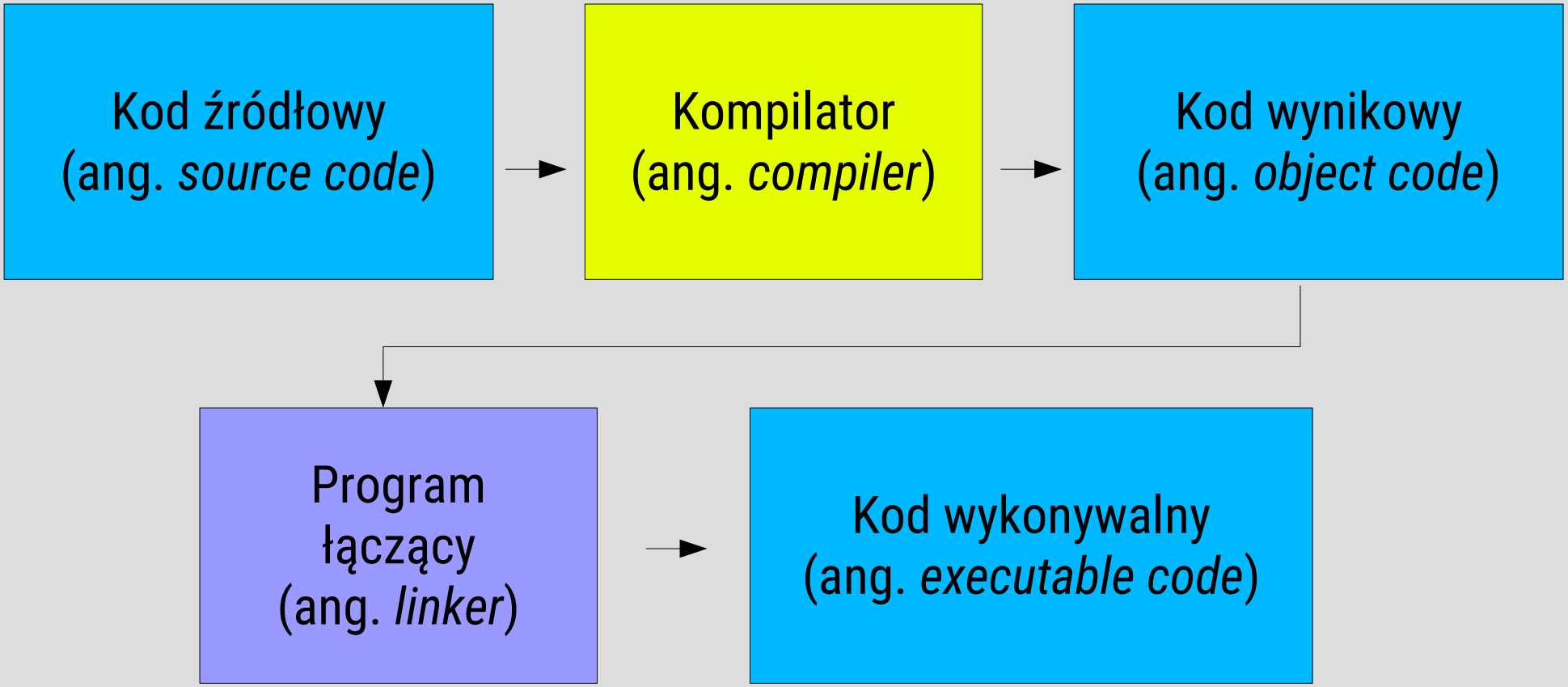
Kompilacja

- **dekompilacja** – proces odwrotny do kompilacji tzn. tłumaczenie kodu w języku niższego poziomu na kod poziomu wyższego.

– rzadko kiedy możliwa jest w pełnym zakresie, bowiem:
  - brak informacji o nazwach zmiennych
  - brak pełnej odpowiedniości między strukturą kodu wynikowego a kodem źródłowym
  - oparta na zawodnych heurystykach
  - może naruszać prawa autorskie (inżynieria wsteczna)

Kompilacja

- rzeczywisty schemat kompilacji



Kompilacja

Linker – zadania:

- dołączenie do kodu wynikowego powstałego w czasie kompilacji kodu źródłowego z kodem bibliotek standardowych (najczęściej dostarczanych razem z kompilatorem)
- połączenie kodu wynikowego powstałego z rozdzielnej kompilacji różnych fragmentów kodu źródłowego
- dołączenie do kodu wynikowego dodatkowych informacji (np. słowników symboli)

Kompilacja

Plik wykonywalny:

- najczęściej **absolutnie** nieprzenośny, nierozerwalnie związany z platformą (sprzętem, systemem operacyjnym) dla którego został wygenerowany
- każdy odrębny system operacyjny ma własny format plików wykonywalnych

Kompilacja

System DOS:

- format COM (pliki *.COM) – zawierał binarny obraz programu, gotowy do natychmiastowego załadowania do pamięci i wykonania; maks. łączny rozmiar kodu i danych – 64KB
- spadek po systemach ośmiobitowych (CP/M)

Kompilacja

System DOS:

- format EXE (pliki *.EXE), zwany również formatem MZ (dlaczego?); zawiera kod przemieszczalny; wymaga specjalnej procedury ładowania do pamięci, ale pozwala na nieograniczony (?) rozmiar kodu i danych

Kompilacja

System Windows:

- starsze wersje – format **NE** (New Executable) – używany w starych aplikacjach 16 bitowych
- aktualnie – format **PE** (Portable Executable)

Pliki **NE** i **PE** oprócz kodu mogą zawierać:

- pliki ikon
- pliki łańcuchów tekstowych
- inne pliki zasobów

Kompilacja

Linux:

- format **ELF**
  (ang. *Executable and Linkable Format*)
- wspólny format plików wykonywalnych i bibliotecznych
- inne (starsze formaty): a.out, COFF

Kompilacja

Narzędzia operujące na kodzie wykonywalnym:

- programy uruchomieniowe (ang. *debugger*)
- programy profilujące (ang. *profiler*)

Kompilacja

- **Debugger**:
  - w dosłownym tłumaczeniu: *odrobaczacz*
  - program pozwalający programiście śledzić wykonanie programu na poziomie kodu źródłowego
  - środki udostępniane przez debugger:
    - pułapka
    - inspektor
  - powinien być ostatnią deską ratunku, w praktyce jednak bywa inaczej - dlaczego?

Kompilacja

Profiler:

- narzędzie pozwalające zdejmować charakterystykę czasową pracującego programu
- środek wykrywania partii kodu wymagających optymalizacji
- nieodzowny przy pisaniu i uruchamianiu kodu, dla którego czas wykonania ma znaczenie krytyczne

Kompilacja

- **Sytuacja standardowa:**
  - kompilator generuje kod wynikowy/wykonywalny dla tej samej platformy, na której sam pracuje
  - np. kompilator języka C pod systemem Windows produkuje plik formatu PE przeznaczony do wykonania pod systemem Windows
  - **kompilacja skrośna (ang. *cross compilation*)** kompilator generuje kod dla innej platformy niż jego własna

Kompilacja

- Czy dla działania kodu źródłowego konieczna jest kompilacja?
- Dwa angielskie znaczenia polskiego słowa „tłumacz”:
  - *translator* – tłumacz tłumaczący np. książkę
  - *interpreter* – tłumacz tłumaczący symultanicznie
- W informatyce:
  - **translator** – program tłumaczący kod źródłowy (jednokrotnie) w celu wykonania kodu wynikowego (wielokrotnie) (inaczej – kompilator)
  - **interpreter** – program wykonujący kod źródłowy „w locie” bez jego tłumaczenia na kod wynikowy

Kompilacja

- **Języki interpretowane**:
  - PERL
  - BASIC
  - Python
  - Ruby
  - REXX
  - Forth
  - języki shellowe (np. bash)
  - i wiele, wiele innych...
- zwyczajowe określenie programu w języku interpretowanym – **skrypt**
- zalety i wady języków interpretowanych

Kompilacja

- Czy program zawsze pisze się w całości w jednym języku programowania?
- **Program hybrydowy –** program uzyskany ze złożenia części napisanych w różnych językach programowania
- Prawie każdy system operacyjny jest programem hybrydowym
- Możliwe hybrydy:
  - Asembler – C
  - C – C++
  - Java – C
  - itd.

Kompilacja

- Tworzenie programu hybrydowego:
  - pisanie kodu źródłowego
  - kompilacja (z użyciem różnych kompilatorów)
  - kod wynikowy w jednym formacie
  - połączenie w jeden komponent – linker
- Możliwe wtedy, gdy:
  - wszystkie kompilatory tworzą identyczny format kodu wynikowego
  - istnieje standard wymiany danych pomiędzy kodem napisanym w różnych językach
  - ABI – ang. *Application Binary Interface*

Kompilacja

- Czemu tworzy się programy hybrydowe?
  - podniesienie wydajności
  - zmniejszenie rozmiaru kodu
  - wykorzystanie najlepszych cech różnych języków programowania

Kompilacja

- **Fazy kompilacji:**
  - przedprzetwarzanie (ang. *preprocessing*)
  - analiza leksykalna
  - analiza składniowa (syntaktyczna)
  - analiza znaczeniowa (semantyczna)
  - generowanie postaci pośredniej
  - generowanie kodu wynikowego
  - optymalizacja

Kompilacja

- Ze względu na sposób działania kompilatory dzielimy na:
  - **jednoprzebiegowe** (ang. *single-pass*)
    - wszystkie czynności wykonywane są po jednokrotnym przeczytaniu kodu źródłowego
    - szybciej, ale ze zużyciem dużego obszaru pamięci operacyjnej
  - **wielprzebiegowe** (ang. *multi-pass*)
    - każda z czynności może wymagać ponownego przeczytania kodu źródłowego lub kodu pośredniego
    - wolniej, ale nie jest potrzebna duża pamięć

- *Istniał kompilator języka Algol, któremu do pracy wystarczyło 16KB pamięci – wymagał 14 przebiegów (sic!)*

Kompilacja

- **Przedprzetwarzanie**
  - dostępne tylko w niektórych językach (np. C, C++)
  - odpowiednik makr z asemblerów
  - pierwszy język z preprocesorem – PL/I
  - funkcje preprocesora:
    - włączanie obcych plików źródłowych
      #include
    - zastępowanie symboli
      #define
    - rozwijanie makr
      #define
    - generowanie źródła

Kompilacja

- Wpływ na zachowanie kompilatora:
  - poprzez opcje (parametry) przekazywane z zewnątrz kompilatora
  - poprzez dyrektywy umieszczane w kodzie źródłowym – **pragma**
  - **pragma** nie wpływa na znaczenie kodu źródłowego
  - **pragma** nie jest częścią definicji języka programowania
  - **pragma** instruuje kompilator, jaki wariant pracy wybrać w danym miejscu kodu

    np. #pragma once

Kompilacja

- Analiza leksykalna
  - ma na celu wyodrębnienie z kodu źródłowego tzw. atomów leksykalnych tzn. najmniejszych niepodzielnych jednostek programu
  - atom leksykalny → **leksem** lub **token**
  - działanie analizatora leksykalnego:
    - ciąg znaków (napis) → ciąg leksemów
  - **leksemy:**
    - słowa kluczowe
    - literały
    - symbole (nazwy bytów)
    - komentarze
    - operatory
    - etc

Kompilacja

- Na etapie analizy leksykalnej wykrywa się:
  - niedozwolone znaki (spoza alfabetu języka)
  - niedozwolone postaci identyfikatorów
  - niedozwolone literały (spoza zakresu)
  - niedomknięte komentarze
  - niedomknięte napisy

Kompilacja

- Przykładowy efekt analizy leksykalnej:
- wejście:

```
int main(void) { puts(”Hello”); return 0; }
```

- wyjście:

```
<keyword int> <ident main><separator (>
<keyword void> <separator )> <separator {>
<ident puts> <literal Hello> <separator ;>
<keyword return> <literal 0> <separator ;>
<separator }> <end>
```

Kompilacja

- **Analiza składniowa**
  (syntaktyczna – od ang. *syntax* - składnia)
- Ma na celu:
  - sprawdzenie, czy układ leksemów jest zgodny z gramatyką języka
  - sprawdzenie sparowania nawiasów (faktycznych i składniowych np. w Pascalu **begin** i **end**)
  - identyfikację przeznaczenia identyfikatorów (zmienne, funkcje, etykiety, etc)
  - sprawdzenie poprawności wyrażeń
  - sprawdzenie poprawności użycia separatorów
  - wstępne wypełnienie tablicy symboli kompilatora

Kompilacja

- Tablica symboli (ang. *symbol table*) – wewnętrzna „baza danych” kompilatora przechowująca informację o każdym zdefiniowanym w programie symbolu:
  - nazwa
  - rodzaj
  - typ
  - zakres
  - zasięg
  - użycie
- Mimo swojej nazwy realizowana najczęściej jako drzewo

Kompilacja

- Analiza składniowa:
  - wykonywana na podstawie gramatyki języka
  - gramatyka języka programowania – zbiór reguł i prawideł, pozwalający jednoznacznie ustalić, czy pewien program jest poprawny, czy nie
  - gramatyki formalne:
    - notacja BNF (Backus-Naur Form)
      - John Backus – FORTRAN
      - Peter Naur - Algol
    - diagramy składniowe Wirtha

Kompilacja

- Dygresja – co to jest „metajęzyk”?
- Metajęzyk – język używany do opisu innego języka
- Gramatyka formalna jest przykładem metajęzyka

Kompilacja

- Na etapie analizy składniowej wykrywa się:
  - błędy składniowe (sic!)
  - zgubione bądź nadużyte znaki przestankowe i separatory
  - źle skonstruowane wyrażenia
  - niepoprawnie zapisane słowa kluczowe
  - użycie słów kluczowych w niewłaściwym kontekście
- Analizator składniowy realizuje się jako tzw. automat skończony (co to jest?)

Kompilacja

- **Analizator semantyczny**
- sprawdza:
  - zgodność typów
  - dopasowanie danych i operatorów
  - konteksty wywołań funkcji
  - sensowność deklaracji
  - zakresy i zasięgi deklaracji
- Należy pamiętać, że nawet najdoskonalszy analizator semantyczny nie jest w stanie czytać w myślach programisty!

Kompilacja

- Dygresja: co to jest **typizacja**?
- Cecha języka programowania określająca surowość z jaką język podchodzi do kwestii zgodności typów danych
- Typizacja może być:
  - silna (Java, C#)
  - słaba (C, Perl)
- Oraz może być:
  - statyczna
  - dynamiczna

Kompilacja

- Analizator semantyczny wykrywa np:
  - wywołanie funkcji z niewłaściwą liczbą parametrów
  - wywołanie funkcji niezgodne z jej typem
  - użycie instrukcji **return** w niewłaściwym kontekście
  - użycie niewłaściwych operatorów lub danych

Kompilacja

- **Generowanie postaci pośredniej**
  - ma na celu przekształcenie postaci programu
  - nie musi być zależne od platformy docelowej (np. kompilator GCC)
  - musi być jednoznaczne

Kompilacja

- **Generowanie postaci pośredniej**
  - najczęściej stosowane formy postaci pośredniej
    - trójki (triady)
    - czwórki (tetrady)
    - Odwrotna Notacja Polska (ang. RPN – Reverse Polish Notation)

Kompilacja

- **Trójki**
  - program przekształca się do ciągu elementarnych operacji postaci:

    <operator, źródło, cel>

    gdzie:
    operator – operacja atomowa (np. arytmetyczna)
    źródło – argument dla operacji
    cel – przeznaczenie wyniku operacji

Kompilacja

- **Trójki - przykład**
  - program źródłowy:

    a = 2 * b + c;

  - zbiór triad:

```
<=,2,temp>
<*,b,temp>
<+,c,temp>
<=,temp,a>
```

Kompilacja

- **Czwórki**
  - program przekształca się do ciągu elementarnych operacji postaci:

    <operator, źródło1, źródło2, cel>

    gdzie:
    operator – operacja atomowa (np. arytmetyczna)
    źródło1 – pierwszy argument dla operacji
    źródło2 – drugi argument dla operacji
    cel – przeznaczenie wyniku operacji

Kompilacja

- **Czwórki - przykład**
  - program źródłowy:

    a = 2 * b + c;

  - zbiór tetrad:

    <*,2,b,temp>
    <+,temp,c,a>

Kompilacja

- **Notacja Polska**
  - Jan Łukasiewicz (1878-1956)
  - Notacja Polska – 1920
  - formalizm pozwalający zapisywać dowolnie złożone wyrażenie bez użycia nawiasów
  - zdefiniowana dla potrzeb logiki formalnej, następnie przeniesiona na algebrę
  - reguła podstawowa:
    najpierw operator,
    potem jego argumenty
  - zwana również notacją prefiksową (w odróżnieniu od klasycznej infiksowej)

Kompilacja

- **Notacja Polska – przykłady**
  - wyrażenie w postaci infiksowej:

    ```
    b * b – 4 * a * c
    ```

  - wyrażenie w postaci prefiksowej:

    ```
    – * b b * * 4 a c
    ```

Kompilacja

- **Odwrotna Notacja Polska:**
  - oparta na odwróceniu pomysłu Łukasiewicza: najpierw argumenty, potem operator – notacja postfiksowa
  - zwana również jako „Azciweisakul notation”
  - zaproponowana przez Charlesa Hamblina (1957)
  - opiera się na wykorzystaniu mechanizmu stosu (ang. *stack*)

Kompilacja

- **Odwrotna Notacja Polska – przykład**:
  - wyrażenie infiksowe:

    `b * b – 4 * a * c`

  - wyrażenie postfiksowe

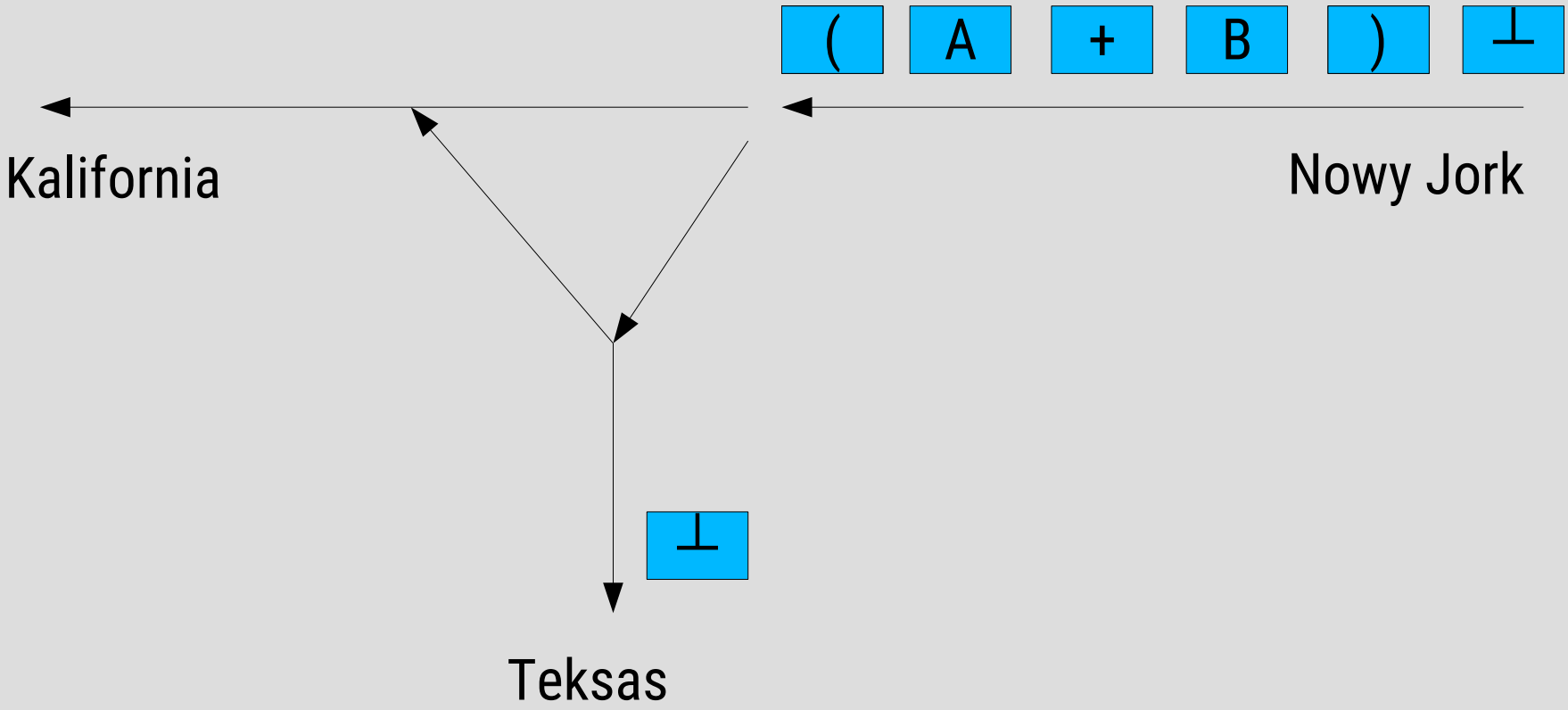
    `b b * 4 a c * * –`

Kompilacja

- **Tłumaczenie wyrażeń z postaci infiksowej na postfiksową**
  - wiele algorytmów różniących się skomplikowaniem i wydajnością
  - Dijkstra - algorytm „bocznicy kolejowej”

Kompilacja

- **Stacja rozrządowa gdzieś w Ameryce...**



Kompilacja

- **Algorytm zwrotniczego:**

Teksas \ Nowy Jork	⊥	+	-	*	/	(	)
⊥	4	1	1	1	1	1	5
+	2	2	2	1	1	1	2
-	2	2	2	1	1	1	2
*	2	2	2	2	2	1	2
/	2	2	2	2	2	1	2
(	5	1	1	1	1	1	3

1. Nowy Jork → Teksas
2. Teksas → Kalifornia
3. Porwanie!
4. Koniec
5. Błąd!

- pierwszy ⊥ zawsze jedzie do Teksasu
- dane zawsze jadą do Teksasu

Kompilacja

- **Odwrotna Notacja Polska – obliczanie wartości wyrażeń**:
  - przetwarzaj wyrażenie od lewej do prawej
  - jeśli na wejściu masz liczbę, odłóż ją na stos
  - jeśli na wejściu masz operator, zdejmij ze stosu argumenty dla niego, wykonaj operację i odłóż wynik na stos
  - po przetworzeniu całego wyrażenia na stosie zostanie ostateczny wynik

Kompilacja

- **Odwrotna Notacja Polska**:
  - używana w kalkulatorach naukowych (np. Hewlett Packard, National Semiconductor)
  - stosowana w niektórych językach programowania (FORTH, Postscript)
  - używana w uniksowym kalkulatorze dc

Kompilacja

- **Odwrotna Notacja Polska:**
  - zalety:
    - każde wyrażenie można zapisać bez użycia nawiasów – redukuje się liczbę używanych symboli
    - jeżeli pewna architektura procesora pozwala na wykonywania operacji na stosie, to jest to naturalna forma realizacji obliczeń (np. koprocesor w procesorach x86)
    - eliminacja niejednoznaczności wynikających z różnego priorytetu i siły wiązania różnych operatorów

Kompilacja

- **Odwrotna Notacja Polska – reprezentacji instrukcji warunkowych**:
  - kod:
    ```
    if(a > 2) c=0; else c = 1;
    ```
  - RPN:
    ```
    a 2 > if c 0 = then c 1 = else
    ```

Kompilacja

- **Generowanie kodu wynikowego**:
  - jedyny komponent kompilatora zależny od platformy docelowej
  - postać wyjściowa:
    - kod maszynowy (półskompilowany)
    - asembler
  - generator przekształca postać pośrednią na kod maszynowy uwzględniając:
    - architekturę procesora
    - liczbę dostępnych rejestrów
    - dostępne tryby adresowania
    - listę rozkazów

Kompilacja

- Generowanie kodu wynikowego – przykład:

```
int main(void) {
  int a = 4, b = 2, c = 5, x;

  x = b * b - 4 * a * c;
  return 0;
}
```

Kompilacja

- **Generowanie kodu wynikowego – wyjście z kompilatora GCC z opcją -S:**

```
movl    $4, -4(%ebp)
movl    $2, -8(%ebp)
movl    $5, -12(%ebp)
movl    -8(%ebp), %eax
movl    %eax, %edx
imull   -8(%ebp), %edx
movl    -4(%ebp), %eax
imull   -12(%ebp), %eax
sall    $2, %eax
subl    %eax, %edx
movl    %edx, %eax
movl    %eax, -16(%ebp)
movl    $0, %eax
ret
```

Kompilacja

- **Optymalizacja**: automatyczne „poprawianie” gotowego już kodu wynikowego w celu podniesienia jego wydajności (ang. *speed optimizing*) lub zmniejszenia rozmiaru (ang. *size optimizing*)
- Kod może być albo szybki, albo mały – *tertium non datur*
- Poziomy optymalizacji (np. w kompilatorze gcc możliwe są 3 poziomy optymalizacji)

Kompilacja

- Jak może działać „speed optimizing”?
- kod na wejściu:

```
for(i = 0; i < 1000000; i++)
      t[i] = 2 * k + t[i];
```

- kompilator wykrywa podwyrażenie, które nie zmienia się wewnątrz pętli i wynosi je przed pętlę:

```
temp = 2 * k;
for(i = 0; i < 1000000; i++)
   t[i] = temp + t[i];
```

Kompilacja

- Kompilator może wykrywać powtarzające się podwyrażenia:
- kod na wejściu:

```
t[i] = 2 * k + t[i] + u[2 * k]
```

- kompilator wykrywa wspólne podwyrażenie i oblicza jego wartość tylko raz:

```
temp = 2 * k;
t[i] = temp + t[i] + u[temp];
```

Kompilacja

- Kompilator może wykrywać intensywnie wykorzystywane zmienne i przechowywać je nie w pamięci, a w rejestrach procesora
- Kompilator może wykrywać intensywnie wykorzystywane literały i przechowywać je w rejestrach procesora
- Kompilator może wykrywać przypadki mnożenia/dzielenia przez potęgi dwójki i zastępować je przesunięciami
- etc, etc, etc ...

Kompilacja

- im agresywniejsza optymalizacja, tym dłuższa kompilacja
- konieczne znalezienie złotego środka
- typowe postępowanie:
  - tzw. wersje **deweloperskie** kompiluje się bez optymalizacji bądź ze słabą optymalizacją
  - tzw. wersje **produkcyjne** kompiluje się z silną optymalizacja

Kompilacja

- niekiedy producent kompilatora dostarcza dwa oddzielne produkty:
  - kompilator diagnostyczny (ang. *checkout compiler*)
    - kompiluje szybko
    - generuje słaby kod
    - dysponuje bardzo rozbudowaną diagnostyką
    - analizuje kod pod kątem wyszukiwania typowych i uciążliwych błędów
  - kompilator optymalizujący (ang. optimizing compiler)
    - generuje kod o wysokiej jakości
    - powolny
    - uboga diagnostyka