Компајлери-вовед

Преведување и интерпретација

Преведување, интерпретација, компајлер-интуитивно

- Преведување – процес на пишување, искажување на реченица од еден јазик во друг
  - Од англиски на македонски
  - Пр. How to construct a compiler? – Како се конструира компајлер? (Алтернативно: Како да конструираме компајлер? Или Како се конструира компајлер?)
  - Од Јава во Ц++, од Паскал во Асембли јазик
- Преведувач – систем кој врши преведување од еден во друг јазик

Преведување, интерпретација, компајлер-интуитивно

- Интерпретација – да се разбере смислата на искажаната реченица
  - Пр. Take the chalk and start writing on the board
  - Не не интересира дали точниот превод е “Земете креда и почнете да пишувате на таблата” или “Земете ја кредата и започнете со пишување на таблата”, туку само значењето.
- Интерпретатор – систем кој ја разбира смислата, но не прави буквален превод
  - Пр. Човековиот мозок, кој разбирајчи ја смислата го прави дејството.

Преведување, интерпретација, компајлер-интуитивно

- Што треба да се знае за да може да се преведе од еден на друг јазик?
  - Да се владеат и двата јазици
    - За се знаат азбуките од двата јазици (ако треба да се напише преводот)
    - Да се знаат зборовите од двата јазици
    - Да се разбираат реченициите кои се преведуваат
    - Да се знаат граматичките правила од двата јазици
  - Да се знае како една реченица формулирана во едниот јазик ќе се искаже на другиот јазик

Преведување, интерпретација, компајлер-интуитивно

- Граматичките правила во говорните јазици не се строго дефинирани, ист збор во различен контекст може да се замени со различни зборови
- Граматичките правила во машинските јазици се строго дефинирани
- Компајлер – преведувач од програмски јазик во машински јазик, или асембли јазик (виш јазик во ниж јазик)

Програмски јазици

- Основните операции во машински јазик се многу попримитивни во споредба со комплексните функции кои се јавуваат во математиката, инженерството и други дисциплини
  - Едноставен приказ со математички симболи и формули наспроти сложени примитивни функции, Пример множење на матрици

$$A=B*C$$

Програмски јазици

- Програмските јазици се олеснување во споредба со машинските јазици, но и тие самите воведуваат голем дел нови проблеми
- Бидејќи машините сеуште можат да ги “разберат” само машинските јазици, програмата напишана во виш јазик мора да биде преведена на машински јазик
- Постапката се нарекува компајлирање

Програмски јазици

- Друг проблем е спецификација на самиот јазик
- Минимални спецификации
  - Множество симболи кои можат да се користат во една валидна програма
  - Множество од валидни програми
  - Значење на секоја валидна програма

Спецификација на јазикот

- Множество симболи кои можат да се користат во една валидна програма е лесно за дефинирање
- Множество од валидни програми,
  - Дефинирање на граматички правила
  - Дозволуваат точна конструкција на програмата
- Повеќе аспекти за дефинирање значењето на секоја валидна програма
  - Пресликување од секоја наредба во реченица во говорниот јазик, која ќе објасни што тоа значи
  - Конструкција на идеализирана машина (интерпретер за јазикот)
  - Игнорирање на продлабочување на значењето, при што значењето е она што ќе се добие како излез кога ќе се искористи компајлерот врз изворна програма.

Спецификација на јазикот

- Множество симболи кои можат да се користат во една валидна програма е лесно за дефинирање
- Множество од валидни програми,
  - Дефинирање на граматички правила
  - Дозволуваат точна конструкција на програмата
- Повеќе аспекти за дефинирање значењето на секоја валидна програма
  - Пресликување од секоја наредба во реченица во говорниот јазик, која ќе објасни што тоа значи
  - Конструкција на идеализирана машина (интерпретер за јазикот)
  - Игнорирање на продлабочување на значењето, при што значењето е она што ќе се добие како излез кога ќе се искористи компајлерот врз изворна програма.

Спецификација на компајлер

- Множество од подредени парови (x, y), каде x е програма (наредба) напишана во изворниот јазик, а y е програма (наредба) напишана во целниот јазик, во кој што x треба да се преведе.
- Ако x е стринг напишан над азбуката Σ, а y над азбуката Δ, тогаш преведувањето е едноставно едно пресликување од Σ^* во Δ^*

Преведување и интерпретација

Преведувач

Преведувач е програма или систем кој конвертира внесен текст во еден јазик во текст од друг јазик, со истото значење

Интерпретер

Интерпретер е преведувач кој конвертира внесен текст во неговата смисла, така како што е дефинирано во неговата семантика.

Преведување и интерпретација

- Класичен интерпретер е BASIC, GW-BASIC
- Пример, 2+3
  - Интерпретерот ја пресметува вредноста 5
  - Преведувачот го преведува во машински код, кој после ја пресметува вредноста.

Преведување и интерпретација

- Класите преведувачи и интерпретери не се јасно дистанцирани
- Дефиниција на Meijer која ги одделува преведувачите од интерпретерите
  - Ако внесениот текст во преведувачот е некоја програма, таа програма може да има свои сопствени влезни податоци. Таквата програма може да се преведе, без знаење на содржината на влезните податоци, но не може да се толкува (интерпретира).

Пример за интерпретер

- Се калкулираат едноставни математички изрази
  - + (собирање)
  - - (одземање)
  - * (множење)
  - / (делење)

2+3*4-5

5-2-1

2^{3^2}

Пример за интерпретер

- Листа на приоритети

Оператори	^, *, /, +, -, [,]
Приоритет	1 2 2 3 3 4 4
Асоцијативност	д л л л л

Преведување и интерпретација

- *p*-програма за преведување, напишана во јазикот P, а i-влез. Тогаш интерпретер е функција ν_P, и резултатот на преводот на p со влез i е

$$\nu_P(p,i)$$

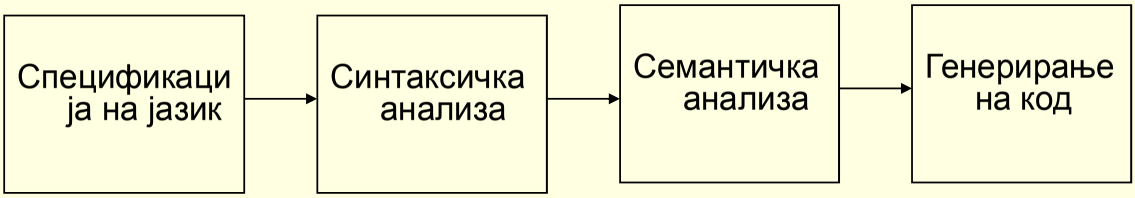
Ако c е преведувач, истиот резултат се добива со користење на преводот c(p) во програмскиот јазик M

$$\nu_M(c(p),i)$$

Конструкција на компајлер

- Специфицирање на програмски јазик
- Пишување на граматика за јазикот
- Компајлерот ја чита програмата, напишана во измисленият јазик и проверува дали синтаксички е точна
- Компајлерот верифицира дали смислата е коректна (ја проверува семантиката)
- Генерира код во јазик-цел (target language)

Конструкција на компајлер



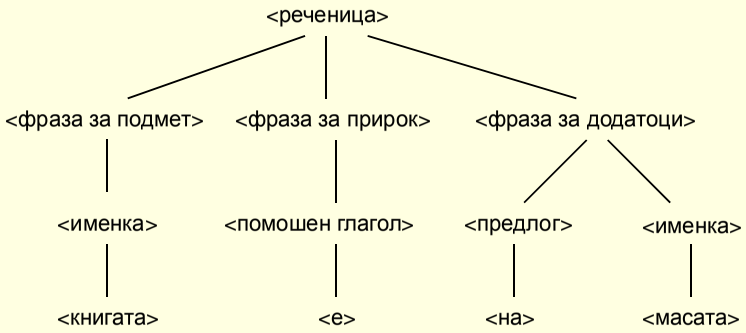
Синтакса и семантика

- Честопати при специфицирање и имплементирање на преведувачот, поудобно е да се смета дека тој е композиција од две попрости пресликувања, познати како *синтаксичко пресликување* и *семантичко пресликување*
  - Доменот на семантичкото пресликување е множеството слики од синтактичкото пресликување.
  - Најчесто, означеното синтасичко дрво се покажува многу корисна структура на место на влез за семантичкото пресликување .

Синтаксичко дрво

- Пример:

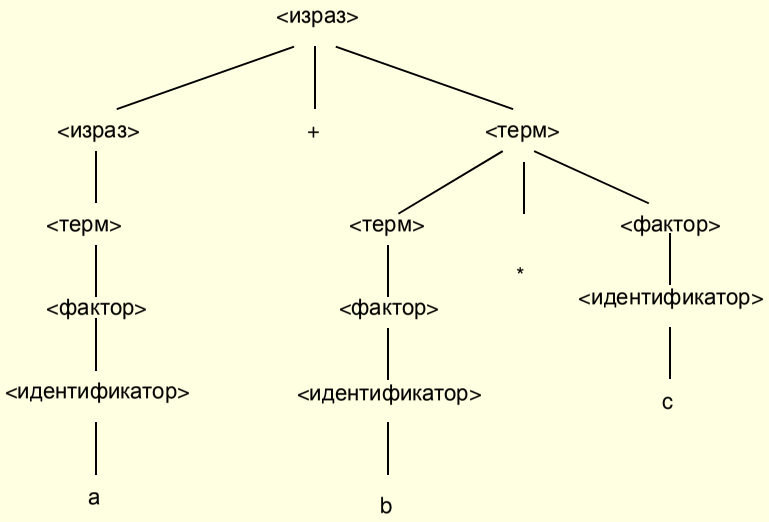
  “Книгата е на масата”



Синтаксичко дрво

- Пример:

  a+b*c



Синтакса

- Дрвото претставува сунтаксичка структура на реченицата
- Процесот на наоѓање на синтаксичката структура на реченицата се нарекува *синтаксичка анализа* или *парсирање*.
- *Синтакса на јазик* се однесува на релација која е соодветна на секоја реченица од Синтаксичката структура на јазикот,
  - Може да се дефинира валидна реченица преку низа од симболи која претставува покривач на синтаксичката структура на <реченица>

Семантичко пресликување

- Семантичкото пресликување е пресликување кое структурираниот влез кој се добива преку синтаксичката анализа го пресликува во излез кој вообичаено е програма напишана во машински јазик.
- Терминот *Семантика на јазик* се однесува на пресликување кое одговара на синтаксичката структура на секој влез во некој јазик (обично истиот) во која ние ја разгледуваме “смислата” на оригиналната реченица.
  - Не се целосно разрешени за говорни јазици.

Семантичко пресликување

- Два концепти за разрешување на проблемот на спецификација на синтакса и семантика кај програмските јазици
  - Контексно слободни граматики
    - Повеќето правила од јазикот можат да се претстават преку КСГ. Тие даваат доволно прецизни описи кои можат да се користат како дел од спецификацијата на компајлерот
  - Синтаксно-ориентирани шеми за превод
    - Се користат за специфицирање на пресликувањето од едниот во другиот јазик

Делови на компајлерот

- Изворниот код на било која програма напишана во било кој јазик не е ништо друго туку низа од стрингови. Компајлерот оваа низа ја претвара во низа од битови. Во тој процес можеме да одделиме некои делови
  - Лексичка анализа
  - Операции со симболни табели
  - Парсирање или синтаксичка анализа
  - Генерирање на код во мажински јазик или некој меѓу-јазик (асембли јазик)
  - Оптимизација на кодот

Спецификации за курсот

- 2 часа предавања
- 1 час аудиториумски вежби
- 2 часа лабараториски вежби

Теми за изработка

- Изработка на јазик
- Синтакса
  - Лексичка анализа
  - Граматика
  - Парсирање
  - Препроцесор
  - Поправка на грешки
- Семантика
  - Симболни табели
  - Проверка на типови
  - Различни семантички проверки
- Генерирање на код

Лексичка анализа

- Во програмата некои комбинации од стрингови се третираат како посебни делови
  - Се специфицира азбуката, т.е сите симболи од тастатурата кои можат да се внесат (букви, броеви, специјални симболи)
  - Се специфицираат клучни зборови како BEGIN, END, DO, FOR и т.н.
  - Стрингови кои ги репрезентираат нумеричките константи кои се третираат како посебни ставки.
  - Идентификатори кои се користат како имиња на променливите, функциите, процедурите, и слично

Лексичка анализа

- Работа на лексичкиот анализер е ваквите терминали да се групираат во посебни синтаксички делови, кои се нарекуваат токени, кои се специфицирани со тип на токенот и каков тип на податок може да се стави на негово место.
  - (тип на токен, податок)
  - Првото може да биде “константа” или “идентификатор”
  - Втората компонента е покажувачот кон податокот каде овој токен е запамтен.
- Лексичкиот анализер всушност претставува преведувач кој влезот го трансформира во низа од токени. Ова после станува влез за синтаксичкиот анализер.

Лексичка анализа

- Пр. cost=(prise+tax)*0.98
- Cost, prise и tax се токени од тип идентификатори, додека 0.98 е токен од тип константа. (,), +, * и = се токени сами за себе
- Може сите идентификатори и константи да ги обележиме со <id>
  - Првата компонента се користи од парсерот
  - Претпоставуваме дека има и втора компонента која е покажувачот кон податокот во табелата која ги содржи имињата на идентификаторите, заедно со други податоци за нив. Тој податок се користи во фазата на генерирање на код.
- Како излез се добива

  $<id>_1=(<id>_2+ <id>_3)* <id>_4$.

Симболни табели или книги за зачувување

- Табели во кои се зачувуваат податоците за секој идентификатор
  - Пр.

1	cost	реална променлива
2	price	реална променлива
3	tax	реална променлива
4	0.98	реална променлива

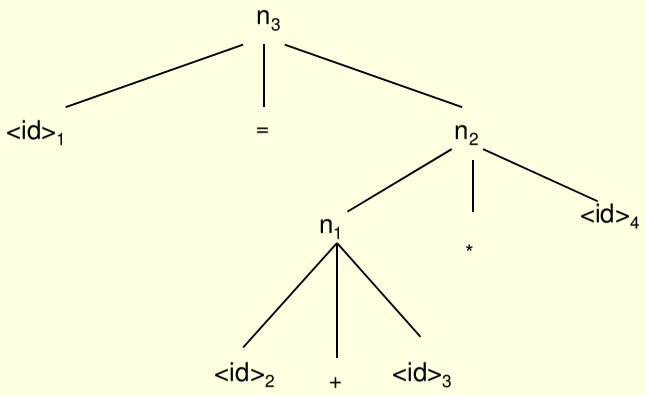
- Секој нов идентификатор кој ќе се најде се проверува дали се наоѓа во табелата
  - Ако го нема се додава
  - Ако е таму, за новото појавување се користат истите дефиниции, и може да се додадат дополнителни параметри
- При конструкцијата на табелите треба да се води сметка за нивното рапидно растење

Парсирање

- Влез во парсерот е низата од токени
- Треба да се утврди редоследот по кој се вршат операциите
- Се користи множеството од синтаксички правила, за да се конструира автоматски парсер
- Се согледува дали програмата е точно напишана
- Има повеќе техники за парсирање и алгоритми за генеририрање на парсери од дадена граматика
- Во оваа фаза се генерира парсирачко дрво

Парсирање

- Пр. <id>$_1$=(<id> $_2$+ <id>$_3$)* <id>$_4$



Генерирање на код

- Дрвото кое е изградено од страна на парсерот се користи за да се генерира код во машински јазик
  - Најчесто тоа не е машински јазик, туку асембли јазик, како измеѓу јазик
- Ни треба познавање на некој асембли јазик
- Фазата на конструкција на дрвото и генерирање на код обично оди заедно
- За генерирање на кодот може да се користи било кој јазик кој ви е познат, Java, C++, Pascal.

Оптимизација на код

- Под оптимизација на код се подразбираат било какви обиди со кои ќе се постигне код кој побргу би работел
- Има различни техники за оптимизација на кодот

Анализа и поправка на грешки

- Не можеме де претпоставиме дека програмата што е напишана во нашиот јазик нема да има грешки
- Дури и ако програмата се изврши, може да се јават грешки во резултатот
- Багови, непретпоставени излези или влезови во некоја процедура
- Техники кои ќе даваат
  - Каде е појавена грешка
  - Каков тип на грешка
  - Ќе даваат упатство што фали или што е вишок
- Техники за поправка
  - За да застанува копајлирањето при секоја грешка, може да се направат соодветни претпоставки и поправки (не на самиот код) и да се продолжи со компајлирање
  - Да се јават повеќе грешки при едно компајлирање

Историја на компајлери

- Процедурално програмирање
- Функционално програмирање
- Објектно ориентирано програмирање

Процедурално програмирање

- FORTRAN 1957 од IBM –за математички проблеми
- Algol 60 –доцни педесети-универзален јазик
- COBOL-процесирање на податоци
- PL/I-комбинација
- Algol 68
- Pascal-Wirth-моќен и лесен
- Modula2
- C-системско програмирање
- Ada-голем и комплексен

Функционално програмирање

- Базирани на апстрактни модели на програмирање (Тјурингова машина)
  - LISP
  - Scheme
  - SASL
  - SML

Објектно ориентирано програмирање

- Simula
- SmallTalk
- CLU
- C++
- Effel
- Java
- Kevo

Историја на компајлери-временски

- 1946, Konorand Zuse-*Plankalkul*
- Short Code-prv компајлиран и прв кој се користи во електронски компјутери
- A-0, 1951, Grace Hopper, кој работел за Remington Rand, прв познат компајлер за MATH-MATIC
- 1957, FORTRAN, John Backus
- 1958, LISP, John McCarthy и ALGOL, John Backus
- 1959 LISP1.5, COBOL, ALGOL60
- 60-ти, LOGO, SNOBOL, BASIC, PASCAL…
- 70-ти, првите значајни програми и имплементации на сериски компјутери

Историја на компајлери-временски

- 1972-манускрипта за Plankalkul
- 1975-TinyBASIC од Bob Albrecht и Dennis Allison (имлементирани од Dick Whipple и John Arnold) работи на микрокомпјутер.
- 1984-прв компајлер за C и C++ на микрокомпјутер
- Сега, актуелна .NET платформата од Microsoft.