Езикови процесори. Въведение

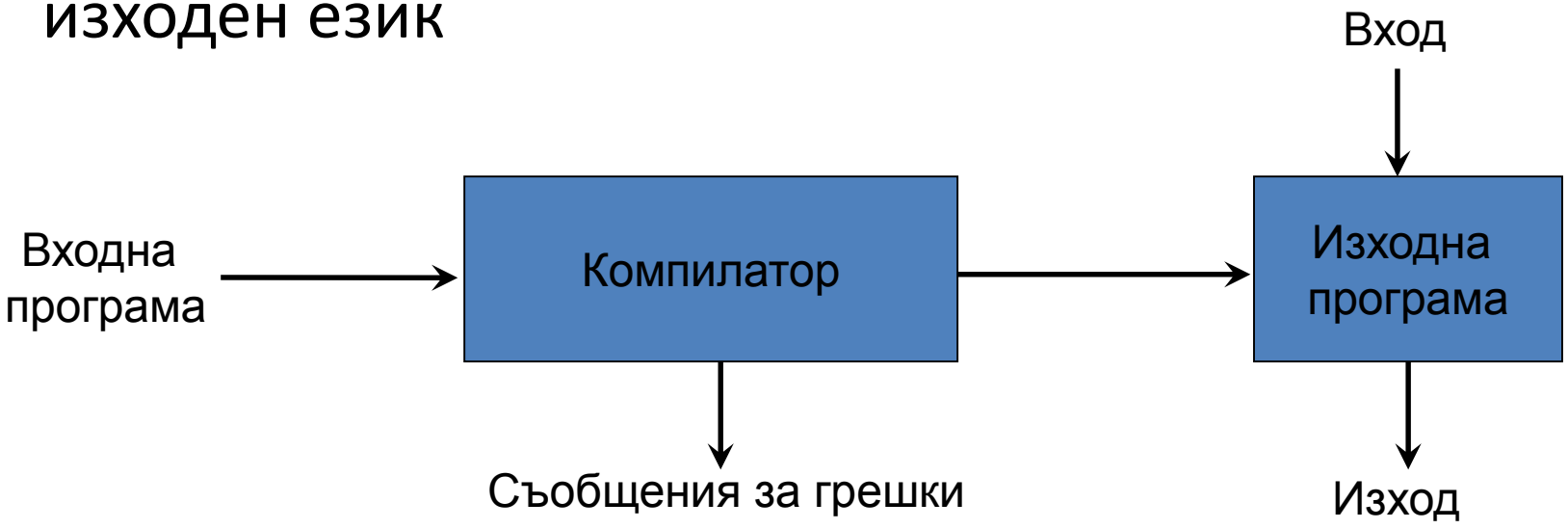
Езиков процесор

- Езиковият процесор е програма, която чете програма, написана на един език (*входен език*) и я превежда (транслира) в еквивалентна програма на друг език (*изходен език*).

- Важна роля на езиковия процесор е да регистрира и извежда съобщения за грешки, открити във входната програма по време на транслацията.

Компилатори и интерпретатори

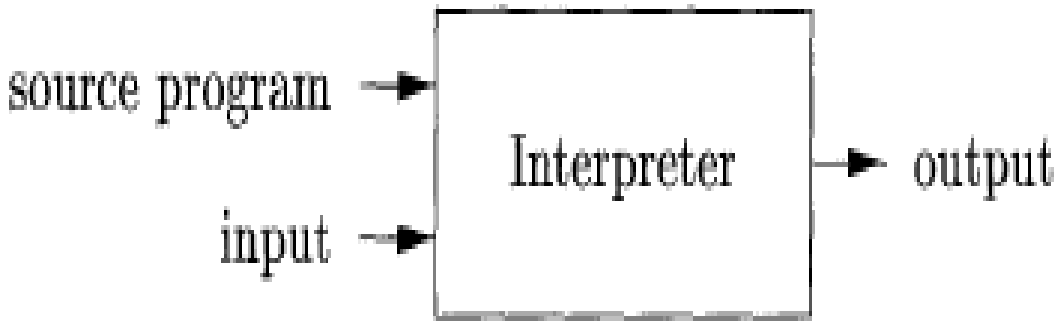
- *“Компилиране”*
  - Транслиране на програма, написана на входен език, в семантично еквивалентна програма на изходен език



Ако изходната програма е изпълнима програма на машинен език, тя може да бъде стартирана от потребител за обработване на входни данни и генериране на резултати

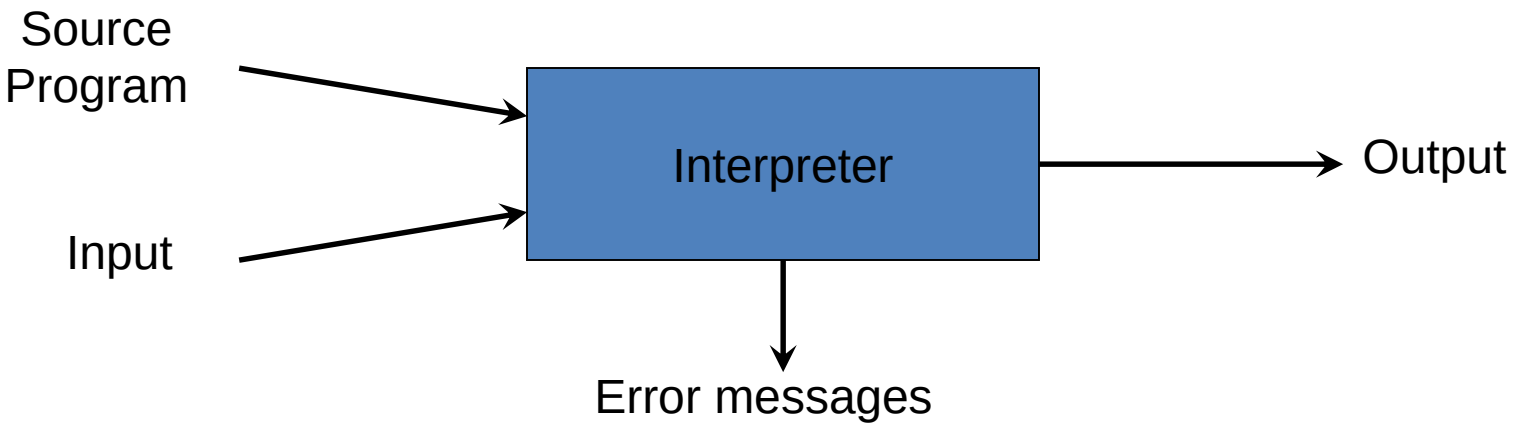
Интерпретатор

Интерпретатор е друг вид езиков процесор. Вместо да генерира изходна програма интерпретаторът директно изпълнява операциите, зададени във входната програма с въведени входни данни.



Компилатори и интерпретатори (продължение)

- *“Интерпретиране”*
  - Изпълнение на операциите от входната програма

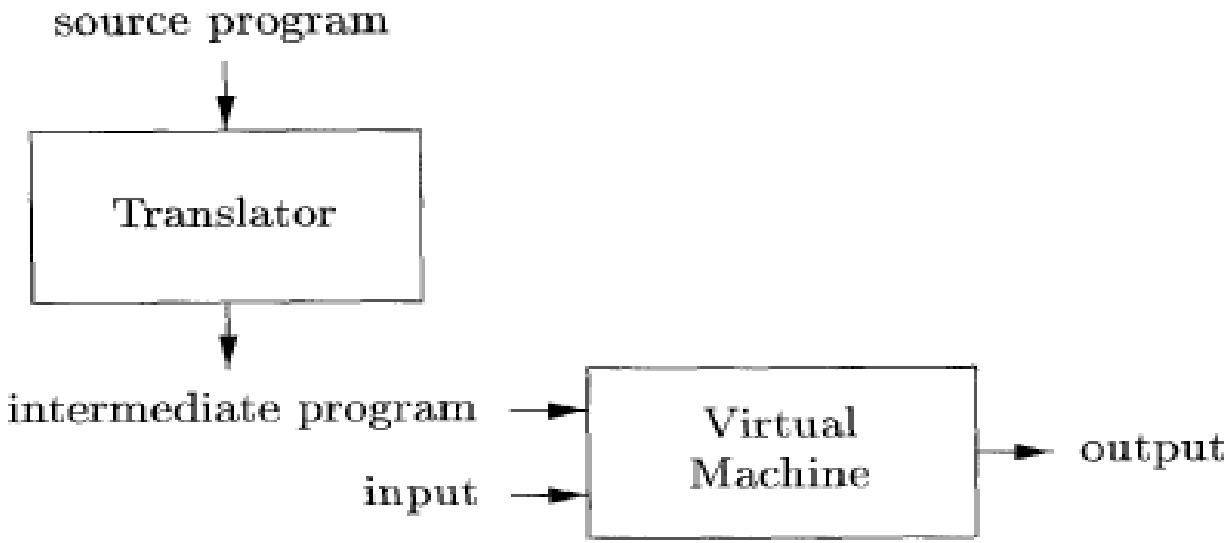


Компилатори и интерпретатори (сравнение)

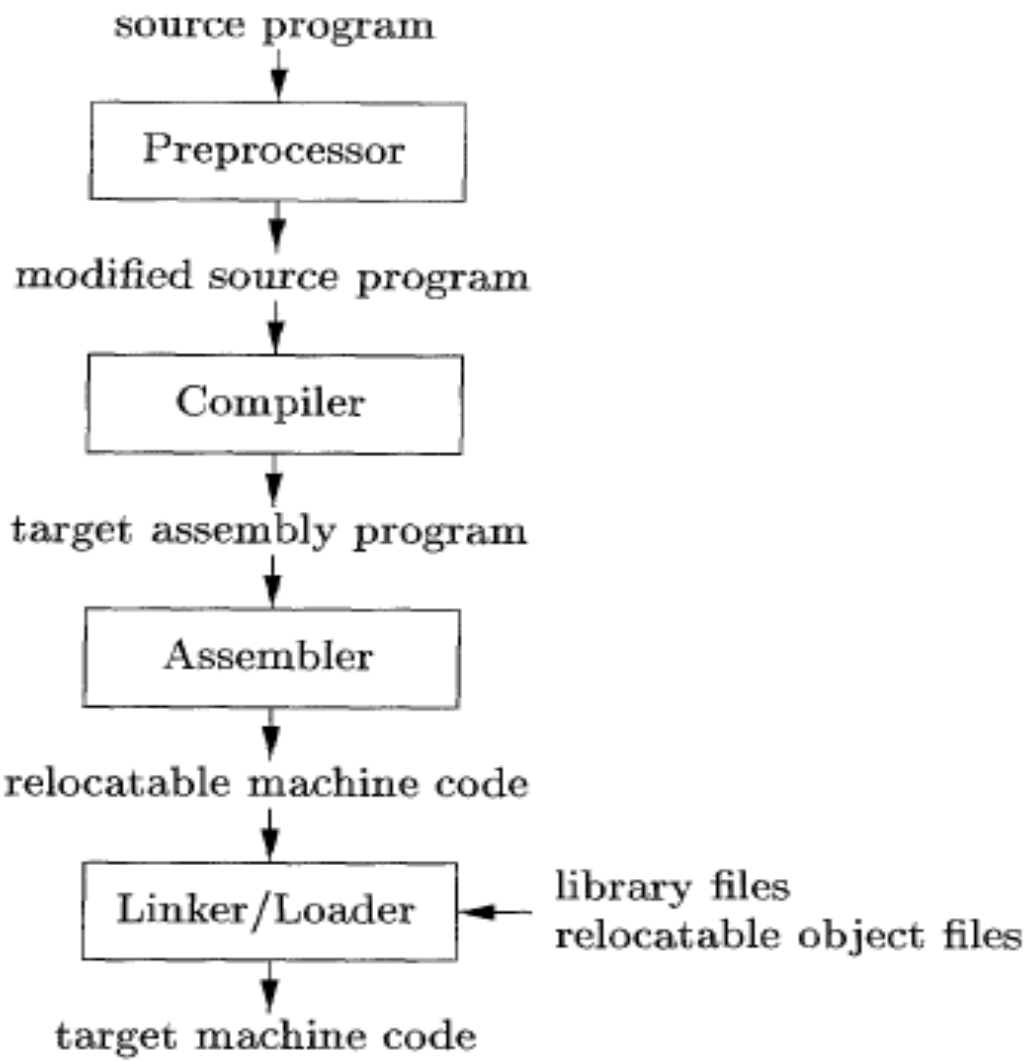
- Компилаторите генерират програми, които се изпълняват много по-бързо в сравнение с програмите, изпълнение от интерпретатори.
- Интерпретаторите предоставят по-добра диагностика на грешки в сравнение с компилаторите, защото изпълняват програмата оператор по оператор.

Хибриден компилатор

Хибридните компилатори комбинират компилиране и интерпретиране. Пример за хибриден компилатор е езиковият процесор на Java. Входната програма първо се компилира до междинен код, нар. *байт код (bytecode).* Байт кодът в последствие се интерпретира от виртуална машина. Основната полза е, че веднъж компилиран на една машина, байт кодът може да бъде изпълняван на всяка друга машина, разполагаща с виртуалната машина на Java.



Обща схема на езиков процесор



За създаване на изпълнима изходна програма са нужни следните системни средства (програмни инструменти):

- Препроцесор (Preprocessor) - Препроцесорът обединява програмен код от различни модули, съхранявани като отделни файлове. Може да преобразува кратки програмни фрагменти, нар. макроси, в оператори на входния език. Препроцесорът модифицира входната програма.
- Компилаторът – Компилаторът генерира програмен код на асемблер, защото асемблерният код е по-лесен за генериране и за трасиране (debug).
- Асемблер (Assembler) – Генерира машинен код, който е премества в адресното пространство (относителни адреси).
- Свързваща програма (Linker/Loader) - По-големите програми обикновено се компилират на отделни части. В резултат се налага преместваемият машинен код да бъде свързан с други премества обектни и библиотечни файлове за получаване на код за изпълнение от дадена машина. Свързващата програма (linker) изпълнява т.нар. резолиране на външни адреси, т.е. насочва адресите от даден модул към външни модули. Зареждащата програма (loader) позиционира всички изпълними обектни файлове в паметта за изпълнение.

Структура на компилатор

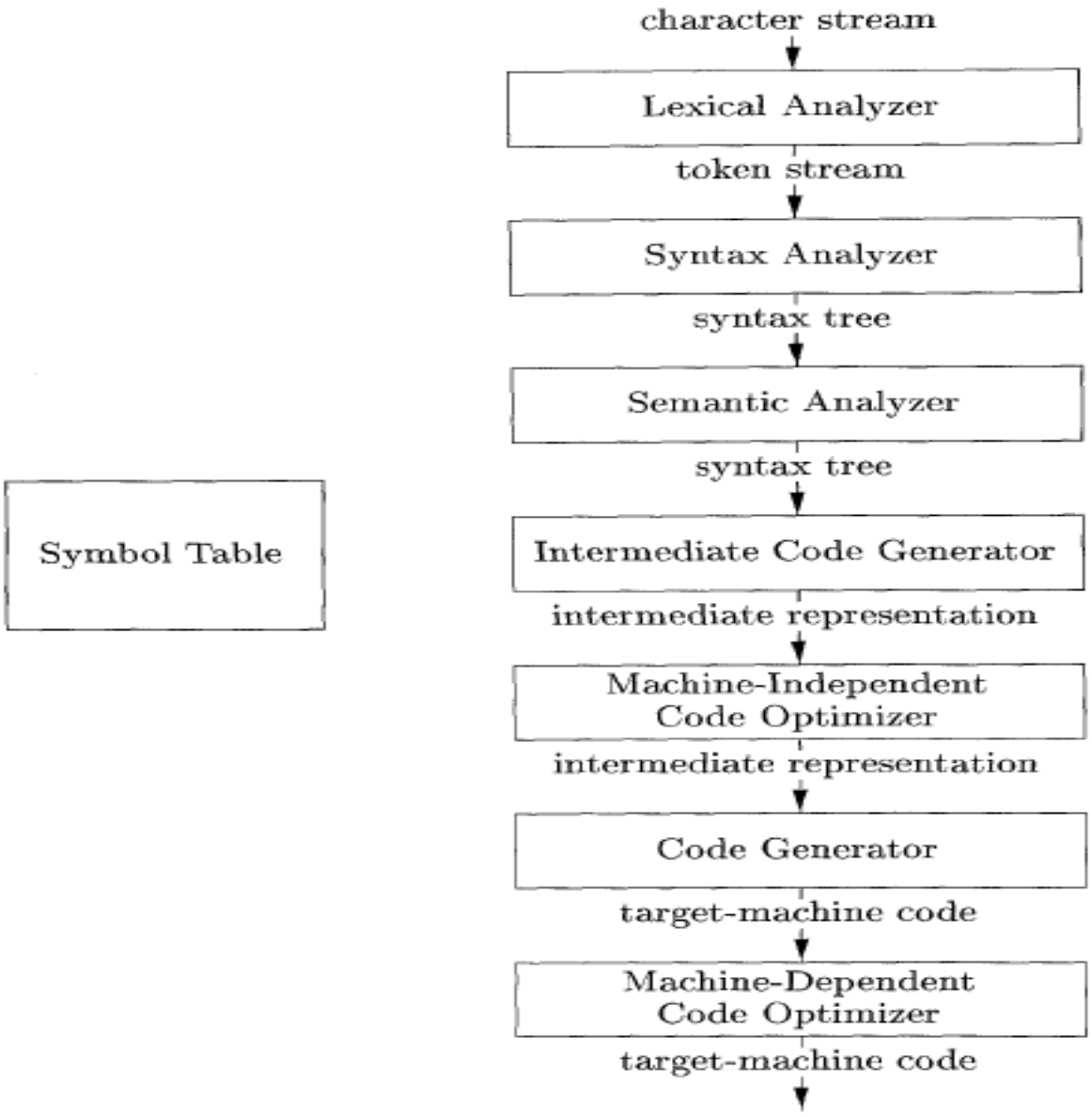
Процесът на компилиране условно преминава през два етапа : *анализ* и *синтез*.

- **анализ -** Проверява входната програма за съответствие с граматическата структура на езика и създава междинно представяне (междинна форма) на програмата. По време на анализа също се събира информация за входната програма, която се съхранява в т.нар. **символна таблица**. Тази таблица заедно с междинната форма се изпраща за синтез.
- **синтез** - Синтезът построява изходна програма от получените междинна форма и символна таблица

Често анализът се нарича **front end** на компилатора, а синтезът **back end**.

- Разгледан в детайли, процесът на компилиране се изпълнява като последователност от етапи, във всеки от които едно представяне (една форма) на входната програма се трансформира (преобразува) в друго представяне (друга форма).
- На практика няколко етапа мога да бъдат групирани (обединени).
- Символната таблица, която съхранява информация за цялата входна програма, се използва във всички етапи на компилатора.

Етапи на компилирането



Лексически анализ

Лексическият анализатор чете поток от символи, съставящи входната програма и обединява символите в поток от значещи последователности, нар. лексеми или думи (tokens). За всяка лексема лексическият анализатор генерира двойка стойности:

<клас на лексемата(token-name), стойност(attribute-value)>

Клас на лексемата се използва при синтактичния анализ

Стойност сочи към елемент от символната таблица. Информацията от символната таблица се използва при семантичния анализ и генериране на код.

Потокът от лексеми се изпраща към следващия етап – синтактичен анализ.

Лексемите се наричат също *символи на речника на езика*.

Примери за лексеми, разпознавани на етапа на лексическия анализ:

- Идентификатори – последователности от букви и цифри
- Числа – последователности от цифри
- Оператори/разделители – последователности от специални символи

Синтактичен анализ

- Потокът от лексеми се представя във форма, която отразява синтактичната структура на входната програма и позволява лесно разпознаване на тази структура. Този етап се нарича *синтактичен анализ* (или *парсиране - parsing*) .
- Синтактичният анализатор (parser) използва класовете лексеми, получени от лексическия анализатор, за построяване на дървовидно представяне на входната програма, описващо нейната граматическа структура.
- Типичното представяне е във вид на синтактично дърво, в което даден представя операция, а неговите наследници листа представляват аргументите (операндите) на операцията.
- Полученото синтактично дърво, отразяващо граматическата структура, се използва от следващите етапи на компилатора за семантичен анализ и генериране на изходна програма.

Семантичен анализ

- Семантичният анализатор използва синтактичното дърво и информацията от символната таблица, за да провери дали семантиката (смисълът) на входната програма съответства на дефинициите на входния език. Семантичният анализатор също събира информация за използваните типове данни и я съхранява в синтактичното дърво или в символната таблица за последващо използване при генерирането на код.
- Важна част от семантичния анализ е проверката за съвместимост на типове, при която компилаторът проверява дали всеки оператор има операнди от правилен тип. Пример: В повечето програмни езици индексите на масив трябва да са целочислени стойности. Следователно компилаторът трябва да регистрира грешка, ако за индекс се използва нецяло число.

Генериране на междинен код

След синтактичен и семантичен анализ на входната програма много компилатори генерират междинно представяне на програмата, което се нарича **код за абстрактна (виртуална) машина**. Този код трябва да изпълнява две важни изисквания:

- Трябва да бъде лесен за генериране;
- Трябва да бъде лесен за транслиране в код за дадена реална машина.

Едно често използвано междинно представяне е три адресен код. Той представлява последователност от асемблерни инструкции с три операнда за всяка инструкция.

Оптимизация на кода

- Оптимизацията на получения машинно независим код се опитва да подобри междинния код по отношение на различни показатели – най-често бързодействие или краткост.

Генериране на код

- Генераторът на преобразува междинното представяне на входната програма в програма на **изходен език**. Ако изходният език е машинен, за всяка променлива от входната програма се избират регистри или адреси от паметта. След това инструкциите от междинния код се транслират в последователности от машинни инструкции със същото действия. Критичен момент при генерирането на код е правилно разполагане на стойностите на променливите в съответни регистри.

Управление на символните таблици

- Основна функция на компилатора е да съхранява имената на променливите, използвани във входната програма и да събира информация за атрибутите на всяка променлива – тип, област на видимост (scope), адресно пространство и др.
- Символната таблица е структура, която съдържа запис за всяко име на променлива с полета за съхраняване на атрибутите на променливата. Тази структура трябва да бъде създадена така, че компилаторът да намира бързо всеки запис от нея, както и бързо да записва и прочита данни.

Обединяване на етапи в компилатора

- Описаните етапи засягат логическата организация на един компилатор.
- При реализацията операциите от няколко етапа могат да бъдат обединени в един „пас“, при който се прочита входна програма и се генериране изходна програма.
- Например front-end етапите на лексически, синтактичен и семантичен анализ и генериране на междинен код могат да бъдат обединени в един пас. Оптимизацията на кода може да бъде незадължителен пас. След това може да има отделен Back-end пас за генериране на код за дадена машина.