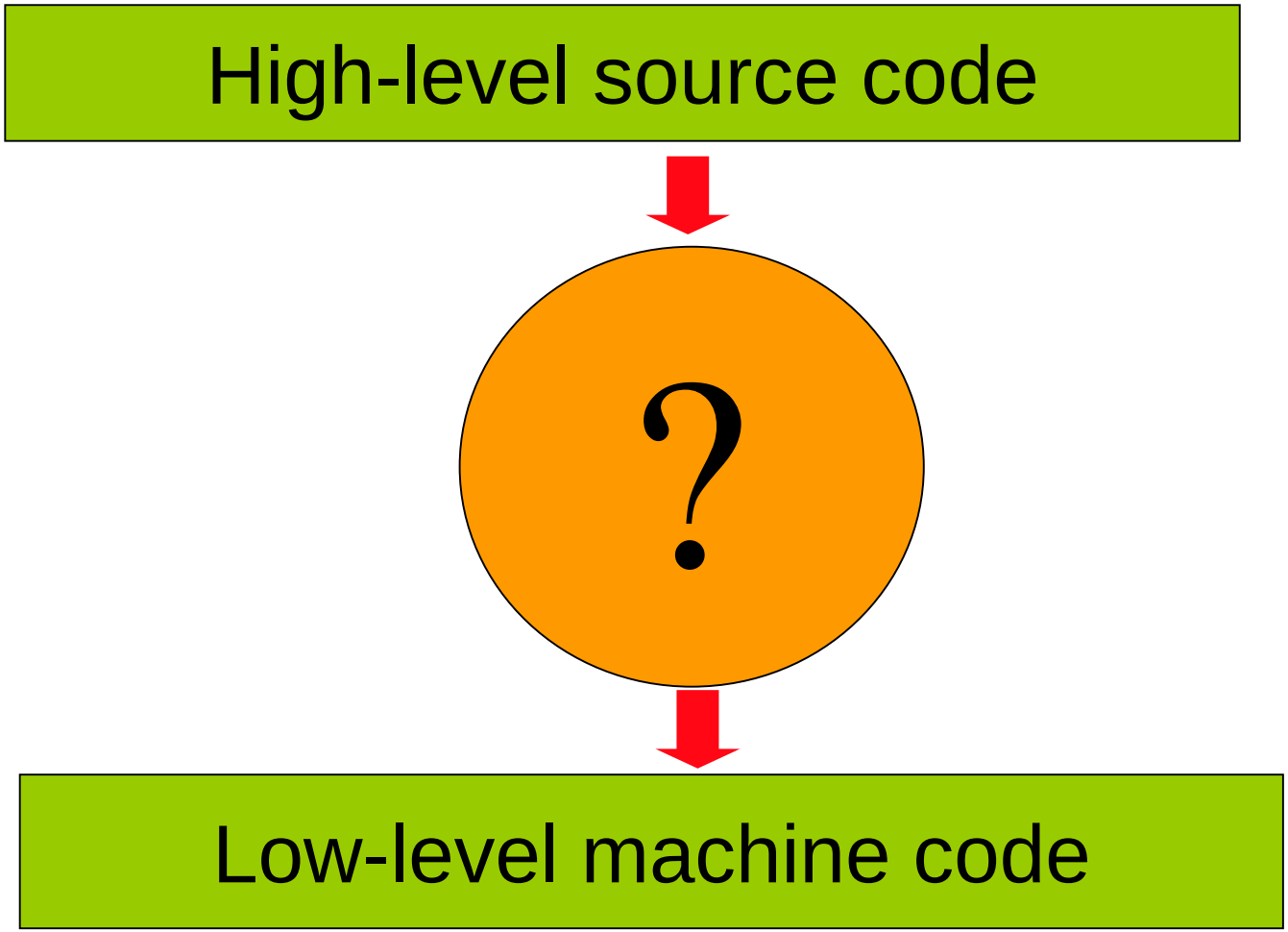
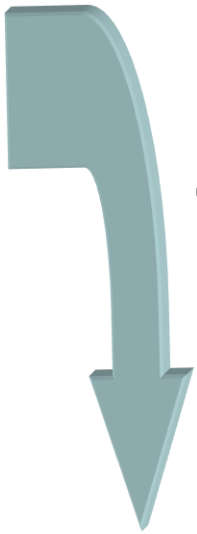
DİL ÇEVRİMİ

- Yüksek düzeyli bir dilde yazılmış bir program ancak makine diline çevrilerek bir bilgisayarda çalıştırılabilir.


High-level source code
?
Low-level machine code

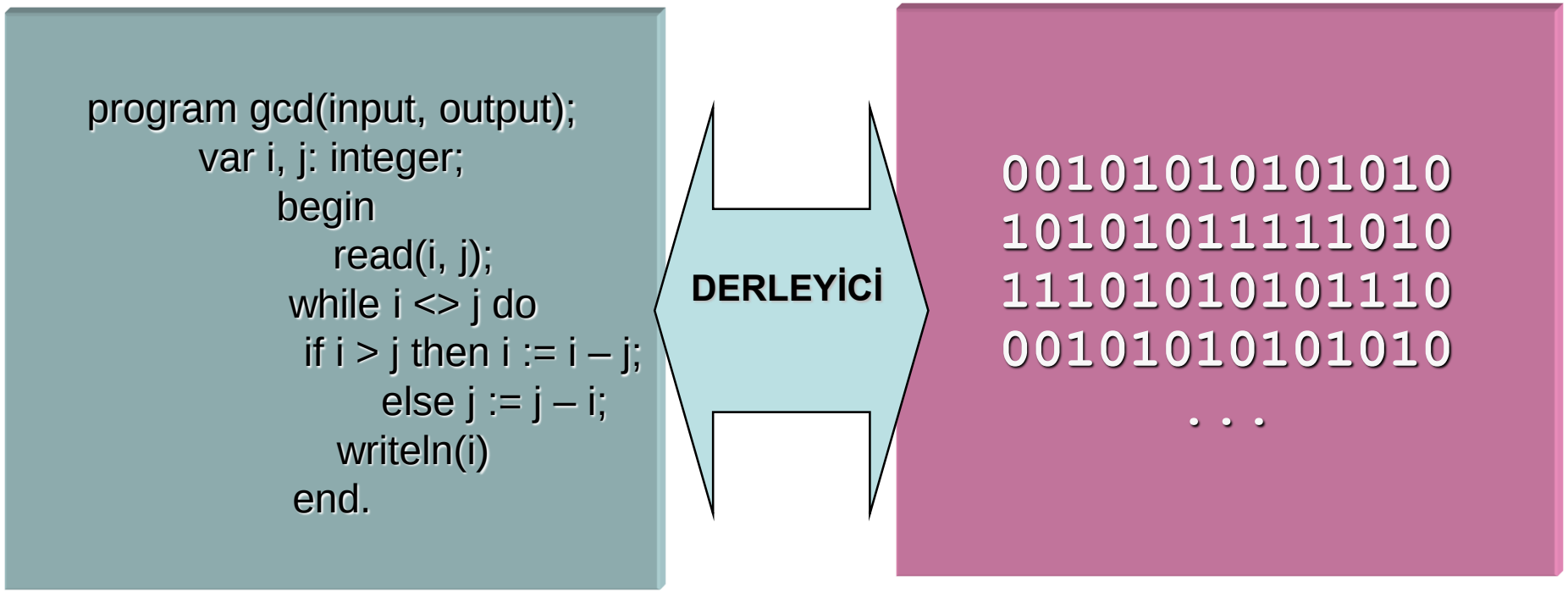
Yüksek düzeyli bir dilde yazılmış kaynak kodun makine diline (Hedef kod) dönüştürülmesi zorunluluğu Dil çevirici yazılımların oluşturulmasına neden olmuştur.

```
program gcd(input, output);
var i, j: integer;
begin
        read(i, j);
        while i <> j do
                if i > j then i := i - j;
                else j := j - i;
        writeln(i)
end.
```



Compilation

```
27bdffd0 afbf0014 0c1002a8 00000000 0c1002a8 afa2001c 8fa4001c
00401825 10820008 0064082a 10200003 00000000 10000002 00832023
00641823 1483fffa 0064082a 0c1002b2 00000000 8fbf0014 27bd0020
03e00008 00001025
```

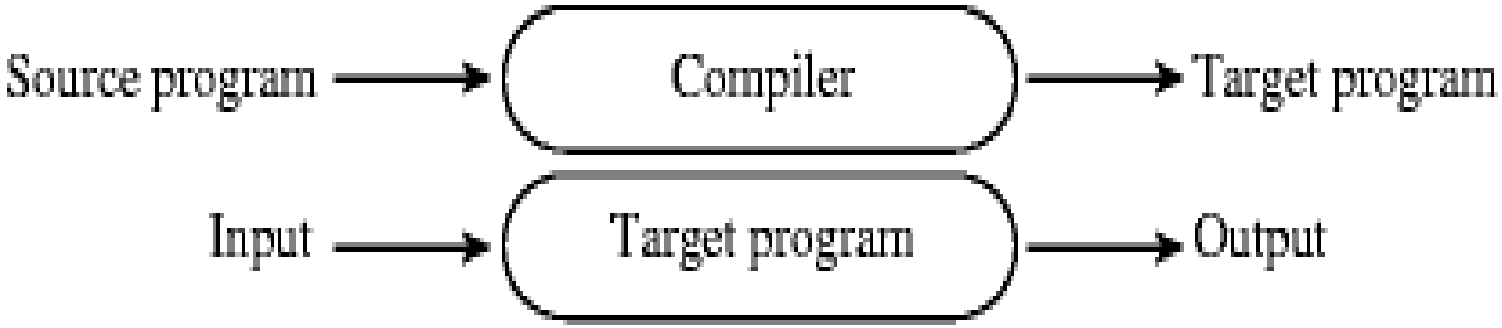

program gcd(input, output);
var i, j: integer;
begin
read(i, j);
while i <> j do
if i > j then i := i – j;
else j := j – i;
writeln(i)
end.
DERLEYİCİ
00101010101010
10101011111010
11101010101110
00101010101010
. . .


Source language program
Translator
Object language program

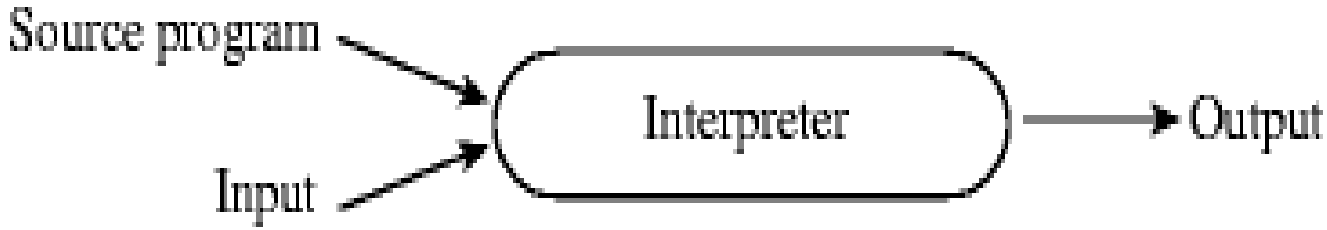
- Dönüşüm için **derleme** ve **yorumlama** olmak üzere iki temel yöntem vardır.

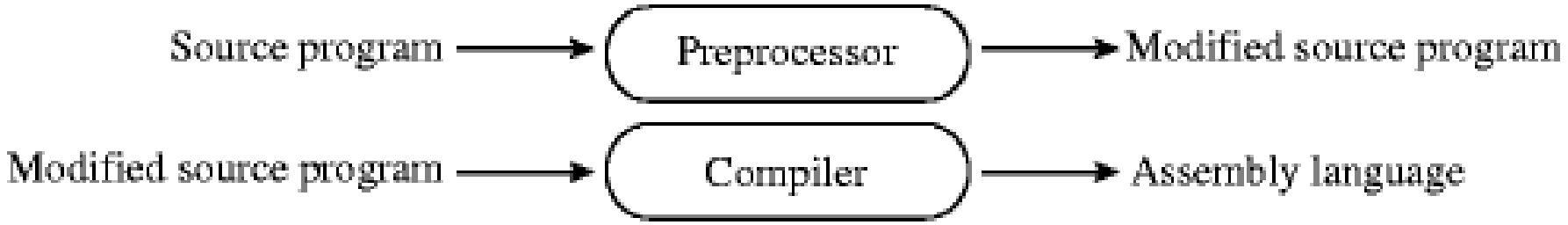
- Bir **yorumlayıcı**, bir programın her deyimini birer birer makine diline çevirerek çalıştırır.

- D**erleyici**, bir programlama dilinde yazılmış bir kaynak kod için o koda eşdeğer olan makine dilinde bir program oluşturur.

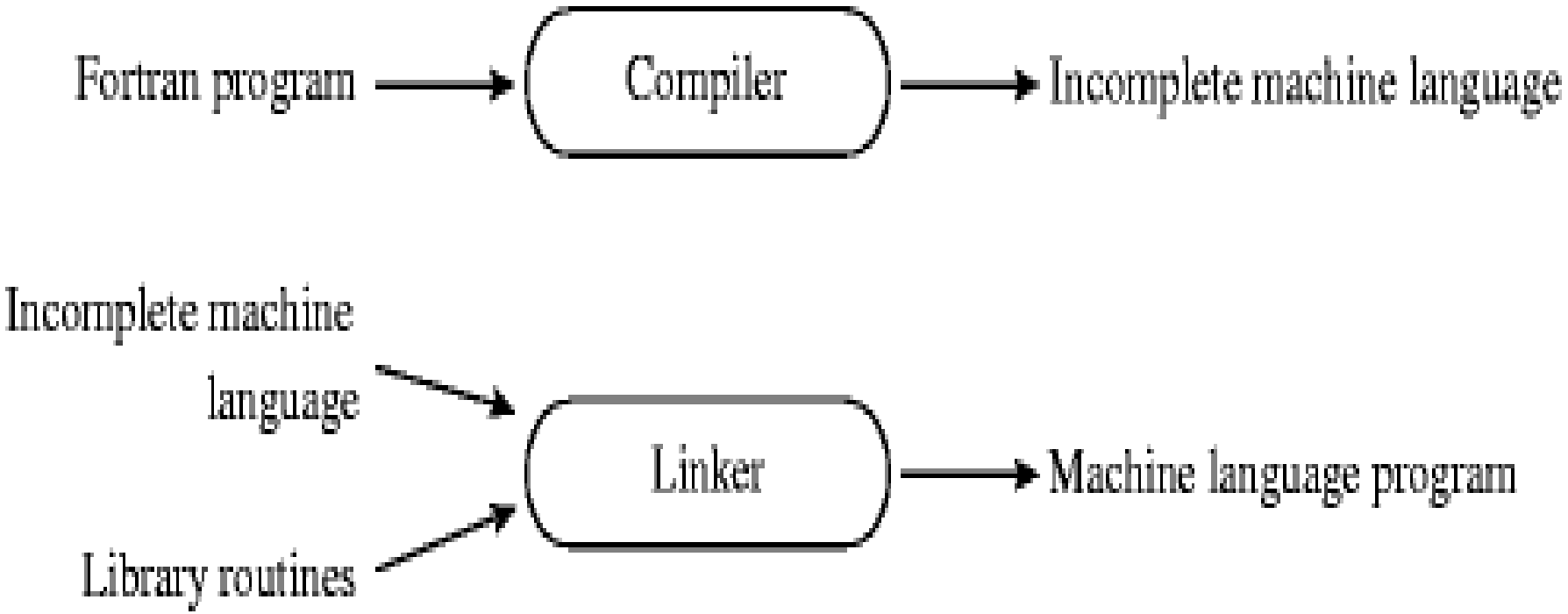


- Yorumlayıcı






```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
```


Fortran program
Compiler
Incomplete machine language
Incomplete machine language
Library routines
Linker
Machine language program

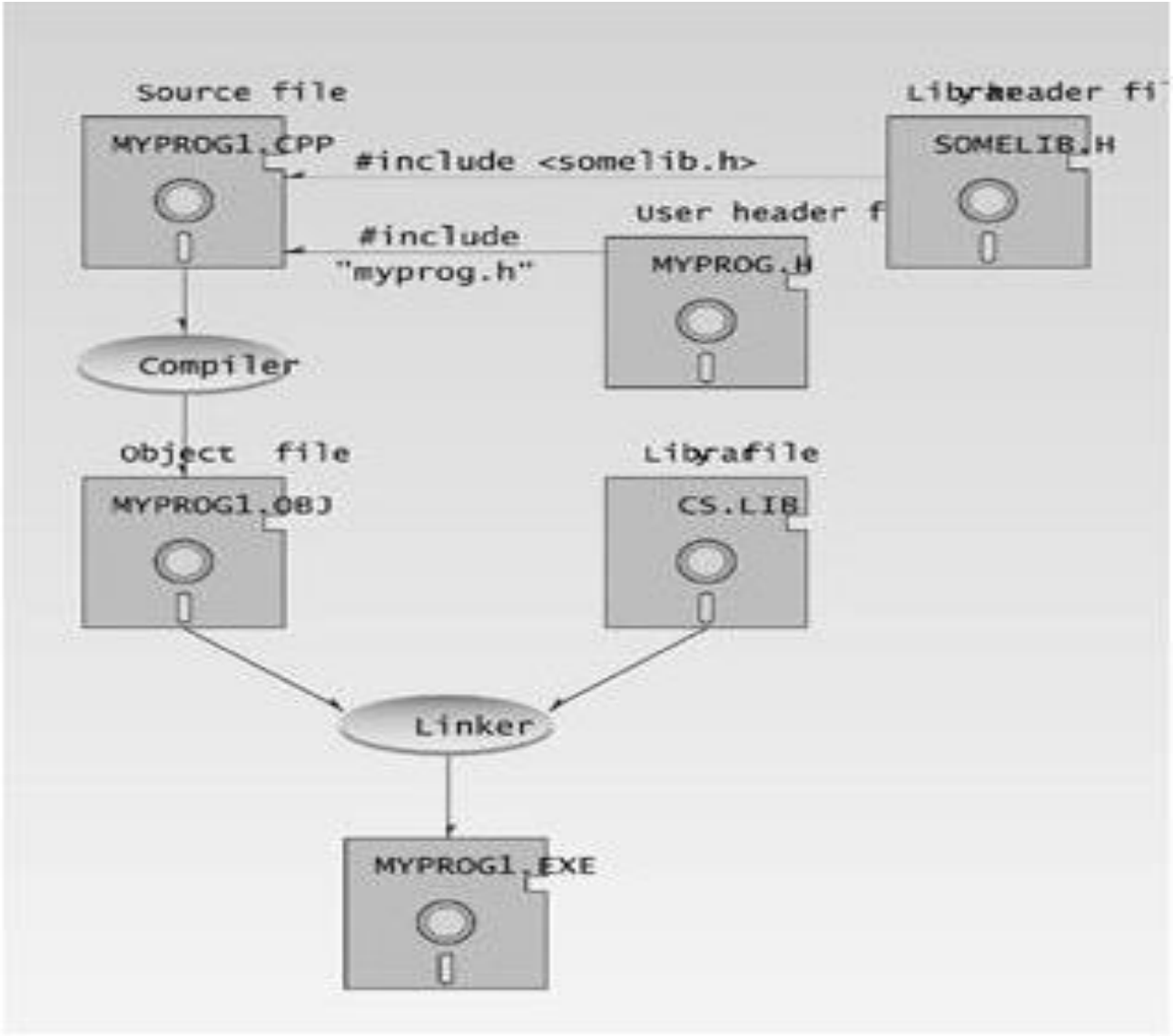


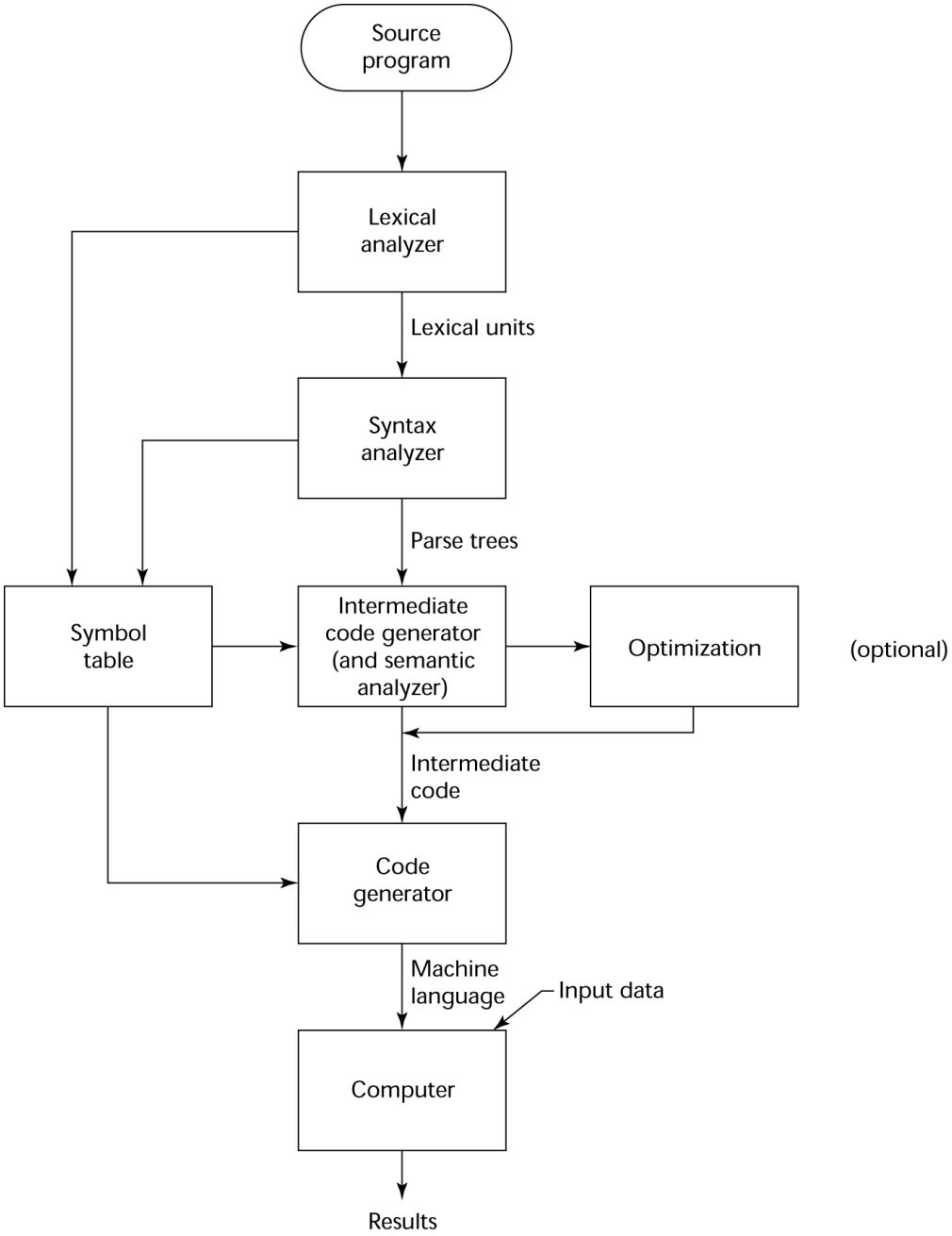
Figure 2.12 Header and library files.

Derleme Süreci

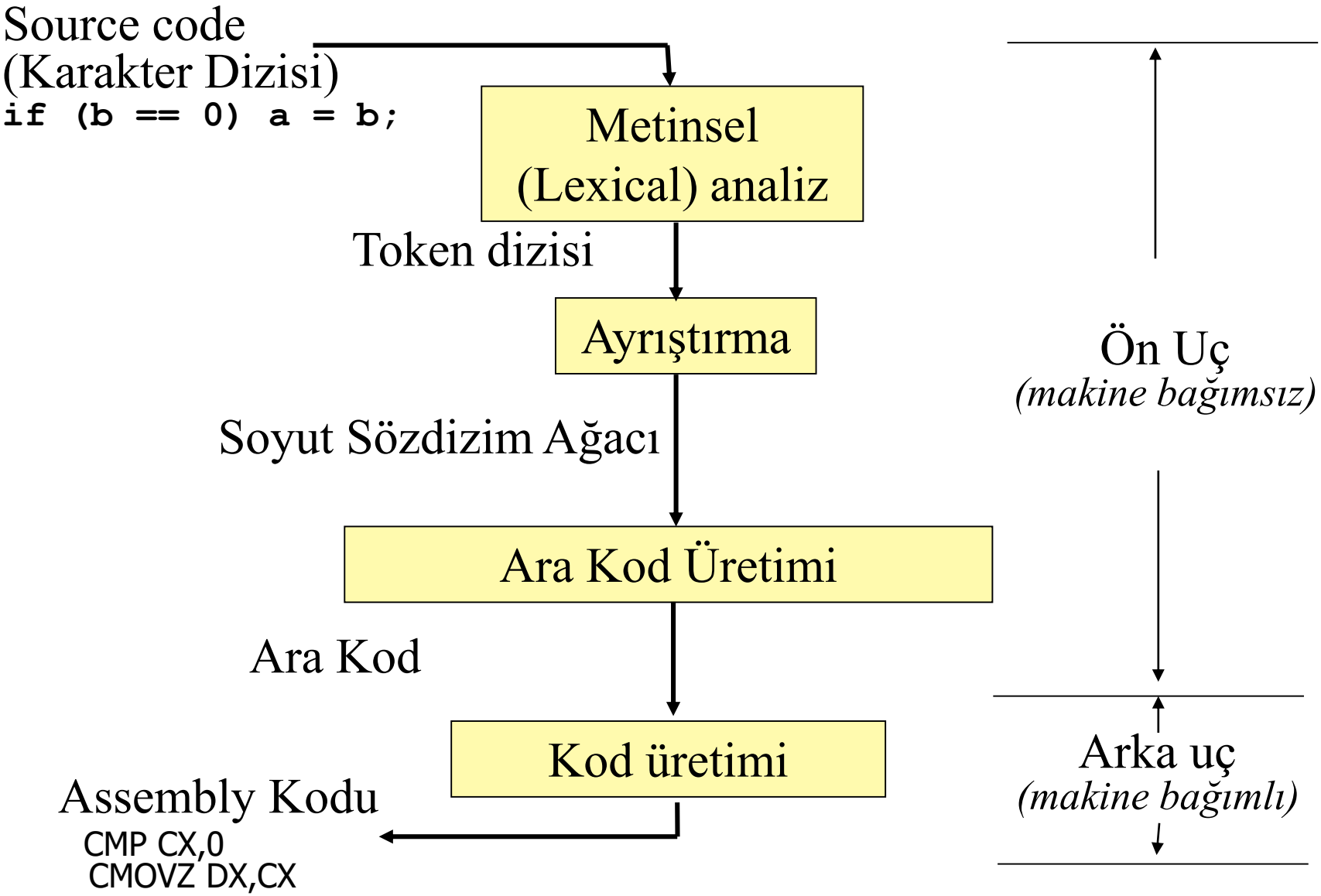
- Derleme sürecinin başlangıcında derleyiciye verilen yüksek düzeyli bir programlama dili deyimlerini içeren programa, **kaynak** (*source*) **program**, derleme sürecinin sonucunda oluşan makine dilindeki programa ise **amaç** (*object*) **program** adı verilir.

Derleme süreci

- Derleyicinin çalışması sırasında geçen zamana **derleme zamanı** (*compile time*) denir.
- Hedef programların çalışması sırasında geçen zamana **çalışma zamanı** (*run time*) adı verilir.


Source program
Lexical analyzer
Lexical units
Syntax analyzer
Parse trees
Symbol table
Intermediate code generator (and semantic analyzer)
Optimization
(optional)
Intermediate code
Code generator
Machine language
Input data
Computer
Results

Sadeleştirilmiş Derleyici Yapısı

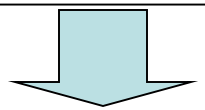


Metinsel Analiz-Scanner

- Derleme sürecindeki ilk ve en uzun süren aşamadır.
- Bir derleyicinin ön ucunda yer alan **metinsel çözümleyici** (*lexical analyzer*), bir kaynak programı bir dizi *token dizisine* çevirir.
- Regüler ifade aracı kullanır

Scanner

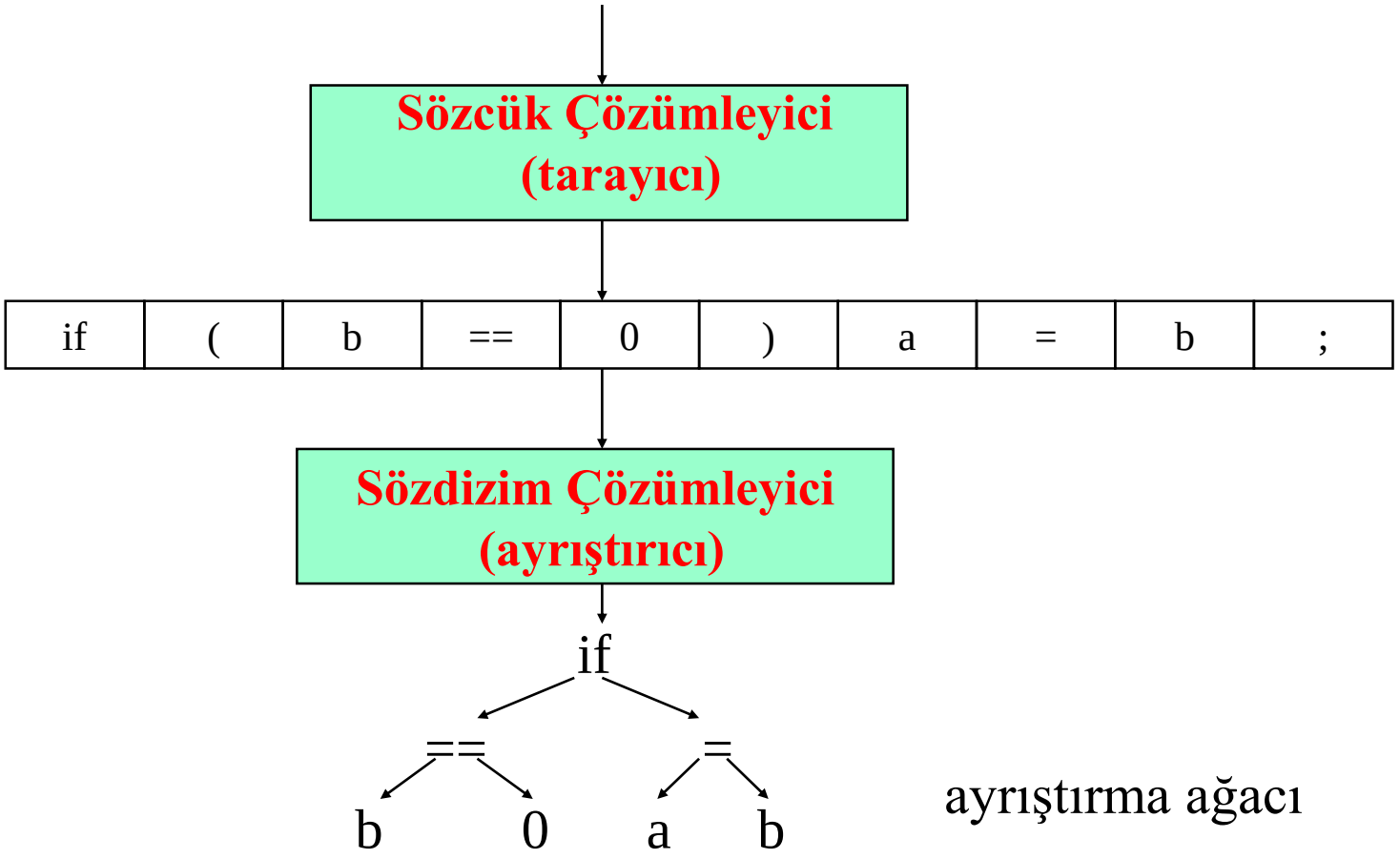
```
program gcd (input, output);
var i, j : integer;
begin
  read (i, j);
  while i <> j do
    if i > j then i := i - j else j := j - i;
  writeln (i)
end.
```



program	gcd	(	input	,	output	)	;
var	i	,	j	:	**integer**	;	**begin**
read	(	i	,	j	)	;	**while**
i	<>	j	**do**	**if**	i	>	j
then	i	:=	i	-	j	**else**	j
:=	i	-	i	;	writeln	(	i
)	**end**	.					

Syntax Analiz

if (b == 0) a = b;

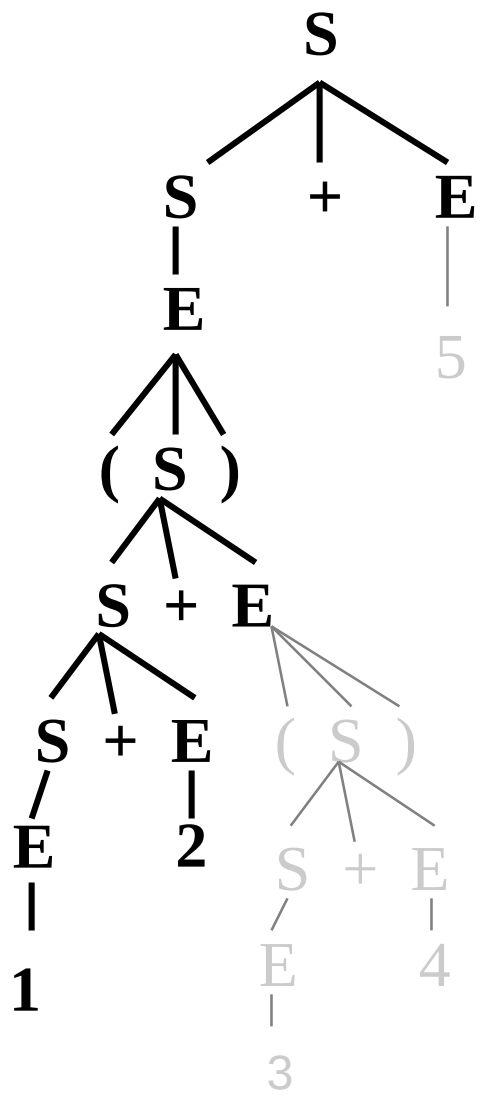


ayrıştırma ağacı

Ayrıştırma (Parser) yöntemleri

- Ayrıştırıcı bir sözcük katarının dilin gramer kuralları ile türetilebilirliğini inceler ve sözcük katarına ilişkin bir ayrıştırma ağacı oluşturur
- İki yol vardır.
  - Yukarıdan-aşağıya (top-down)
  - Aşağıdan-yukarıya (bottom-up)

Top-Down



Sembol Tablosu

- Derleme sürecinde programdaki her tanımlayıcı için bir eleman içeren **sembol tablosu** oluşturulur. Sembol tablosu, derleme sürecindeki çeşitli aşamalarda kullanılır ve güncellenir.

- Bir tanımlayıcı kaynak programda ilk kez bulunduğunda, o tanımlayıcı için sembol tablosunda bir eleman oluşturulur. Aynı tanımlayıcının daha sonraki kullanımları için ilgili *token*, aynı sembol tablosu elemanına başvuru içerir.

- Metinsel çözümleme aşamasının sonunda, programdaki *token*'lar ve her *token*'ın özelliklerinin tutulduğu sembol tablosu elemanına işaret edilen göstergeleri içeren *token* dizisi oluşturulur

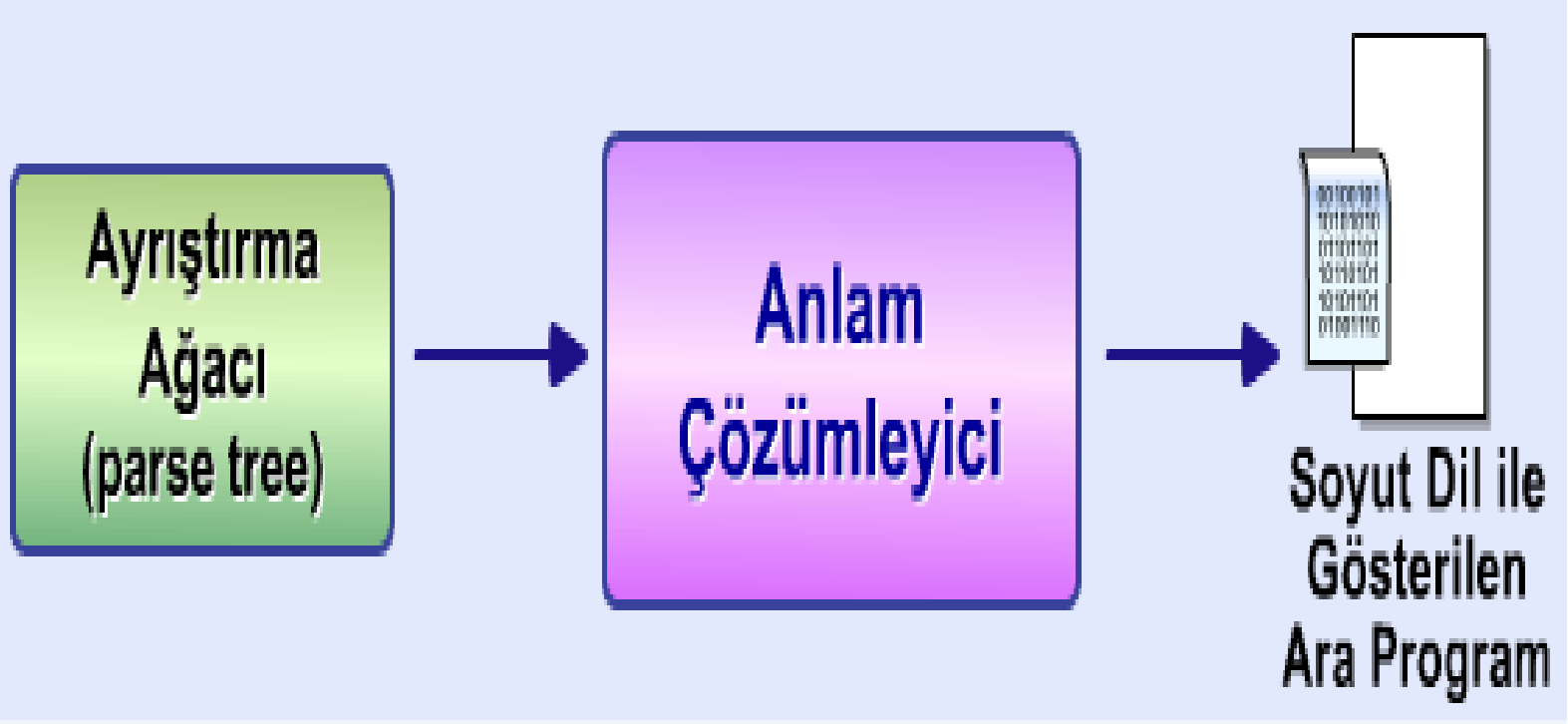
Sembol tablosu

Ortalama := toplam / 10		
Ortalama	(identifier,1)	tanımlayıcı
:=	(atama,nil)	Atama işlemcisi
toplam	(identifier, 2)	Tanımlayıcı
/	(bolme,nil)	Bölme işlemcisi
10	(tamsayı, 3)	tamsayı

Anlam Çözümleme

- **Anlam çözümleme**, kaynak program için sözdizim çözümleme sırasında oluşturulmuş ayrıştırma ağacı kullanılarak, soyut bir programlama dilinde bir program oluşturulmasıdır.

Anlam Çözümleme



Soyut Dil

- Anlam çözümleme sonucunda üretilen kod için kullanılan ara diller, genel olarak, üst düzeyli bir birleştirici diline benzerler. Bu soyut dil, kaynak dilin veri türleri ve işlemleriyle uyumlu olacak şekilde tasarlanmış, hayali bir makine için bir makine dili olup, derleyicinin kaynak ve amaç dilleri arasında bir ara adım oluşturur.

Kod Oluşturma

- Soyut dilde ifade edilen kodu alınarak belirli bir bilgisayar için makine kodunu oluşturulur.
- Derleyicinin ön ucu programlama diline bağımlı, arka ucu ise bilgisayara bağımlıdır.

Eniyileme (Optimizasyon)

- İsteğe bağlı olarak ara kod kısmında iyileştirmeler yapılabilir.
- Buradaki çalışma bir programın daha etkin olarak çalışacak bir eşdeğer programa dönüştürülmesi için yapılır.

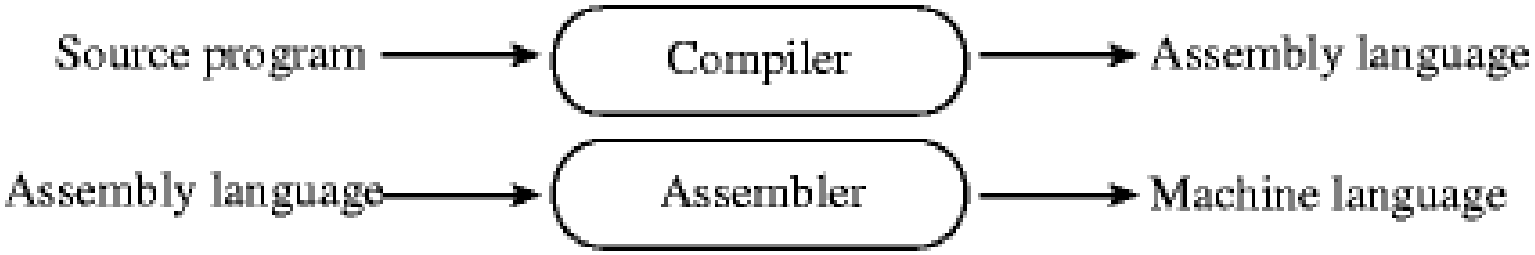
Dil Çevrim Yöntemlerinin Karşılaştırılması

Çevirici	Zaman	Hafıza	Hata Yakalama
Derleyici	+	-	-
Yorumlayıcı	-	+	+

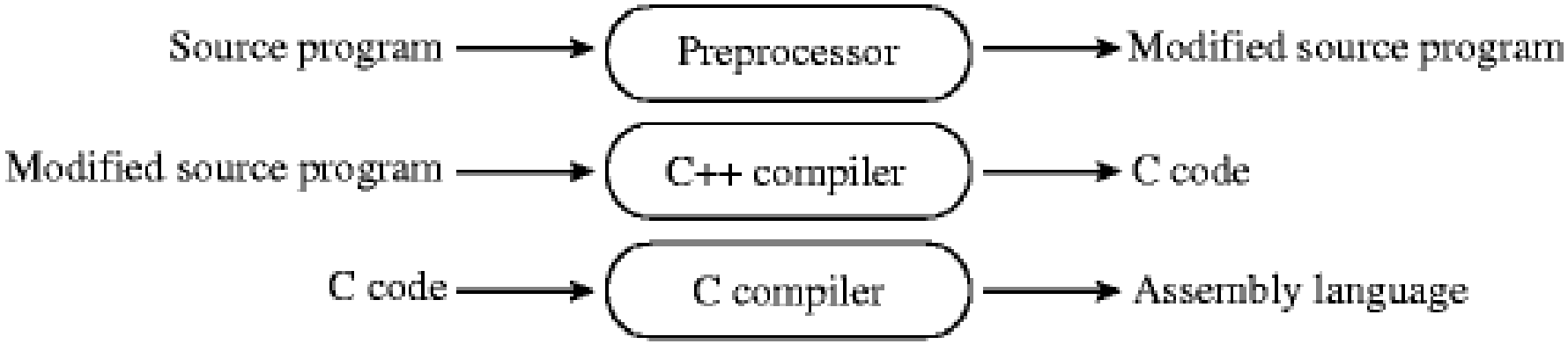
Taşınabilir Kod

- Dil çevriminde bütünüyle yorumlama ve bütünüyle derleme yöntemleri, iki uç durumdur. Bazı programlama dilleri, iki yöntemin birleştirilmesi ile gerçekleştirilirler.
- Bir program, kaynak program üzerinde basit düzenlemeler yapılarak bir sanal makinenin daha sonra yorumlanacak olan makine kodu olarak nitelenebilen bir ara koda çevrilebilir. Bu çözüm, ağırlıklı olarak derlemeye dayanır ve farklı makinelerde çalıştırılabilen **taşınabilir kod** üretmek amacıyla kullanılabilir.

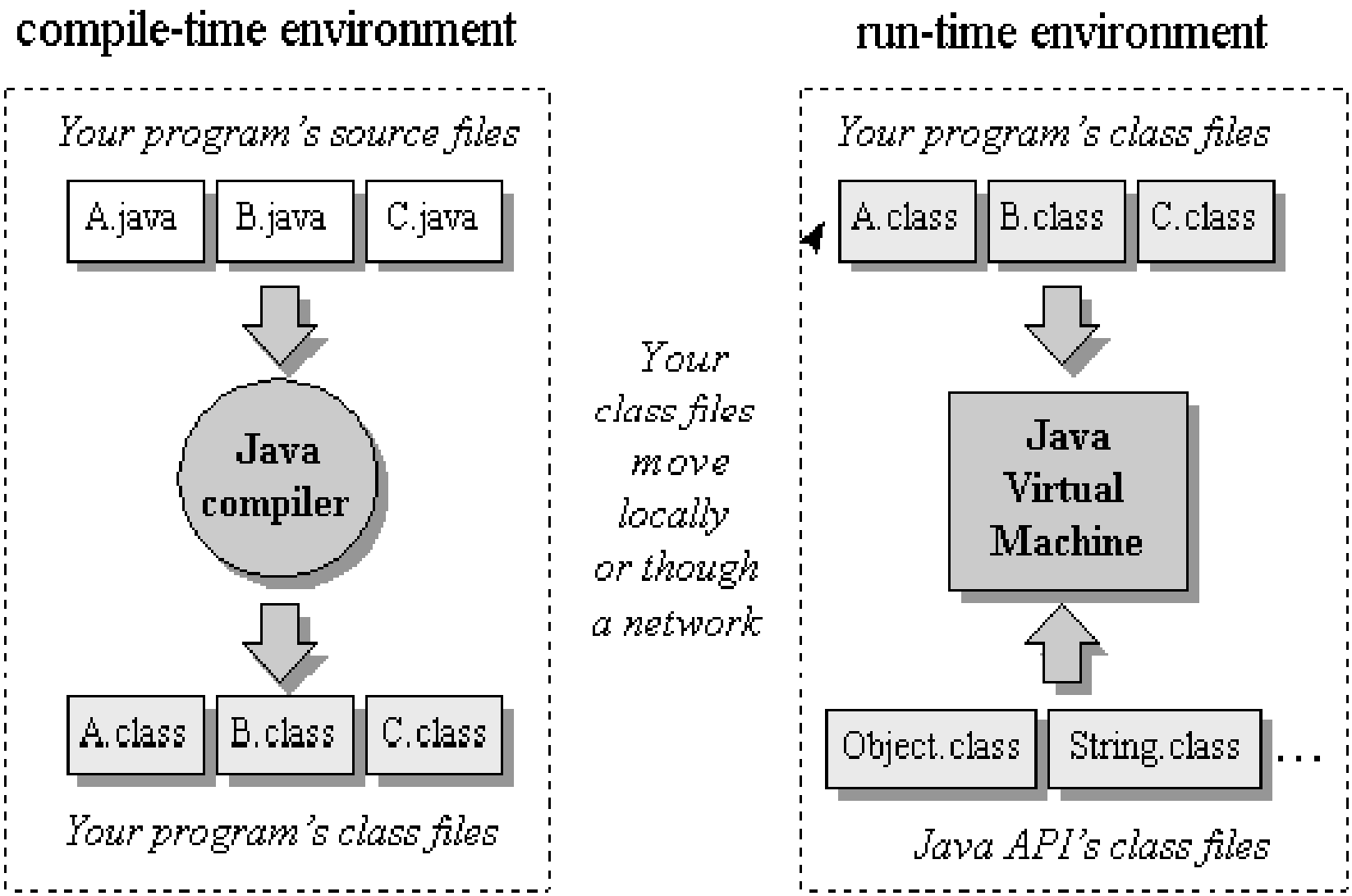
Taşınabilirlik



- Ara kod



- Java dili bu tür dil çevrimini uygulamaktadır. Java programları, Java *bytecode*'u adı verilen bir ara koda dönüştürüldükten sonra yorumlanır.



Örnek

Örnek

```
read A
read B
sum := A + B
write sum
write sum / 2
```

Metinsel Analiz (Lexical Analysis)

- Tokens:

  id = letter (letter | digit) * [except "read" and "write"]

  literal = digit digit *

  ":=", "+", "-", "*", "/", "(", ")“

  $$$ [end of file]

Sözdizim Analizi (Syntax Analysis)

- EBNF

```
<pgm>          ->  <statement list> $$$
<stmt list>    ->  <stmt list> <stmt> | E
<stmt>         ->  id := <expr> | read <id> | write <expr>
<expr>         ->  <term> | <expr> <add op> <term>
<term>         ->  <factor | <term> <mult op> <factor
<factor>       ->  ( <expr> ) | id | literal
<add op>       ->  + | -
<mult op>      ->  * | /
```

Ara Kod Dönüşümü

- Intermediate code:

```
read
pop A
read
pop B
push A
push B
add
pop sum
push sum
write
push sum
push 2
div
write
```

Ara Kod Dönüşümü

- Target code:

```
        .data
A:      .long 0
B:      .long 0
sum:    .long 0
        .text
main:   jsr read
        movl    d0,d1
        movl    d1,A
        jsr read
        movl    d0,d1
        movl    d1,B
        movl    A,d1
```

Kod üretimi

```
movl    B,d2
addl    d1,d2
movl    d1,sum
movl    sum,d1
movl    d1,d0
jsr write
movl    sum,d1
movl    #2,d2
divsl   d1,d2
movl    d1,d0
jsr write
```

Bölüm Özeti

- Dil çevrimi
- Dil çevriminin Aşamaları
- Dil çevrim Yöntemleri
- Dil çevrim yöntemlerinin karşılaştırılması