Derleyici Tasarımı Temelleri Nedir?

Çalışma Kağıdı

Derleyici tasarımı temelleri, kaynak kodunun analiz edilip hedeflenen dile çevrilmesi için gereken teorik bilgileri ve pratik yöntemleri kapsar.

Sorular

1. Aşağıdakilerden hangisi derleyici tasarımının temel aşamalarından biri DEĞİLDİR?

A) Sözcüksel Analiz
B) Sözdizimsel Analiz
C) Veritabanı Yönetimi
D) Hedef Kod Üretimi

2. Parse tree (ayrıştırma ağacı) hangi aşamada oluşturulur?

A) Sözcüksel Analiz
B) Sözdizimsel Analiz
C) Anlamsal Analiz
D) Kod Optimizasyonu

3. Derleyicinin temel amacı nedir?

A) Kullanıcı arayüzü tasarlamak
B) Kaynak kodu makine koduna çevirmek
C) Ağ iletişimi sağlamak
D) Veri saklamak

4. Bir C++ programının derlenmesi hangi temel aşamalardan geçer?

5. Derleyicinin önemi nedir?

6. Tanımla: Sözcüksel Analiz (Lexical Analysis)

7. Tanımla: Sözdizimsel Analiz (Syntax Analysis)

8. Tanımla: Anlamsal Analiz (Semantic Analysis)

Cevap Anahtarı

1. C) Veritabanı Yönetimi — Veritabanı yönetimi derleyici tasarımının doğrudan bir aşaması değildir, ancak derleyiciler tarafından kullanılan veri yapıları (sembol tablosu gibi) veritabanı kavramlarıyla ilişkilendirilebilir.

2. B) Sözdizimsel Analiz — Parse tree, sözdizimsel analiz aşamasında, token dizisinin dilbilgisine uygunluğunu göstermek için oluşturulur.

3. B) Kaynak kodu makine koduna çevirmek — Derleyicinin ana görevi, insanların yazdığı kaynak kodu, bilgisayarın anlayabileceği makine koduna dönüştürmektir.

4. 1. Lexical Analysis (Sözcüksel Analiz): Kaynak kodu token'lara ayırır. 2. Syntax Analysis (Sözdizimsel Analiz): Token'ların dilbilgisine uygunluğunu kontrol eder (parse tree oluşturur). 3. Semantic Analysis (Anlamsal Analiz): Kodun anlamını kontrol eder (tip kontrolü vb.). 4. Intermediate Code Generation (Ara Kod Üretimi): Hedef dilden bağımsız bir ara kod üretir. 5. Code Optimization (Kod Optimizasyonu): Üretilen kodu daha verimli hale getirir. 6. Target Code Generation (Hedef Kod Üretimi): Nihai makine kodunu üretir.

5. Derleyiciler, insanların anlayabileceği yüksek seviyeli programlama dillerini, makinelerin işleyebileceği düşük seviyeli dillere çevirerek yazılım geliştirme sürecini kolaylaştırır ve verimliliği artırır.

6. Kaynak kodu anlamlı birimlere (token) ayırma işlemidir.

7. Token dizisinin programlama dilinin dilbilgisine uygun olup olmadığını kontrol eder.

8. Kodun anlamsal doğruluğunu, örneğin tip uyumluluğunu kontrol eder.