

PROGRAMLAMA DİLLERİNİN GELİŞİMİ

- Programlama dili tasarım ve gerçekleştirmeleri, 1950'li yıllarda tanıtılan ilk yüksek düzeyli diller olan FORTRAN, COBOL ve LISP'den beri sürekli olarak gelişmiştir.
- Günümüzde hızla değişen bilgisayar teknolojileri, yeni gereksinimleri ortaya çıkarmaktadır. Bunun sonucu olarak, gelecekte de yeni programlama dillerinin geliştirilmesi kaçınılmazdır.

Programlama Dillerinin Sınıflandırılması (Seviyesine göre)

- Çok Yüksek Seviyeli Programlama Dilleri (insana en yakın) VISUAL BASIC, Access....(Dekleratif Diller)
- Yüksek Seviyeli Programlama Dilleri (PASCAL, COBOL)
- Orta Seviyeli Programlama Dilleri (C, ADA)
- Alçak Seviyeli Programlama Dilleri (Sembolik Makine Dilleri)
- Makine Dilleri (Bilgisayara en yakın)

Dillerdeki seviye yükseldikçe programcının işi daha kolay hale gelirken genel olarak esneklik ve verimlilik azalmaktadır.

Uygulama Alanlarına Göre

- Sayısal Uygulamalar için programlama dilleri
- Ticari Uygulamalar için programlama dilleri
- Yapay Zeka Uygulamaları için programlama dilleri
- Sistem programlama için programlama dilleri

Sayısal Uygulamalara Yönelik Programlama Dilleri

- Bilgisayarların ilk olarak kullanıldıkları alan sayısal uygulamaların ağırlıklı olduğu bilimsel çalışmalar olmuştur. Bu nedenle ilk geliştirilen programlama dilleri, sayısal programlama özelliklerini vurgulamışlardır.

FORTRAN

ALGOL

PL/I

BASIC

APL

SIMULA 67

PASCAL

C

ADA

Ticari Uygulamalara Yönelik Programlama Dilleri

- Ticari uygulamalardaki veri işleme, sayısal hesaplamalardan sonra ilk olarak gelişen uygulama alanıdır.
- **COBOL** (COmmon Bussiness Oriented Language) :
- A.B.D. Savunma Bakanlığı'nın, birçok şirketle birlikte İngilizce'ye yakın ve ticari uygulamalara yönelik bir dilin geliştirilmesi çalışmalarını desteklemesi sonucu, 1959 yılında COBOL tanıtılmıştır.
- COBOL dili, özellikle yoğun miktarda veri işleme kolaylıkları sağlayan deyimleri ve yapıları nedeniyle ticari uygulamalar alanında yazılım geliştirmek için popüler olmuştur.
- COBOL, hiyerarşik veri yapıları gibi birçok yeni kavram içeren ve özellikle raporlama açısından çeşitli olanaklar sağlayan bir dildir. 1961 ve 1962'de yenilenen dil, 1968'de standartlaştırılmış ve 1984'de tekrar yenilenmiştir.

Yapay Zeka Uygulamaları İçin Programlama Dilleri

- **LISP :**
LISP 1950'li yılların sonunda, *liste işleme* amaçlı fonksiyonel bir dil olarak, John McCarthy tarafından IBM 704 bilgisayarları için geliştirilmiştir. LISP, diferansiyel ve integral hesaplamalarında, sayısal mantık ve yapay zekanın diğer alanlarında sembolik hesaplamalar için kullanılmıştır. Sonraki yıllarda, birçok kez yenilenen LISP'ten başka, Scheme (1975) ve ML (1988) de LISP'i izleyen yapay zeka alanındaki fonksiyonel dillere örnektir.
- **PROLOG:**Prolog, temel denetim yapısı sembolik mantık kavramlarına dayanan özel amaçlı bir dildir. 1972 yılında tanıtılmış olan Prolog'un temel uygulama alanı, doğal dil işlemedir. Günümüze kadar Prolog, veritabanlarından uzman sistemlere kadar çeşitli uygulama alanlarında kullanılmıştır.

GİRİŞ/ÇIKIŞ KOLAYLIĞI

- Sıralı, indexli ve rastgele dosyalara erişme, veritabanı kayıtlarını geri alma, güncelleştirme ve sorgulama yeteneği olarak tanımlanabilir.
- C dili bu bakımdan zayıftır.
- Genel olarak veritabanı programları bu bakımdan güçlüdür.

VERİMLİLİK

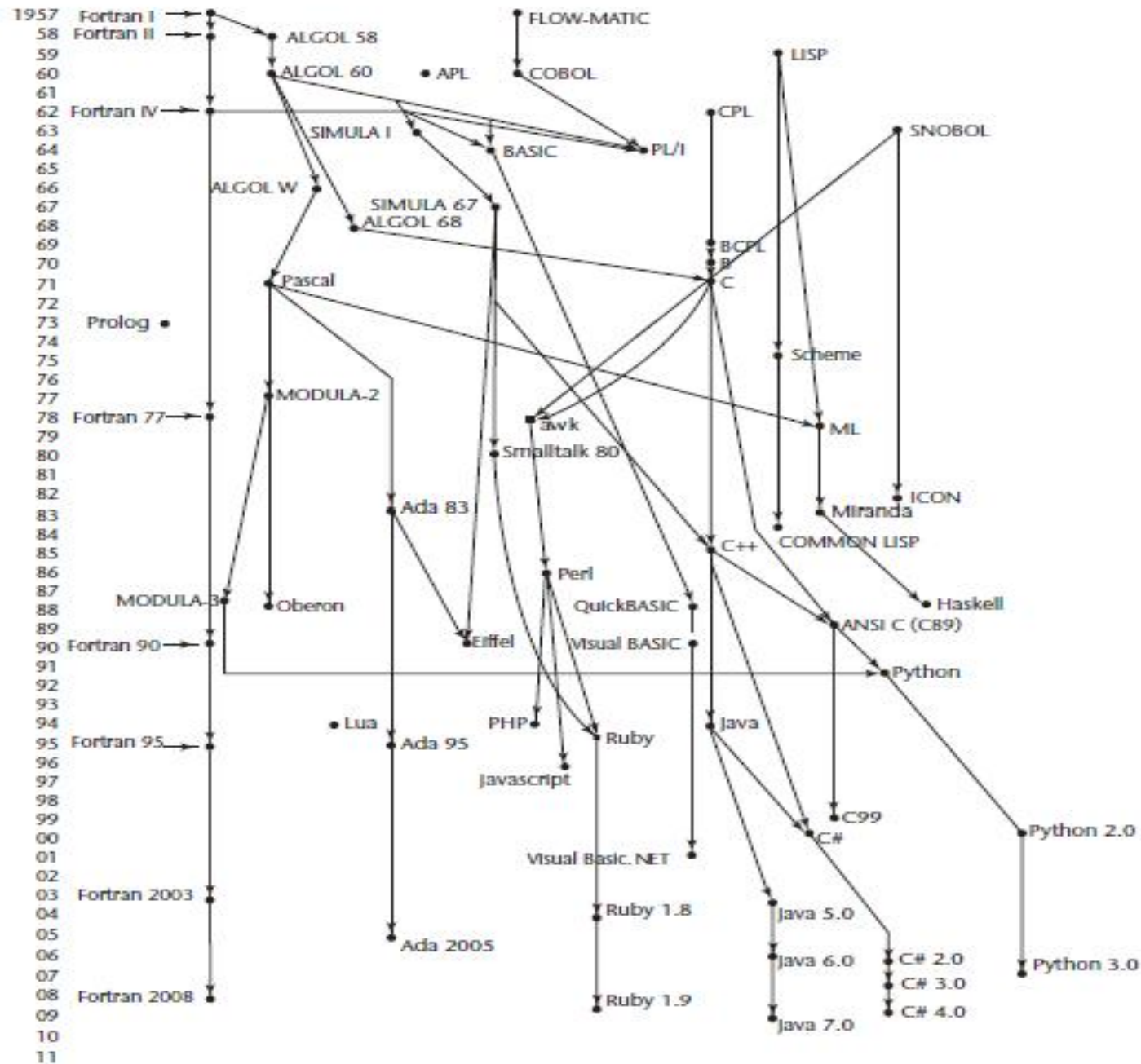
- Bir dilde yazıldıktan sonra amaç koda dönüştürülmüş programların hızlı çalışabilmesine verimlilik denir.
- C programları hızlı çalışır ve az yer kaplar.
- Çalışabilir kodun küçüklüğü ile çalışma hızı arasında doğrusal bir ilişki vardır.

YAPISALLIK

Burada bloklar halinde yazım ön plandadır.

- Yoğun olarak altprogram kullanılmaktadır.
- Program akışında atlamaların yapılması okumayı ve algılamayı zorlaştırabilir.
- Altprogramlar sayesinde soyutlama söz konusudur.

Evolution of the Major Programming Languages



Fortran (devam)

- Fortran ingilizce sözcüklerden oluşuyordu
- Anlaşılması makine diline göre çok kolaydı
- Matematiksel uygulamalar için iyi özelliklere sahipti.
- Fakat;
- Fortran dilinin bilgisayar tarafından anlaşılması için DERLEYİCİ'ye ihtiyacı vardı.
- John Backus, 2 yılda, 51000 satırlık makine kodundan oluşan ve bir teyp biriminde saklanan Fortran derleyicisini üretmiştir.
- FORTRAN 1957 yılında ticari olarak kullanıma sunuldu
- Aynı kişi 1959 yılında, yüksek seviyeli bir dilin yazış kurallarını açıklama noktasında standart bir notasyon haline gelen **B**ackus **N**aur **F**orm (BNF)'yi bulmuştur.

FORTTRAN ailesi

- FORTRAN I:Taşınabilirliği çok kötü
- FORTRAN 66:Karakter türü verileri işlemede çok kısıtlı, yapısal programlama yok, rekürsif işlemler yapılamıyor.
- FORTRAN 77(American National Standarts Institute-ANSI): Karakter türü verileri işleyebiliyor, Yapısal programlamayı destekliyor, DO çevrimine dışarıdan veri girilebiliyor. Taşınabilirlik daha iyi hale getiriyor.
- FORTRAN 90:Pointer (bağlı liste gerçeklemesi, dinamik bellek yönetimi), rekürsif işlem yeteneği, bit düzeyinde işlem, Dizi yapıları daha iyi kullanılabiliyor, Altprogram yapıları daha esnek, isimler daha uzun yazılabiliyor (okunabilirlik)

LISP (devam)

- LISP esas olarak yorumlayıcı kullanan bir dildir. Derleyici kullanan versiyonları da vardır.
- Veri tiplmesi bakımından esnek bir dildir.
- Olağanüstü esneklik, ifade gücü ve **kodun aynı zamanda veri olarak da kullanılabilmesi** özelliği LISP'i yapay zaka uygulamalarında rakipsiz hale getirmiştir.
- Nesne Yönelimli Programlamayı destekler
- Veri tabanlarına erişim ve GUI olanağı vardır.
- Çok iyi belgelendirilmiş olup, COMMON LISP, ANSI tarafından standart hale getirilmiştir.

ALGOL

- **ALGO**rithmic **L**anguage
- FORTRAN I'den esinlenilmiştir
- İlk kez 1958 yılında Avrupalı ve Amerikalı bir komisyonun Zürih'teki çalışmaları sonucu oluşturulan yüksek seviyeli bir dildir. İlk isim ALGOL 58 olarak verilmiştir.

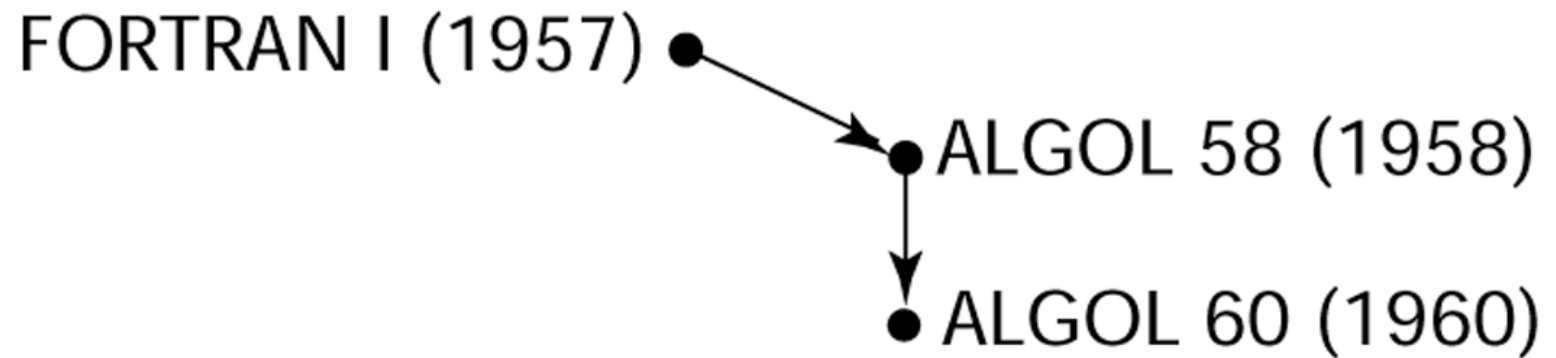
ALGOL 60

- GAMM(German Society for Applied Mathematics and Mechanics) tarafından ALGOL'58'e eklemeler yapılmıştır.
- Amaçlar:
 - Matematiksel notasyonlara yakın ve kolay okunabilen
 - Literatürdeki hesap süreçlerinde kullanılabilen
 - Makine diline kolaylıkla çevrilebilen bir dil olsun
- Makineden bağımsız, daha esnek ve daha güçlübir dil olmuştur.
- Atama ifadesi olarak ilk evrensel kullanım bu dilde olmuştur.
variable := expression

ALGOL 60: Eksik yönler

- Anlaşılır olmakta zorluk ve yürütmede (implementation) verimsizliğe götürecek kadar esnek
- I/O ifadelerindeki yetersizlik yada zayıflıklar (eksiklik)
- 1960'lı yıllar için BNF(Backus Naur Form) kullanımı karmaşıklıkmiş gibi gözükmemektedir.
- Avrupalı bilim adamları arasında sonderece popüler, eğitim ve araştırma aracı olarak kullanılır ve bilimsel makalelerde algoritmaları açıklamak için kullanılmasına rağmen;
- IBM tarafından desteklenmemiştir.
- Çünkü bu sırada IBM FORTRAN dilinde yazılmış zengin bir kütüphaneye sahiptir ve haliyle FORTRAN'ı desteklemektedir.
- O zamanlar IBM'in bilişim sektörünün %80'ine sahip olduğu düşünülürse, bu durum ALGOL60 için bir dezavantajdır.

ALGOL 60'ın şeceresi

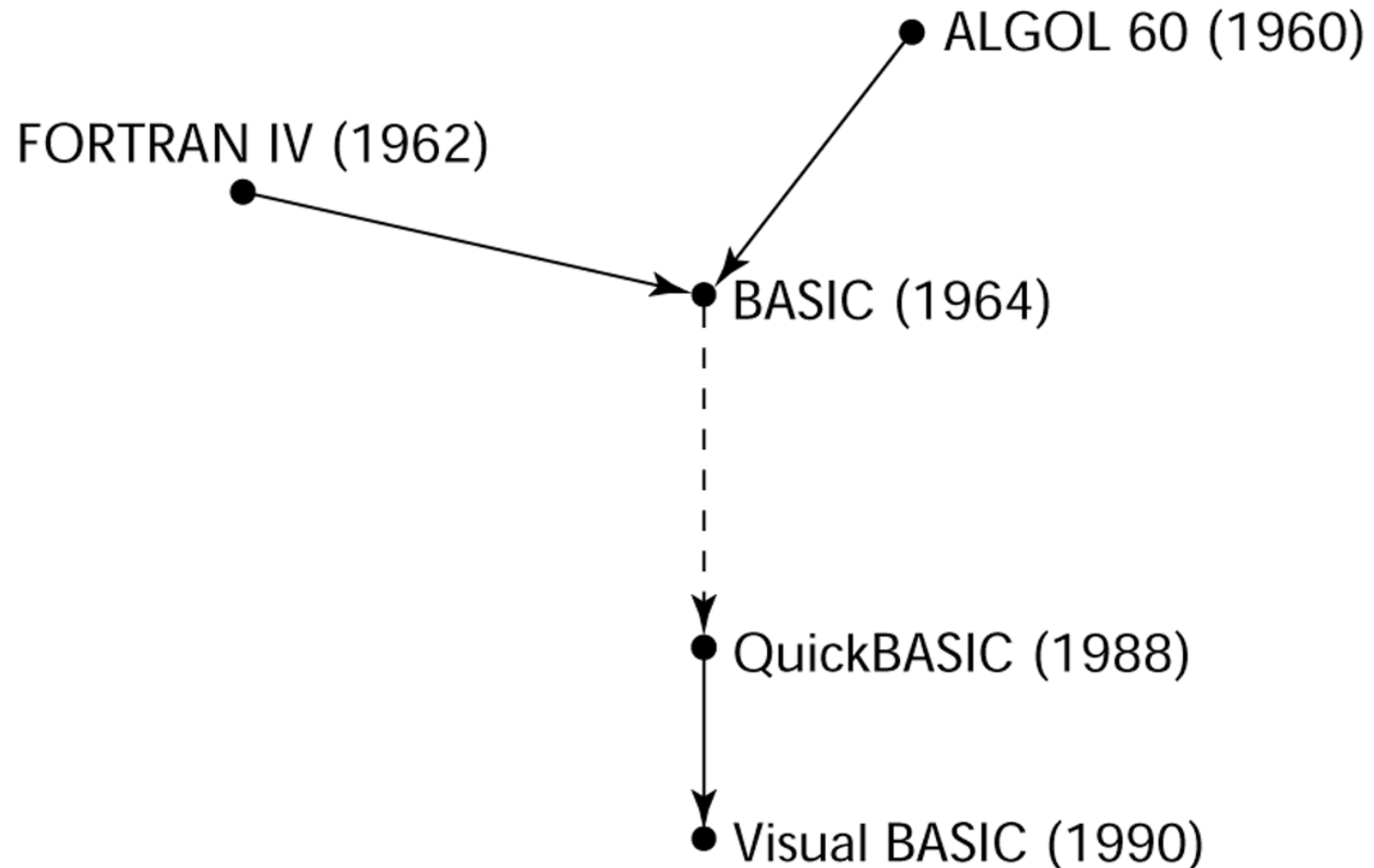


BASIC: İlk Zaman Paylaşımlı Bilgisayar Sistemi (1964)

- Beginner's All-purpose Symbolic Instruction Code
- Öğrencilerin bilgisayara daha kolay erişimlerini sağlamak ve basit ve etkin bir programlama dili ile program yazabilme isteklerine cevap vermek için tasarlanmış bir dildir.
- The first PL used through terminals connecting to a remote computer
- Sadece 14 komuta (LET, PRINT, GOTO...) sahipti. Tek veri tipi (number= kayan noktalı ve tamsayı)
- BASIC, FORTRAN ve ALGOL'den bazı bileşenleri almıştır. FORTRAN'dan DO çevrimini, ALGOL'den ise "until" yerine "to"

- Kolay bir dil ve genel maksatlı, belirli bir alana bağlı değil
- Uzman kişilere de hitap edebiliyor
- Açık ve anlaşılır hata mesajlarına sahip, kullanıcı bilgisayarla etkileşimli çalışabiliyor
- Küçük boyutlu programları hızlı bir biçimde çalıştırabiliyor
- Kullanım için donanım bilgisine sahip olmaya gerek yok
- Kullanıcıyı işletim sistemi ayrıntılarından dahi koruyabiliyor
- Derleyici kullanıyor, programın tümü makine diline çevrildikten sonra icra ediliyor
- BASIC'in pekçok versiyonları olmuştur. 1989'da ise nesne yönelimli uyarılama olan Visual BASIC ve 1998'de VB6.0 sunulmuştur.

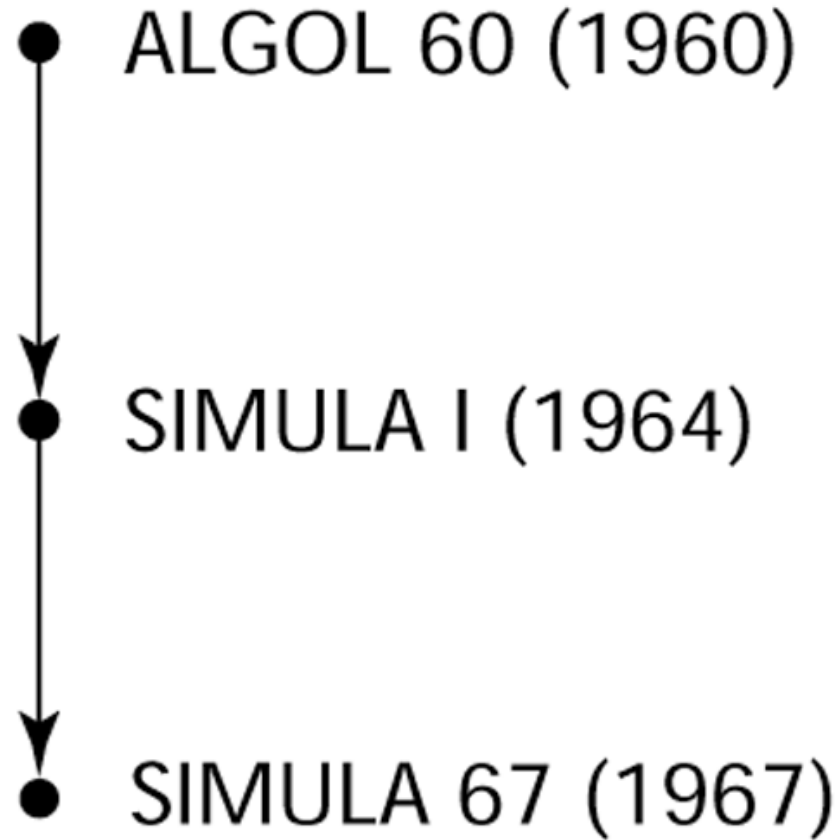
BASIC'in Şeceresi



SIMULA 67: Beginnings of Data Abstraction (1962 ve 1964 arasında)

- İlk olarak simülasyon için tasarlanmış bir dildir.
- ALGOL60'ın genişletilmiş versiyonudur. (Blok yapısı ve kontrol ifadeleri buradan alınmıştır)
- Daha önce durdurulduğu yerden itibaren yeniden çalışmaya başlayan altprogramları desteklemektedir.
- Veri soyutlamasına imkan veren sınıf yapılarını desteklemektedir

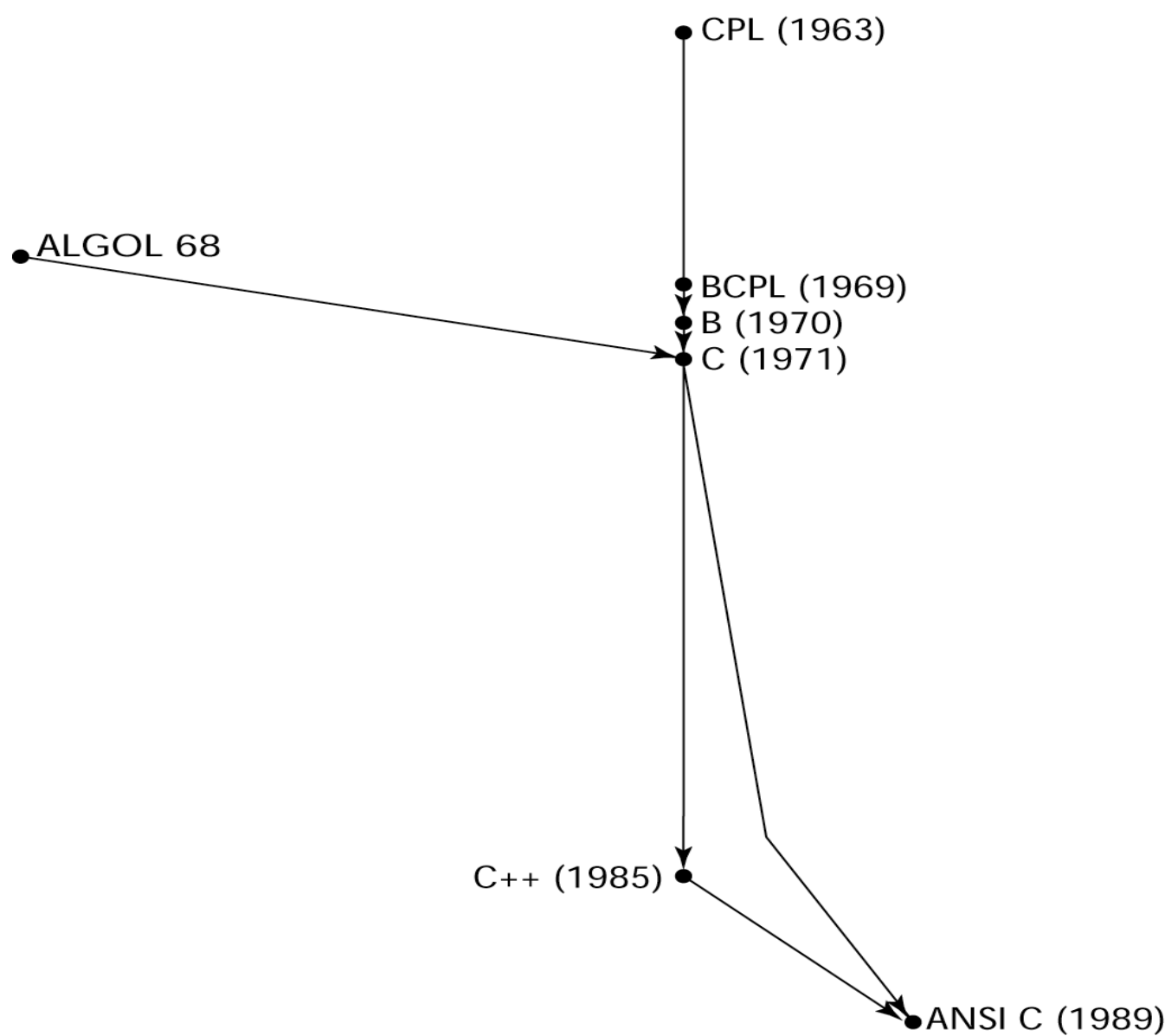
SIMULA 67'in Şeceresi



ALGOL'ün Torunları

- **Pascal (1971):**
 - İlk önceleri eğitim dili olarak kullanıldı
 - Basit ve okunabilir, yazılabilir (expressive)
 - Fortran and C ile karşılaştırıldığında emniyetli bir dildir.
- **C: Taşınabilir Sistem Programlama Dili (1972)**
 - Yeterli derecede kontrol ifadelerine ve veri yapısı olanaklarına sahip.
 - Tip kontrolü yetersiz.
 - Derleyicilerine çok kolay bir biçimde ulaşılabilir.

C'nin Şeceresi



ADA:

Tarihin en geniş tasarım çalışması

- ABD savunma bakanlığının bir çalışması sonucu ortaya çıkmıştır. Gömülü sistemlerin (embedded systems) programlanmasını sağlayacak PL üretimi amaçlanmıştır. (1983). Gerçek zamanlı uygulamalar için
- Augusta Ada Byron'ın (1815-1851) ölümünden sonra isimlendirilmiştir.
- Blok yapılı, nsne yönelimli, genel amaçlı ve eşzamanlılığı destekleyen bir dildir.
- Büyük boyutlu yazılımlar için uygundur.

ADA (devam)

- Şablon (Generic) program birimleri sayesinde yazılımın yeniden kullanılabilmesini Buluşma yeri (rendezvous) mekanizmasının ilavesiyle eşzamanlı çalışmayı desteklemektedir. mechanism
- Çok geniş ve çok karmaşık bir dil. (Özellikle yazım kuralları)
- Çeşitli durumlara uygulanabilecek hazır şablonlara (template) sahiptir.
- İlk sıralar ADA derleyicileri kod üretmekte verimsiz idi.
- En çok gömülü sistemlerde başarılı olmuştur.
- Veri tipleri konusunda çok zengindir. Çok-iş işleme özelliğine sahiptir.

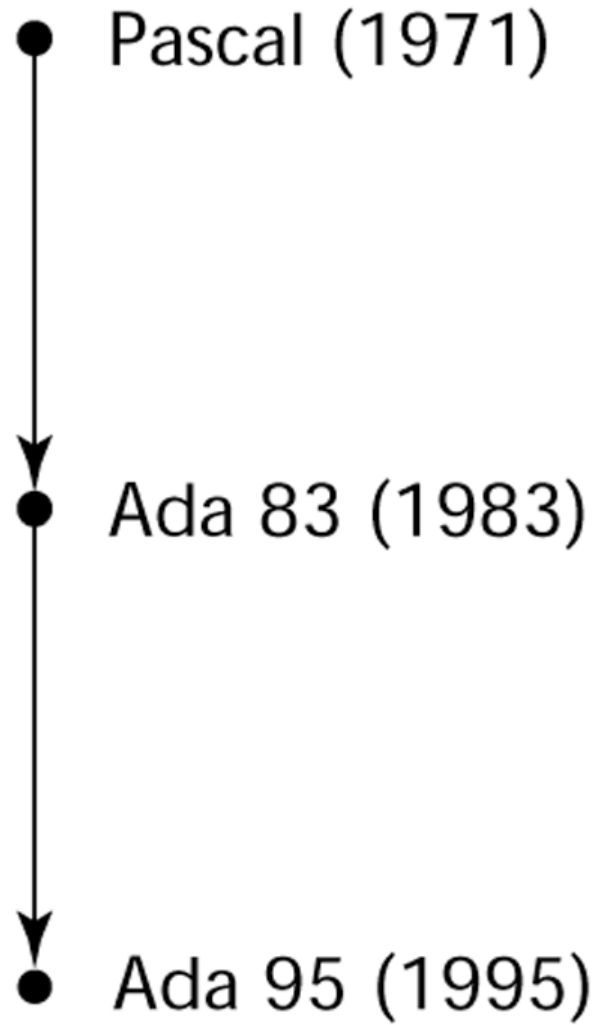
Ada 95

- ADA83'e kalıtım (inheritance), çok biçimlilik (polymorphism) gibi yeni eklemeler yapılarak ve tip türetim mekanizmasını genişleterek Ada 95 elde edilmiştir.
- Altprogramların dinamik kapsam bağlama kurallarına göre çağırılması mekanizması da ADA 95'in özelliklerine katılmıştır.

Ada 95

- ADA83'e kalıtım (inheritance), çok biçimlilik (polymorphism) gibi yeni eklemeler yapılarak ve tip türetim mekanizmasını genişleterek Ada 95 elde edilmiştir.
- Altprogramların dinamik kapsam bağlama kurallarına göre çağırılması mekanizması da ADA 95'in özelliklerine katılmıştır.

ADA'nın şeceresi



- Bir dilin sözdizimini anlatmak amacıyla kullanılan bir araç vardır. BNF (Backus-Naur Form) **metadili** böyle bir araçtır.
- Anlam tanımlama için böyle bir dil yoktur.

character stream

v a l = 1 0 * v a l + i



lexical analysis (scanning)



token stream

1	3	2	4	1	5	1
(ident)	(assign)	(number)	(times)	(ident)	(plus)	(ident)
"val"	-	10	-	"val"	-	"i"

token number

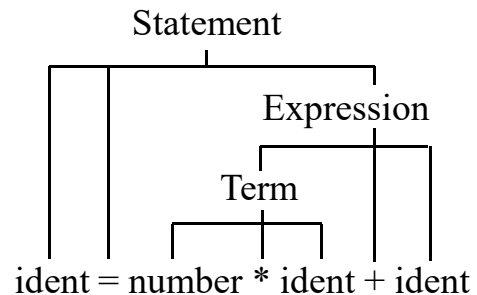
token value



syntax analysis (parsing)



syntax tree



Grammar Nedir?

Example

Statement = "if" "(" Condition ")" Statement ["else" Statement].

4 bileşenden oluşur

terminal symbols	atomik	"if", ">=", ident, number, ...
nonterminal symbols	Sözdizim değişkenleri	Statement, Expr, Type, ...
productions	Nonterminallerin çözümü	Statement = Designator "=" Expr ";" Designator = ident ["." ident]. ...
start symbol	Başlangıç nonterminali	begin

Backus-Naur Form (BNF)-CFG

– CFG

$$\begin{aligned} \text{expression} \rightarrow & \text{identifier} \mid \text{number} \mid - \text{expression} \\ & \mid (\text{expression}) \\ & \mid \text{expression operator expression} \end{aligned}$$
$$\text{operator} \rightarrow + \mid - \mid * \mid /$$

– BNF

$$\begin{aligned} \langle \text{expression} \rangle \rightarrow & \langle \text{identifier} \rangle \mid \langle \text{number} \rangle \mid - \langle \text{expression} \rangle \\ & \mid (\langle \text{expression} \rangle) \\ & \mid \langle \text{expression} \rangle \langle \text{operator} \rangle \langle \text{expression} \rangle \end{aligned}$$
$$\langle \text{operator} \rangle \rightarrow + \mid - \mid * \mid /$$

EBNF Notation

Extended Backus-Naur form

John Backus: developed the first Fortran compiler
Peter Naur: edited the Algol60 report

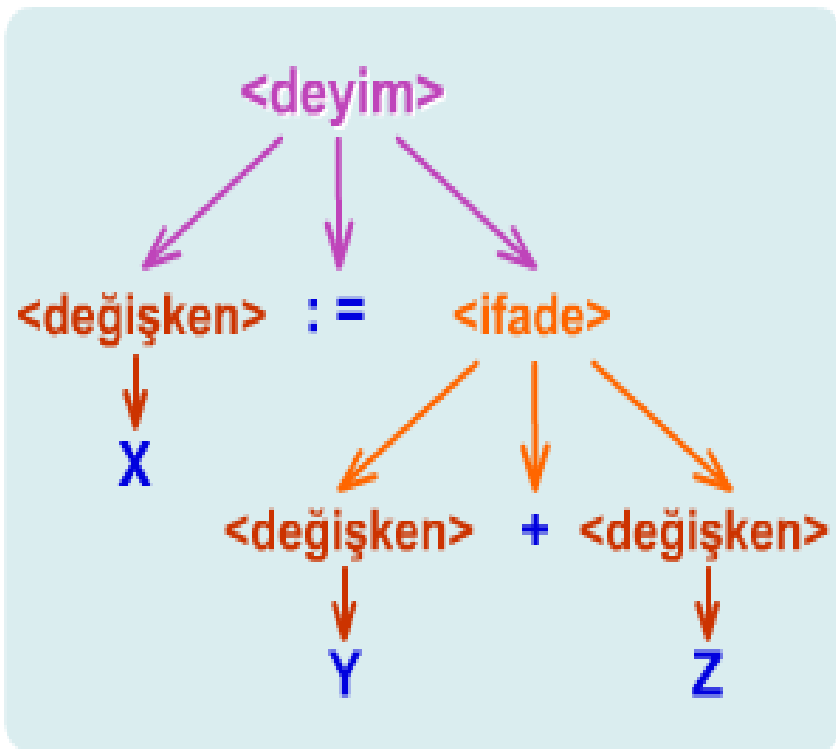
<i>symbol</i>	<i>meaning</i>	<i>examples</i>
string		"=", "while"
name		ident, Statement
=		A = b c d .
.		
	separates alternatives	a b c ⌚ a or b or c
(...)	groups alternatives	a (b c) ⌚ ab ac
[...]	optional part	[a] b ⌚ ab b
{...}	repetitive part	{ a } b ⌚ b ab aab aaab ...

EBNF	<code><seçimlik_deyim> -> lf (<mantıksal>) <deyim> [else <deyim>];</code>
BNF	<code><seçimlik_deyim> -> lf (<mantıksal>) <deyim></code>
	<code><seçimlik_deyim> -> lf (<mantıksal>) <deyim> else <deyim>;</code>

Değiştirme (*alternation*) |

EBNF	<code><for_deyimi>->for<değişken> := <ifade> (to down to) <ifade> do <deyim></code>
BNF	<code><for_deyimi>->for<değişken> := <ifade> (to) <ifade> do <deyim></code>
	<code><for_deyimi>->for<değişken> := <ifade> (down to) <ifade> do <deyim></code>

Grammerler ve Türetimler



`<program> -> begin <deyim_listesi> end`

`<deyim_listesi> -> <deyim>
|<deyim>;<deyim_listesi>`

`<deyim>-> <değişken> :=<ifade>`

`<ifade> -> <değişken> + <değişken>
|<değişken>`

`<değişken> -> X | Y | Z`

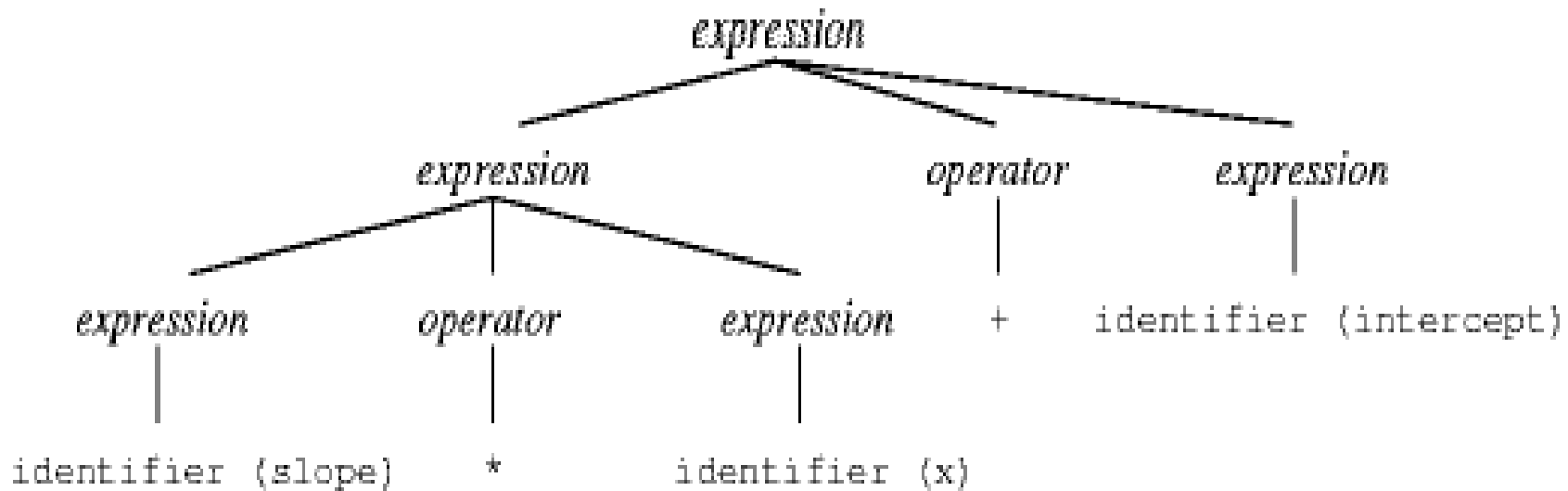
Örnek

- $expression \rightarrow identifier \mid number \mid - expression$
 $\mid (expression)$
 $\mid expression operator expression$
- $operator \rightarrow + \mid - \mid * \mid /$

slope * x + intercept ifadesini
türetelim.

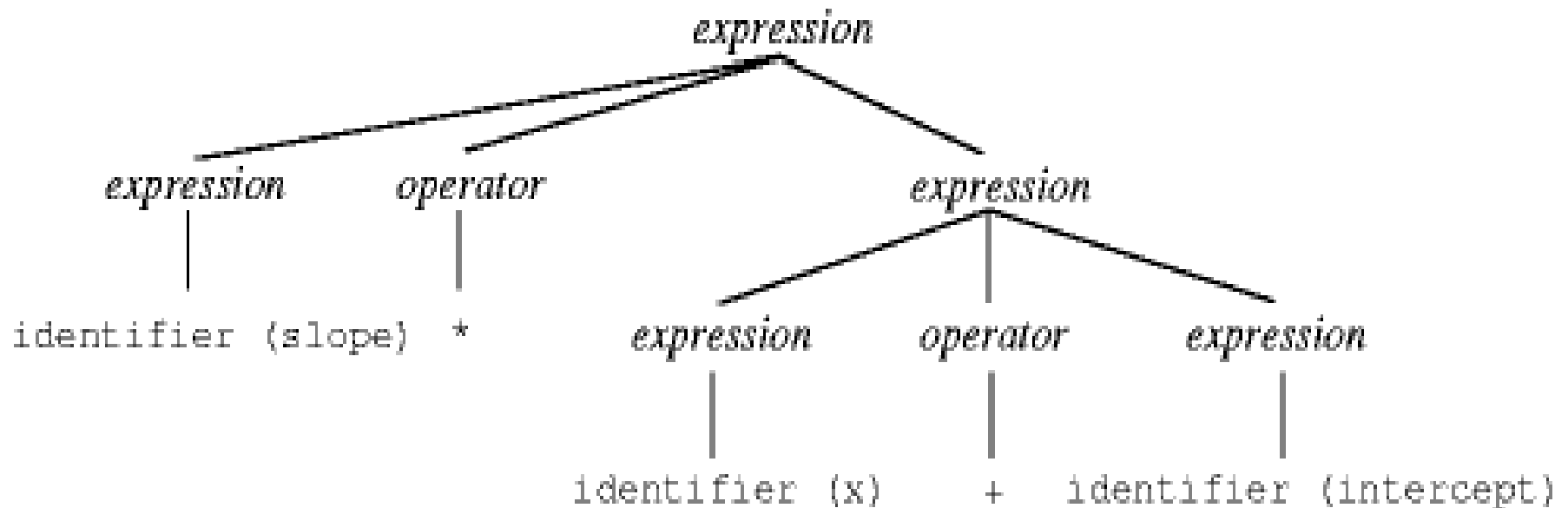
Ayrıştırma Ağacı(Parse Trees)

- Türetimin grafik gösterimi



Belirsiz Gramer

- Bir gramerde aynı ifade için alternatif ayrıştırma ağacı bulunuyorsa bu gramer belirsizdir (ambiguous)



Belirsizlik kaldırılmalıdır.

$$10 - 4 - 3 ===== (10 - 4) - 3$$

$$3 + 4 * 5 ===== 3 + (4 * 5)$$

$$(1) \text{ expression } \longrightarrow \text{ term } \mid \text{ expression add_op term}$$

$$(2) \text{ term } \longrightarrow \text{ factor } \mid \text{ term mult_op factor}$$

$$(3) \text{ factor } \longrightarrow \text{ identifier } \mid \text{ number } \mid - \text{ factor } \mid (\text{ expression })$$

$$(4) \text{ add_op } \longrightarrow + \mid -$$

$$(5) \text{ mult_op } \longrightarrow * \mid /$$

DİL ÇEVİRİMİ

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

Sözdizim Analizi (Syntax Analysis)

- EBNF

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

Sözdizim Analizi (Syntax Analysis)

- EBNF

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

Özel Kelimeler

- Özel kelimeler, bir programlama dilindeki temel yapılar tarafından kullanılan kelimeleri göstermektedir.
- A)Anahtar kelimeler(Keywords)
- B)Ayrılmış kelimeler(Reserved words)

Anahtar Kelime

- Bir anahtar kelime (*keyword*), bir programlama dilinin sadece belirli içeriklerde özel anlam taşıyan kelimelerini göstermektedir.
- Örneğin FORTRAN'da *REAL* kelimesi, bir deyimin başında yer alıp, bir isim tarafından izlenirse, o deyimin tanımlama deyimi olduğunu gösterir. (*REAL apple*)
- Eğer *REAL* kelimesi, atama işlemcisi "=" tarafından izlenirse, bir değişken ismi olarak görülür. *REAL = 10.05* gibi.
- Bu durum dilin okunabilirliğini azaltır.

Ayrılmış Kelime:

- Öte yandan, ayrılmış kelime (*reserved word*), bir programlama dilinde bir isim olarak kullanılamayacak özel kelimeleri göstermektedir.
- C++ dilindeki do, for , while gibi ve
- PASCAL'da procedure, begin, end gibi kelimere ayrılmış kelime denir.

Bağlama ve Kapsam Kavramları

Bağlama(Binding)

- Bir özellekle bir program elemanı arasında ilişki kurulmasına **bağlama** (*binding*) denir.



Bağlama Zamanı

- Bir programlama dilinde çeşitli bağlamalar farklı zamanlarda gerçekleşebilir.



int hesap; ... hesap=hesap+10;	
Hesap için olası tipler	Dilin tasarım zamanında
Hesap değişkeninin tipi	Dilin derlenmesi zamanında
Hesap değişkeninin olası değerleri	Derleyici tasarım zamanı
Hesabın değeri	Bu deyimın yürütülmesi zamanında
+ işlemcisinin muhtemel anlamları	Dilin tanımlanması zamanında
+ işlemcisinin bu deyimdeki anlamı	Derlenme süreci
10 literalinin ara gösterimi	Derleyici tasarımı zamanında
Hesap değişkeninin alacağı son değer	Çalışma zamanında

Bellek Bağlama

- (*allocation*) (*deallocation*) lifetime



etkinlik (*activation*) kaydı

aynı bellek bölümünün
yeniden kullanılabilmesi

Doğrudan adresleme

Pascal-*dispose* Java-otomatik
C'deki malloc fonksiyonu
C++ 'daki new işlemcisi

Statik değişkenler, programın yürütülmesi başlamadan bellek hücrelerine bağlanırlar ve bellek hücreleri ile programın çalışması sonlanıncaya kadar bağlı kalırlar. FORTRAN I, II ve FORTRAN IV'de hepsi statik. C, C++ ve Java **static** anahtarını kullanır.

ALGOL 60 ve bu çizgideki diller yığıt dinamik değişkenleri tanımlamaktadır. FORTRAN77 ve FORTRAN90 yerel olarak yığıt dinamik değişkenlere izin vermektedir. Pascal, C ve C++'da, lokal değişkenler, varsayılan olarak yığıt_dinamik değişkenlerdir.

Dışsal yığın dinamik değişkenlerin bellek yeri bağlaması çalışma zamanında gerçekleşir. Ne kadar bellek gerektiği önceden bilinmez. Çalışma zamanında veriler oldukça belleğe atanır ve bellek yeri yığın bellekten alınır ve daha sonra yığın belleğe iade edilir. Bu verilere sadece işaretçi (pointer) değişkenler aracılığıyla ulaşılabilir. Bu değişkenlerin tip bağlaması derleme zamanında, bellek yeri bağlaması ise çalışma zamanında gerçekleşir.

Statik isim kapsam	Dinamik Kapsam
Değişkenlerin kapsamları, programın metinsel düzenine göre, fiziksel yakınlığa göre, belirlenir.	Bir ismin kapsamının, altprogramların fiziksel yakınlıklarına göre değil, altprogramların çağırılma sırasına göre çalışma zamanında belirlenmesi dinamik kapsam bağlama olarak adlandırılır.
ALGOL 60'ı izleyen çok sayıda dilde tanımlıdır. Altprogramlar iç içe yuvalanabilir. (C++ ve FORTRAN hariç)	LISP, APL dillerinin ilk sürümleri
<ol style="list-style-type: none"> 1. Altprogramların yuvalanması sonucu gereğinden fazla genel değişken kullanımı olabilir. 2. Bir programda genel olarak tanımlanan değişkenler tüm altprogramlara görünebilir olacakları için güvenilirlik azalmaktadır. 	<ol style="list-style-type: none"> 1. Bir altprogramda bir değişkene yapılan başvuru, deyimin her çalışmasında farklı değişkenleri gösterebilir. 2. Programların anlaşılabilirliğini azaltmaktadır

Örnek: Aşağıdaki program parçasının çıkışını

- a) statik kapsam bağlama kurallarına göre
- b) Dinamik kapsam bağlama kurallarına göre bulunuz.

```
int x;  
  
int main() {  
    x = 2;  
    f();  
    g();  
}  
  
void f() {  
    int x = 3;  
    h();  
}  
  
void g() {  
    int x = 4;  
    h();  
}  
  
void h() {  
    printf("%d\\n",x);  
}
```

Dinamik kapsam bağlamaya göre çıkış: 3 4

Statik kapsam bağlamaya göre çıkış: 2 2

Ada (exit deyimi)

```
loop
```

```
....
```

```
exit when koşul;
```


```
....
```

```
end loop;
```



QuickBASIC'te *exit* deyimi

```
for j=1 to 5  
  input miktar  
  if miktar < 0 then exit for  
  toplam = toplam + miktar  
end
```



Neden Altprogram

- Program kodlarının gereksiz yere uzaması önlenir.
- Programın tasarlanmasını kolaylaştırır ve okunulabilirliğini artırır.
- Büyük bir programın, daha küçük ve yazılması daha kolay fonksiyonel parçalara bölünür.
- Ekip çalışması
- Altprogramlar birden çok uygulama arasında paylaşılabilir.

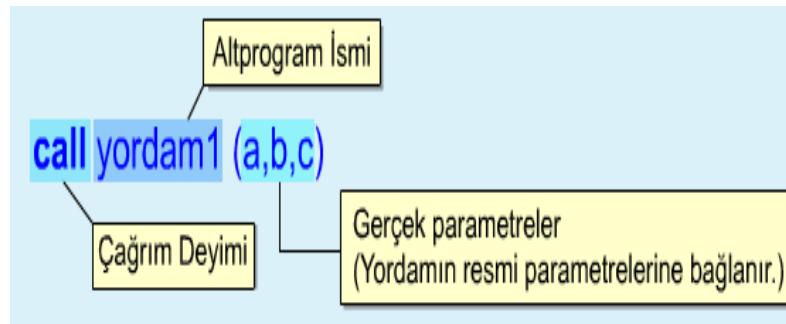
- Bir altprogram bir fonksiyon ise, parametre profiline ek olarak fonksiyonun sonuç değerinin dönüş tipini de içerir ve bu bilgiye **altprogram protokolü** adı verilir.
- C programlama dilinde, altprogramların tanımlandığı deyimler yer alır. Bu deyimlerde altprogramların parametre tipleri belirtilir ve bu deyimlere **prototip** (*prototype*) adı verilir.
- Bir altprogram, bir **yordam** veya bir **fonksiyon** olabilir
- İki altprogram türü arasındaki fark, fonksiyonların çağıran program birimine bir sonuç değeri döndürmelerinin şart olmasıdır.

procedure ortalama (parametreler);

ortalama (parametreler);

Parametreler

- **gerçek parametrelerin** ve **resmi parametrelerin** ilişkilendirilmesi için kullanılan iki yöntem:
- **konumsal** ve **anahtar kelime parametre** yöntemleridir.
- çağırım deyiminde, altprogramın ismine ek olarak, yordamın resmi parametreleriyle eşleştirilecek **gerçek (actual) parametreler** de yer alır.
-



Konumsal Parametreler

- Eğer bir yordam çağrımında gerçek parametreler ile resmi parametreler arasındaki bağlama, parametrelerin çağrım deyimindeki ve yordam başlığındaki konumuna göre yapılıyorsa, bu parametrelere **konumsal** (*positional*) **parametre** adı verilir. Birçok programlama dilinde uygulanan bu yöntem, parametre sayısı az olduğu zaman kullanışlıdır.

- Aşağıda görüldüğü gibi bir çağrım deyimi ile etkin duruma geçen *ortalama* yordamında, *c* resmi parametresi, *a* gerçek parametresi ile; *d* resmi parametresi, *b* gerçek parametresi ile bağlanıyorsa, konumsal parametreler yaklaşımı uygulanmıştır.

```
call ortalama(a,b);
```

```
Procedure ortalama(c: integer; d:integer);
```

```
begin
```

```
    ...
```

```
end;
```

Resmi Parametre	Gerçek Parametre
c	a
d	b

Anahtar Kelime Parametre Yöntemi

- Gerçek ve resmi parametreler arasındaki bağlamayı konuma göre belirlemek yerine, her iki parametrenin de ismini belirterek gösterme, **anahtar kelime parametre** yöntemi olarak adlandırılır. Bu durumda çağırım deyiminde, hem resmi, hem de gerçek parametrelerin isimleri belirtilir.

Call (toplam→alttoplam, liste→dizi, gelir→kazanc);

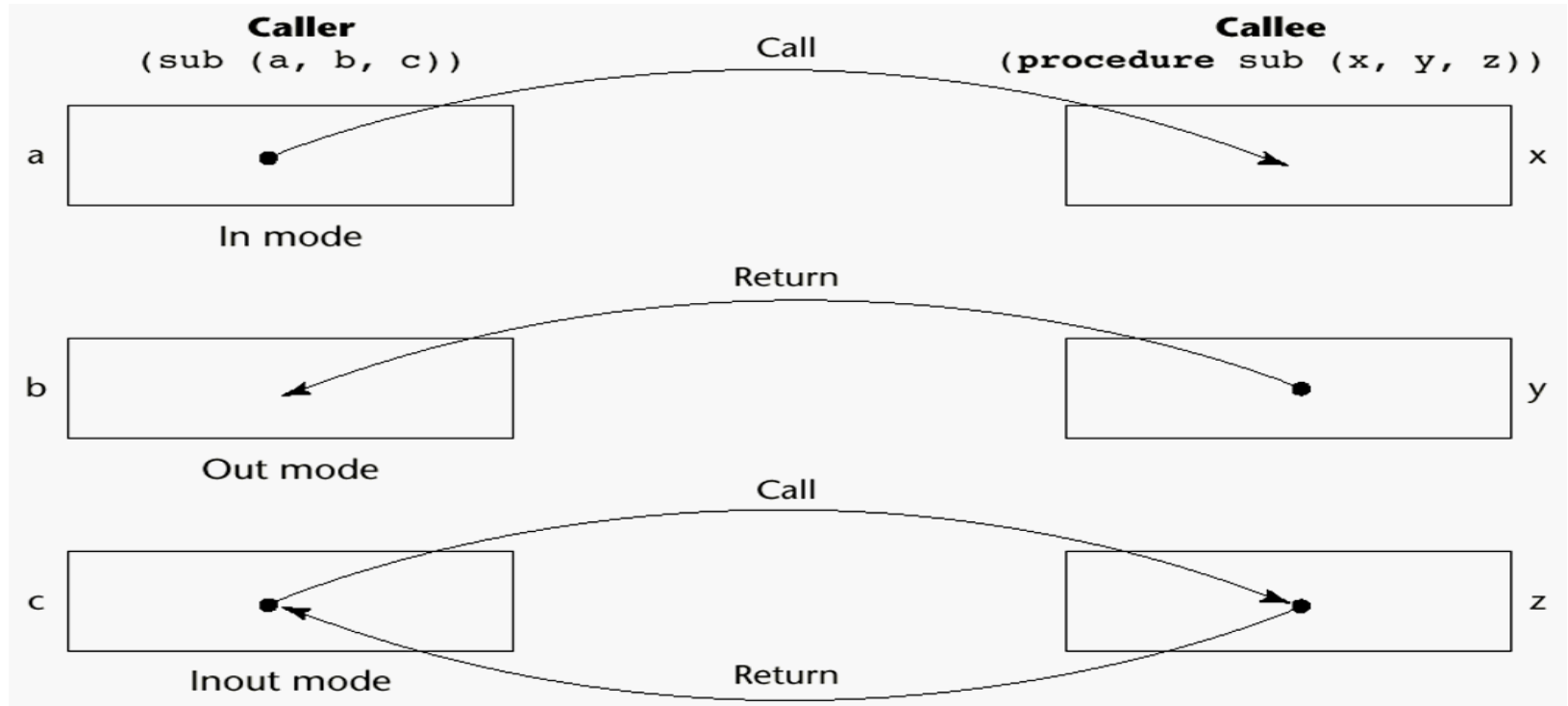
Anahtar kelime parametre yöntemi, parametre sayısının çok olduğu durumda yararlı bir yöntemdir.

Ada ve FORTRAN 90'da hem anahtar kelime hem de konumsal parametreler yöntemi kullanılabilmektedir.

Parametre Aktarım Yöntemleri

- Bir yordam etkin duruma getirilirken, çağrım deyimindeki gerçek parametreler ile yordamın resmi parametrelerine farklı değerler aktarılabilir.

Gerçek parametreler yordamlara aktarılacak değerleri gösterdikleri için, değişken, sabit veya ifade olabilir. Resmi parametreler ise bu değerleri tutacak bellek yerlerini gösterdikleri için değişken olmak durumundadır.



Parametre Aktarım Yöntemleri (Devam)

- Değer ile çağırma, sonuç ile çağırma, değer ve sonuç ile çağırma yöntemlerinde, gerçek ve formal parametreler arasındaki veri fiziksel olarak kopyalanarak aktarılmakta, başvuru ile çağırma yönteminde ise veri yerine verinin adresi aktarılmaktadır.

Değer ile Çağırma (*Call by Value*)

- Bu yöntemde formal parametre, gerçek parametrenin değeriyle iklendikten sonra, altprograma yerel bir değişken olarak değerlendirilir.

```
void f(int n)
{  n++;}
int main() {
    int x = 2;
    f(x);
    cout << x; }
```

Sadece gerçek parametreden formal parametreye değer geçişi olduğu için en güvenilir parametre aktarım yöntemidir

Programın çıktısı= 2 olacaktır.

Sonuç ile Çağırma (*Call by Result*)

- Bu yöntemde çağırım deyimi ile altprograma bir değer aktarılmazken, gerçek bir parametreye karşı gelen formal parametrenin değeri, altprogram sonunda, denetim yeniden çağıran programa geçmeden önce, gerçek parametreyi gösteren değişkene aktarılır. (gerçek parametrenin değişken olmalı). Gerçek parametreye karşı gelen resmi parametre, altprogramın çalışması süresince yerel değişkendir.

procedure sub1(y: out Integer; z: out Integer) is

...;

sub1(x, x);

Değer ve Sonuç ile Çağırma (*Call by Value Result*)

- Değer ile çağırma ve sonuç ile çağırma yöntemlerinin birleşimidir.
- Gerçek parametrenin değeri ile karşı gelen formal parametrenin değeri ilklenir ve sonra resmi parametre, altprogramın çalışması süresince yerel değişken gibi davranır ve altprogram sona erdiğinde formal parametrenin değeri gerçek parametreye aktarılır.

```
int x=0;  
int main()  
{  
    f(x);  
    .....  
}  
void f(int a) {  
    x=3;  
    a++;}
```

x'in son değeri değer-sonuç aktarımına göre 1 olacaktır.

Parametreler için birden çok bellek yeri gerekmesi ve değer kopyalama işlemlerinin zaman almaktadır.

Başvuru ile Çağırma (*Call by Reference*)

- Başvuru ile çağırma yöntemi de gerçek ve formal parametreler arasında iki yönlü veri aktarımı vardır.
- Altprograma verinin adresi aktarılır. Bu adres aracılığıyla altprogram, çağıran program ile aynı bellek yerine erişebilir ve gerçek parametre, çağıran program ve altprogram arasında ortak olarak kullanılır.

```
int x=0;  
int main()  
{  
    f(x);
```

```
.....  
}  
void f(int a) {  
    x=3;  
    a++;}
```

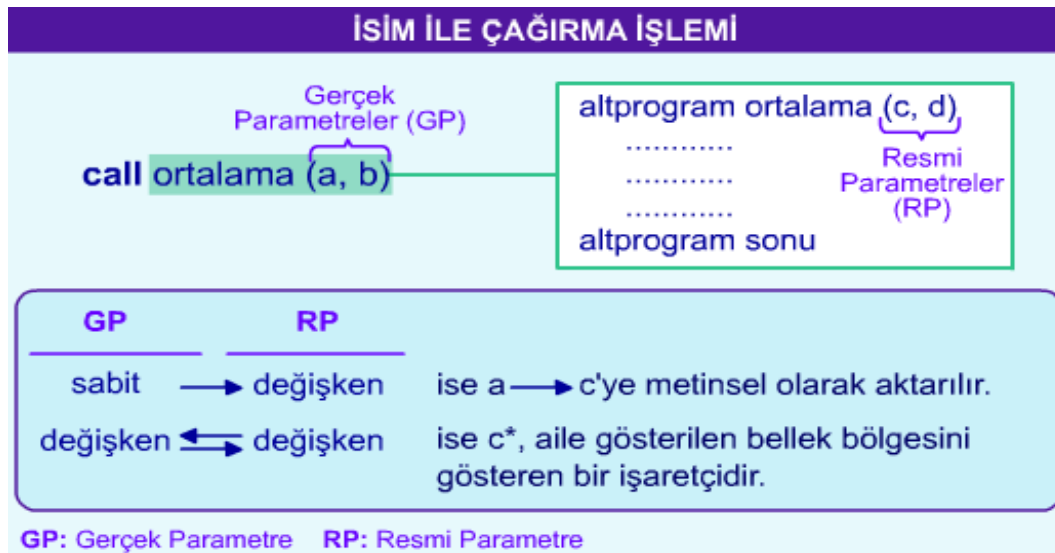
x'in son değeri değer-sonuç aktarımına göre 4 olacaktır.

Hem yer hem de zaman açısından etkindir. Fiziksel kopyalama yoktur.

Formal parametrelere erişim için bir **dolaylı erişim** gerekiyor. Sadece çağıran programdan altprograma değer aktarılması isteniyorsa, bu yöntemde gerçek parametrenin bellekteki yerine altprogram tarafından ulaşılabildiği için, değerinde **istenmeyen değişiklikler** oluşabilir.

İsim ile Çağırma (*Call by Name*)

- Altprogramda gerçek parametreye karşı gelen formal parametrenin bulunduğu her yere metinsel olarak gerçek parametre yerleştirilir.
- Eğer gerçek parametre bir sabit değerse, isim ile çağırma yöntemi, değer ile çağırma yöntemi ile aynı şekilde gerçekleşir.
- Eğer gerçek parametre bir değişkense, isim ile çağırma yöntemi başvuru ile çağırma yöntemi ile aynı şekilde gerçekleşir.
- İsim ile çağırma yönteminin gerçekleştirilmesi güçtür ve kullanıldığı programların hem yazılmasını hem de okunmasını karmaşıktırabilir. Bu nedenle ALGOL 60 ile tanıtılan isim ile çağırma yöntemi, günümüzde popüler olan programlama dillerinde uygulanmamaktadır.



Programlama Dili	Değer	Sonuç	Değer- Sonuç	Referans	İsim
FORTRAN IV				X	
Fortran 77			X		
ALGOL 60	seçimlik				X
ALGOL W			X		
C++	X			X	
PASCAL, Modula2	X			seçimlik	
ADA	X	X	X	X	
Java	X			X	

Eşzamanlılık

(Concurrency)

- Programlama dillerindeki eş zamanlılık kavramı ile bilgisayar donanımındaki paralel çalışma birbirinden bağımsız kavramlardır.
- Eğer çalışma zamanında üst üste gelme durumu varsa donanım işlemlerinde paralellik oluşur.
- Bir programdaki işlemler eğer paralel olarak işlenebiliyorsa program eş zamanlıdır denilir.
- Eş zamanlılık kavramının karşıtı ise belirli bir sıraya göre dizilmiş ardışıl işlemlerdir.

Öncelik grafları:

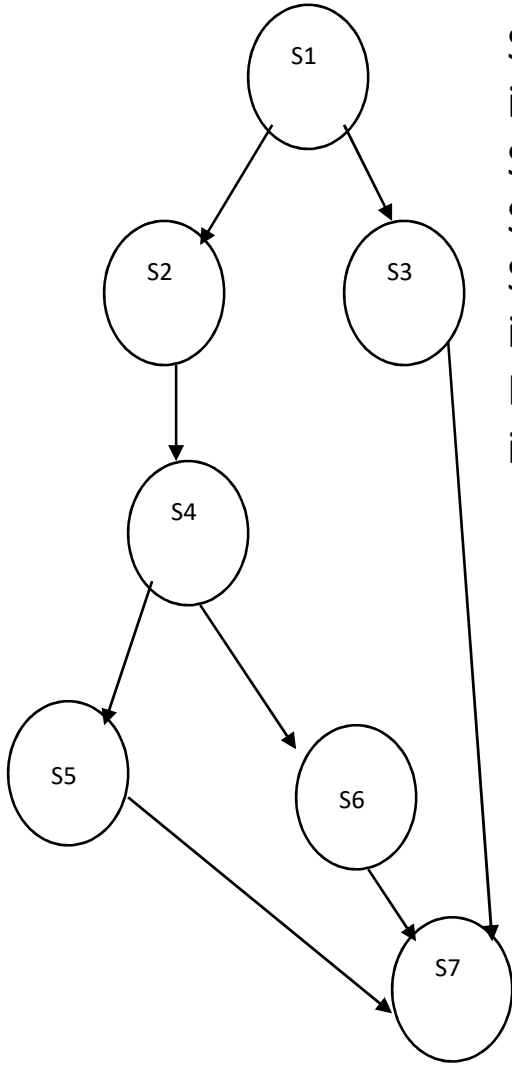
- **$a := x + y;$**
- **$b := z + 1;$**
- **$c := a - b;$**
- **$w := c + 1;$**

Burada $c := a - b$ yi hesaplamak için öncelikle a ve b 'ye değer atanması gerekmektedir.

Benzer biçimde $w := c + 1$ ifadesinin sonucu da c 'nin hesaplanmasına bağlıdır.

Diğer taraftan $a := x + y$ ve $b := z + 1$ deyimleri birbirine bağlı değildir. Bu yüzden bu iki deyim birlikte çalıştırılabilir.

Buradan anlaşılıyor ki bir program parçasında değişik deyimler arasında bir öncelik sıralaması yapılabilir. Bu sıralamanın grafik olarak gösterimine öncelik grafi denir. Bir öncelik grafi, her bir düğümü ayrı bir deyim ifade eden, döngüsel olmayan yönlendirilmiş bir graftır.



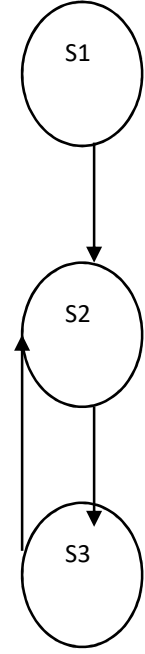
S2 ve S3 deyimleri, S1 tamamlandıktan sonra işletilebilir.

S4, S2 tamamlandıktan sonra işletilebilir.

S5 ve S6, S4 tamamlandıktan sonra işletilebilir.

S7, sadece S5,S6 ve S3 tamamlandıktan sonra işletilebilir.

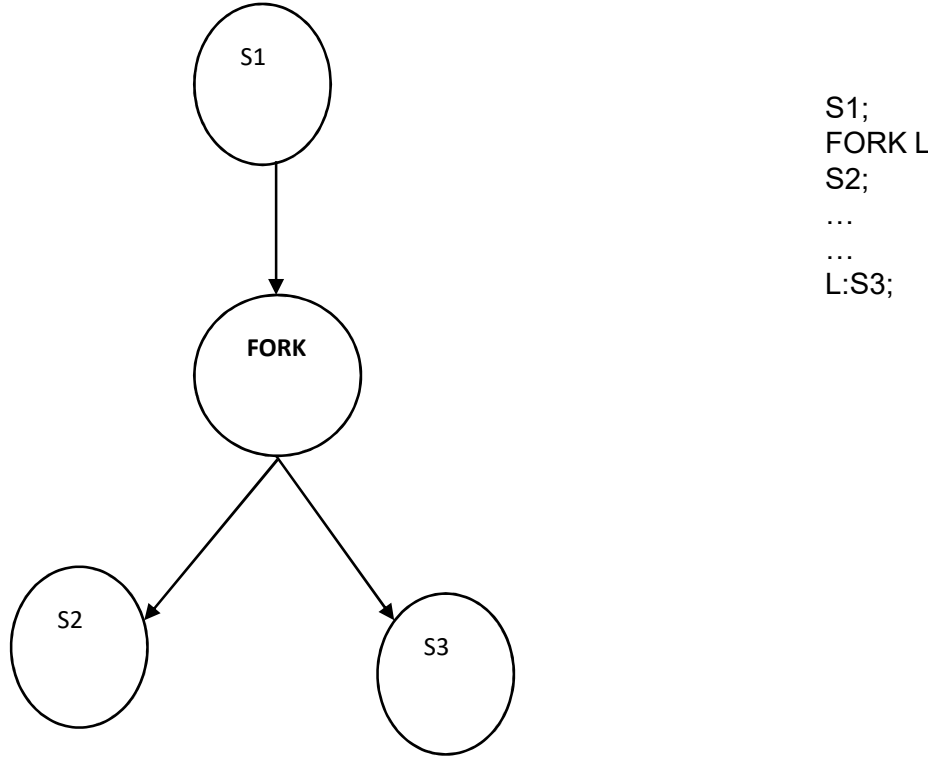
Bu örnekte S3 deyimi S2, S4, S5 ve S6 deyimleri ile eş zamanlı olarak çalışabilir.



Bu grafta görüldüğü gibi, S3 sadece S2 tamamlandıktan sonra işletilebilir. S2 deyimi ise sadece S3 tamamlandıktan sonra işletilebilir. Burada açıkça görülmektedir ki bu iki kısıtlamanın her ikisi aynı anda giderilemez. Yani bir programın akışını ifade eden öncelik grafında döngü içermemelidir.

FORK ve JOIN Yapıları:

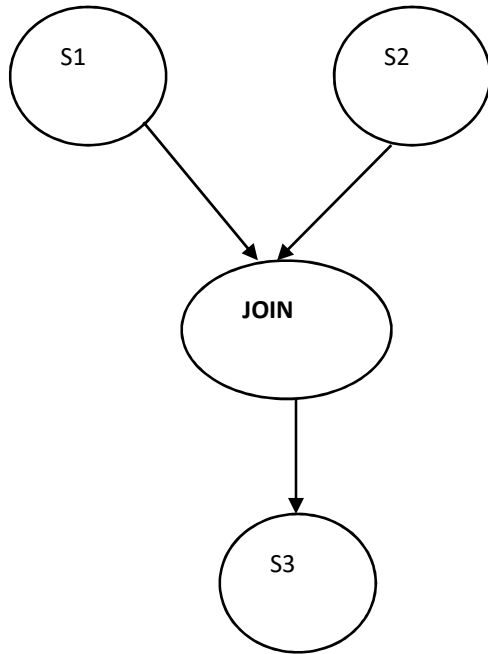
- FORK ve JOIN yapıları eş zamanlılığı tanımlayan ilk programlama dili notasyonlarından biridir. Aşağıdaki öncelik grafi bu komutlardan FORK yapısını ifade etmektedir.



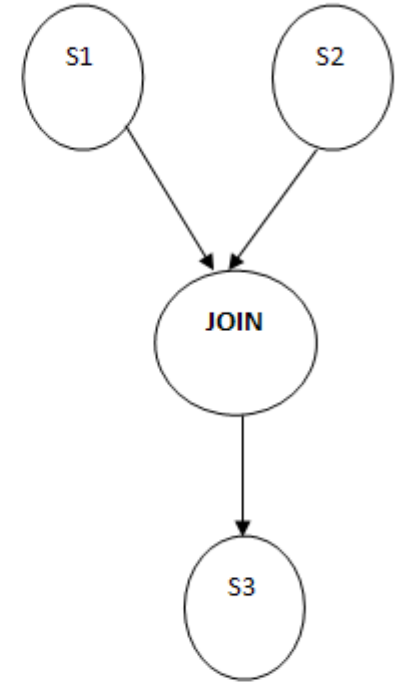
Burada eş zamanlı işlemlerden birisi L etiketi ile gösterilen deyimlerden başlarken diğeri FORK komutunu izleyen deyimlerin işlenmesi ile devam eder.

FORK L deyimi işletildiği zaman S3'de yeni bir hesaplama başlar. Bu yeni hesaplama S2'de devam eden eski hesaplama ile eş zamanlı olarak işletilir.

JOIN komutu iki eş zamanlı hesaplamayı tekrar birleştirir. JOIN komutunun öncelik grafi karşıılığı aşağıda verilmiştir.



```
Count:=2;  
FORK L1;  
...  
...  
S1;  
Go to L2;  
  
L1:S2;  
L2:JOIN count;  
S3;
```



Örnek:

S1

Count:=3;

FORK L1;

S2;

S4;

FORK L2;

S5;

Goto L3;

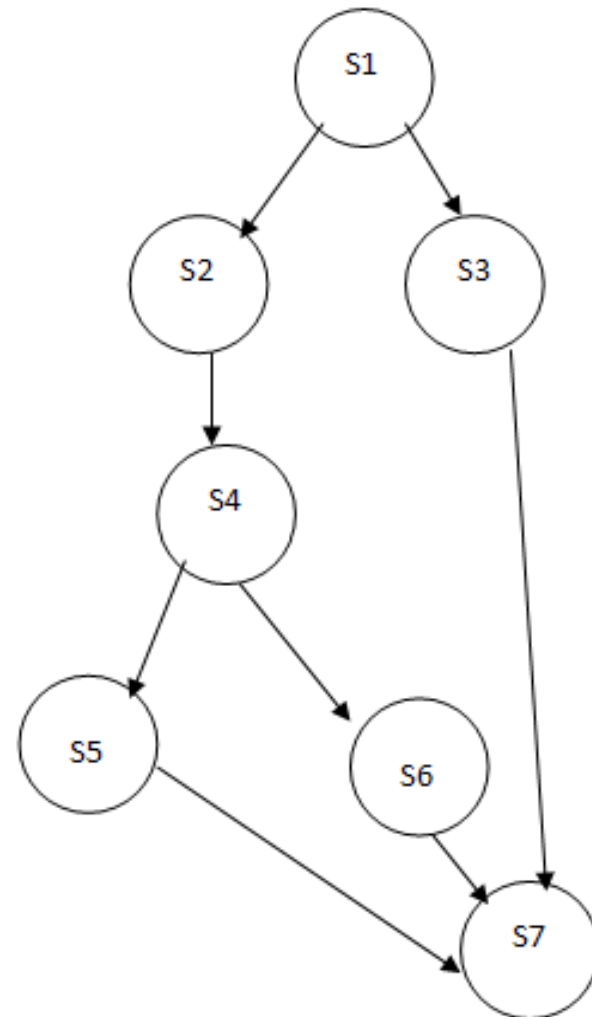
L2:S6;

Goto L3;

L1:S3;

L3: JOIN count;

S7;



Örnek

S1;

Parbegin

S3;

Begin

S2;

S4;

Parbegin

S5;

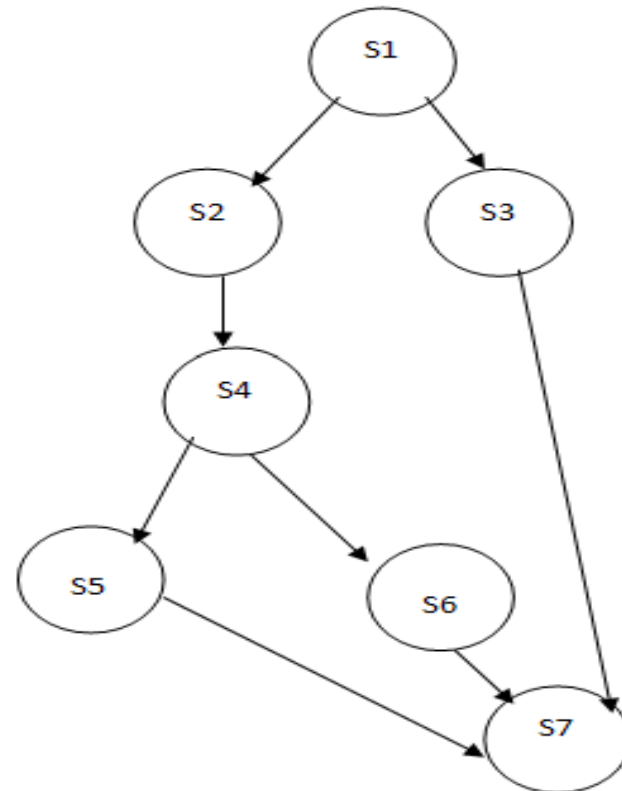
S6;

Parend;

End;

Parend;

S7;



Emir Esaslı Programlama Yönteminin Değerlendirmesi

- “Böl ve yönet” prensibine dayanır. Amaç büyük programları küçük parçalara bölerek yazılım geliştirme işini kolaylaştırmaktır.
- Ancak yazılımların karmaşıklıkları sadece boyutlarından kaynaklanmaz. Küçük problemler de karmaşık olabilir.
- Gerçek dünyadaki sistemler sadece fonksiyonlardan oluşmaz. Dolayısıyla emir esaslı yaklaşımda karmaşık bir problemin gerçeğe yakın bir modelini bilgisayarda oluşturmak zordur.
- Tasarım aşamasında verilerden çok fonksiyonlara odaklanıldığından hatalar nedeniyle veri güvenliği tehlikeye girebilmektedir.
- Kullanıcılar kendi veri tiplerini çok güçlü biçimde tanımlayamazlar.
- Programın güncellenmesi zordur.
- İşleve dayalı yöntemi de kullanarak kaliteli programlar yazmak mümkündür.

Ancak nesneye dayalı yöntem kaliteli programların oluşturulması için programcılara daha çok olanak sağlamaktadır ve yukarıda açıklanan sakıncaları önleyecek yapılara sahiptir.

Yöntemin Değerlendirmesi:

- Gerçek dünya nesnelerden oluştuğundan bu yöntem ile sistemin daha gerçekçi bir modeli oluşturulabilir. Programın okunabilirliği güçlenir.
- Nesne modellerinin içindeki veriler sadece üye fonksiyonların erişebileceği şekilde düzenlenebilirler. Veri saklama (data hiding) adı verilen bu özellik sayesinde verilerin herhangi bir fonksiyon tarafından değiştirilmesi önlenir.
- Programcılar kendi veri tiplerini yaratabilirler.
- Bir nesne modeli oluşturduktan sonra bu modeli çeşitli şekillerde defalarca kullanmak mümkündür (reusability).
- Programları güncellemek daha kolaydır.
- Nesneye dayalı yöntem takım çalışmaları için uygundur.

NESNEYE YÖNELİK PROGRAMLAMA DİLLERİ

- Programlama dilleri literatüründe sınıf ve alt sınıf kavramını ilk olarak SIMULA67dili tanıtmıştır.
- Smalltalk, Eiffel, ADA95 ve Java dillerinde tüm veriler nesne şeklindedir. Yani bu diller tam nesne yönelimlidir.
- C++ ise hem emir esaslı paradigmayı hem de nesneye yönelimli k paradigmayı destekler.

Smalltalk

- SIMULA67'de tanıtılan fikirler, Smalltalk ile güçlenmiş ve Smalltalk ile nesne yönelimli dil popüler hale gelmiştir
- Smalltalk'ta bir program sadece kalıtım hiyerarşisi içinde düzenlenmiş birbirleriyle mesajlar ile etkileşen nesne sınıflarından oluşabilir. Smalltalk'ta tüm veriler nesnelerle gösterilmek zorunda olduğu için tam nesneye yönelik bir programlama dili olarak nitelendirilir.
- Smalltalk dilinde bütün bağlamlar dinamik olarak gerçekleşir.
- Smalltalk'ta sınıfların sadece tek üst sınıfı bulunabilir (tekli kalıtım modeli).

static, abstract ve final tanımlayıcıları

- *static* özelliğindeki bir veri sahası, bir sınıfın tüm örneklerinde paylaşılır. *static* bir metod, sınıfın bir örneği yaratılmadan da çağrılabilir ve *static* bir metod, bir alt sınıfta yeniden tanımlanamaz.
- *abstract* olarak tanımlanmış bir sınıf örneklenemez ve sadece üst sınıf olabilir. Bir *abstract* metod ise bir alt sınıf tarafından gerçekleştirilmek zorundadır.
- *final* olarak tanımlanmış bir sınıfın alt sınıfları tanımlanamaz ve *final* olarak tanımlanmış bir metod, herhangi bir alt sınıfta yeniden tanımlanamaz.

Fonksiyonel Programlama

LISP, ML, Haskel, Scheme

- Emir esaslı dillerin tasarımı doğrudan doğruya von Neumann mimarisine dayanır.
- Bir imperative dilde, işlemler yapılır ve sonuçlar daha sonra kullanım için değişkenlerde(variables) tutulur. Emir esaslı dillerde değişkenlerin yönetimi karmaşıklığa yol açar.
- Fonksiyonel dillerin tasarımı Matematiksel Fonksiyonlara dayalıdır ve değişkenler(variables), matematikte olduğu gibi gerekli değildir. Kullanıcıya da yakın olan sağlam bir teorik temele sahiptir.
- Fonksiyonel programlamada , bir fonksiyon aynı parametreler verildiğinde daima aynı sonucu üretir (referential transparency).

Haskell

- Kuvvetli tipllemeli, statik kapsam bağlamalı ve tip yorumlamalı bir dildir.
- Tam olarak fonksiyonel bir dildir. (değişkenler yoktur, atama ifadeleri yoktur, hiçbir çeşit yan etki yoktur).
- Tembel değerlendirme(**lazy evaluation**) kullanır (değer gerekmediği sürece hiçbir alt-ifadeyi değerlendirme)
- Liste kapsamları(**list comprehensions**), sonsuz listelerle çalışabilmeye izin verir

Örnekler

1. `fib 0 = 1`

`fib 1 = 1`

`fib (n + 2) = fib (n + 1)
+ fib n`

2.

`fact n`

`| n == 0 = 1`

`| n > 0 = n * fact (n - 1)`

3. Liste işlemleri

– Liste gösterimi:

– `directions = ["north", "south", "east", "west"]`

– `Uzunluk(Length): #`

`#directions = 4`

– `..` operatorü ile aritmetik seriler

`[2, 4..10]` gösterimi `2, 4, 6, 8, 10` olarak değerlendirilir.

Örnekler devam

3. Liste işlemleri(devam)

– ++ ile zincirleme(Catenation)

`[10, 30] ++ [50, 70] → [10, 30, 50, 70]`

– :operatörü yoluyla

`1 : [3, 5, 7] → [1, 3, 5, 7]`

```
product [] = 1
product (a:x) = a * product x
```

```
fact n = product [1..n]
```

Haskell örnekler (devam)

4. Liste kapsamı:küme gösterimi

```
[n * n | n ← [1..10]]
```

ilk 10 pozitif tamsayının karelerinden oluşan bir liste tanımlar

```
factors n = [i | i ← [1..n div  
2],  
                n mod i == 0]
```

Bu fonksiyon verilen parametrenin bütün çarpanlarını hesaplar

Haskell örnekleri

- Quicksort(Hızlı Sıralama):

```
sort [] = []
```

```
sort (a:x) = sort [b | b ← x; b  
    <= a]
```

```
++ [a] ++
```

```
sort [b | b ← x; b > a]
```

Haskell örnekleri (devam)

5. Tembel değerlendirme(Lazy evaluation)

```
positives = [0..]
```

```
squares = [n * n | n ← [0..]]
```

(sadece gerekli olanları hesapla)

```
member squares 16
```

True döndürür

1. Aşağıdakilerden hangisi Mantıksal Programlama ile ilişkilidir?
 a) Lambda Kalkülüs b) Kalıtım c) BNF
 d) Predicate Kalkülüs e) Fortran
2. Aşağıdakilerden hangisi Mantıksal Paradigma içerisinde örnek gösterilebilecek bir programlama dilidir?
 a) Prolog b) Common Lisp c) Python
 d) ProLogic e) Basic
3. Aşağıdakilerden hangisi Fonksiyonel paradigmanın avantajı veya avantajlarıdır?
 I. Basit Semantik
 II. Basit Sözdizim
 III. Yüksek Verim
 IV. Otomatik Eşzamanlılık
 a) Yalnız I b) I-II-III
 c) I-II-IV d) Yalnız IV
 e) III-IV
4. Aşağıdaki terimlerden hangisi nesneye yönelik paradigma ile alakalı **değildir**?
 a) Kalıtım b) Geç Bağlama (Late Binding)
 c) Çok Biçimlilik d) Kapsülleme
 e) Ayrışma Ağacı
5. Aşağıdakilerden hangisi Nesneye Yönelik Programlamanın özelliklerinden **değildir**?
 a) Örnekler sınıflardan kurulum yoluyla oluşturulur.
 b) Kapsül haline getirilmiş bir nesnenin alanları dışa kapalıdır.
 c) Kalıtım alınan üst sınıftır.
 d) public olarak tanımlanmamış bir sınıf türetilemez.
 e) Bir nesnenin tüm metotları mesaj arayüzü olarak adlandırılır.

6. İstisnai durumların önlenmesi bakımından C++ dili aşağıdaki terimlerden hangisi veya hangilerini destekler?
 I.try
 II.catch
 III.finally
 IV.throw
 V.throwes
 a) I ve II b) I, II ve III c) I, II, IV ve V
 d) I, II ve IV e) I, II, III ve IV
7. Aşağıdakilerden hangisi Yapısal Programlamanın ilgilendiği başlıklardır?
 I. Alt Programlama
 II. Kontrol Yapıları
 III. Döngü Yapıları
 IV. Kalıtım
 a) Yalnız I b) I-II-III
 c) II-III d) III-IV
 e) II-III-IV
8. Aşağıdaki Programlama dillerinin değerlendirme kriterlerinden hangisinin desteği arttığında güvenliğin azalabileceği söylenebilir?
 a) Esneklik b) İfade Gücü c) Taşınabilirlik
 d) I/O Kolaylığı e) Yapısallık

9. Alt indis sınırı sıfır olan ve her bir elemanın sözcük uzunluğu c olan bir A dizisinin k. indisinin adresi hangi şıkta doğru olarak verilmiştir?
 a) Adres(A[k-1])=Adres(A[0])+(k-1)*c
 b) Adres(A[k])=Adres(A[0])+(k-1)*c
 c) Adres(A[k])=Adres(A[0])+(k)*c
 d) Adres(A[k-1])=Adres(A[0])+(k)*c
 e) Adres(A[k+1])=Adres(A[0])+(k)*c

10. Derlenme sürecinde aşağıdaki aşamalardan hangisinde regüler ifade, DFA ve gramer gibi araçlarla tanımlanamayan bilgiler içerir?
 a) Optimizasyon b) Ara Kod Dönüştürücü
 c) Lexical Analiz d) Syntax Analiz
 e) Semantic Analiz

11. Hata yakalamada Java dilindeki throw ifadesi C dilindeki aşağıdaki hangi fonksiyon ile ilişkilendirilebilir?
 a) longjmp b) setjmp c) jmp_buf d) ifjmp e) thjmp

12. Yandaki kod sırasıyla statik ve dinamik kapsama bağlamaya göre değerlendirildiğinde ekran çıktısı ne olur?

```

int v;
int main() {
  v=7;
  T1();
  T2();
  return 0;
}

void T1() {
  int v=8;
  T2();
}

void T2() {
  void T3(int v) {
    printf("%d",v);
  }
}
  
```

 a) 77 88
 b) 88 77
 c) 77 77
 d) 88 88
 e) 77 87

13. Aşağıdaki dillerin hangisinde Heap bellek bölgesinde oluşan çöpleri otomatik toplayacak bir mekanizma **yoktur**?
 a) Pascal b) Python c) Java d) C# e) Ada

14. Aşağıdaki BNF ifadesi EBNF ile nasıl ifade edilebilir?
 <degisken> ::= <harf> | <degisken><harf> | <degisken><rakam>
 a) <degisken> ::= <harf> {<degisken><harf><rakam>}
 b) <degisken> ::= <harf> {<harf> | <rakam>}
 c) <degisken> ::= <harf> {<harf><rakam>}
 d) <degisken> ::= <harf> {<harf> | <rakam>}
 e) <degisken> ::= <harf> {<harf><rakam>}

15. C# dilinde void fonk(int x,out int y) { ... } şeklinde tanımlanmış bir fonksiyon ile aynı etkiyi yapmak için C++ dilinde nasıl tanımlanmalıdır?
 a) void fonk(int x, int **y){ ... } b) void fonk(int x, int *y){ ... }
 c) void fonk(int x, int *&y){ ... } d) void fonk(int x, int &y){ ... }
 e) void fonk(int x, int y){ ... }

16. Java'nın yandaki parametre çağırma yöntemlerinden hangisini veya hangilerini desteklediği söylenebilir?
 I. Değer ile çağırma
 II. Adres ile çağırma
 III. Referans ile çağırma
 IV. Sonuç ile çağırma
 a) Yalnız I b) II-III c) I-IV d) I-III e) I-II
17. Diamond problemi neden kaynaklanabilir?
 a) Kapsülleme b) Basit Sözdizim c) Çoklu Kalıtım
 d) Soyut Sınıf e) Geç Bağlama

18. Fork ve Join kullanılarak gerçekleştirilen eş zamanlılık yapısına karşılık gelen öncelik grafi hangisidir?



19. Aşağıdaki c++ koduna karşılık gelen, Lisp kodu aşağıdakilerden hangisidir?

```

int Hadi(int a,int b=10){ return a+b; }

a) (defun Hadi (a b)
  (setq b 10)
  (+ a b)
)

b) (defun Hadi (a (optional b))
  (if (eq b nil) (setq b 10))
  (+ a b)
)

c) (defun Hadi (a b)
  (if (eq b nil) (setq b 10))
  (+ a b)
)

d) (defun Hadi (optional a b)
  (if (eq b nil) (setq b 10))
  (+ a b)
)
  
```

e) Tanımlanamaz

20. typedef enum FF{false, true}class; ifadesi ile C dilinde Java'daki hangi türün benzetimi yapılmış olabilir?
 a) class b) int c) boolean d) char e) byte

AAAAAA