**ALGORTİMA**

**1)Algoritma nedir? Algoritmanın Gerekliliği ve avantajları nedir?**

Algoritma, bir problemi çözmek veya belirli bir görevi gerçekleştirmek için izlenmesi gereken belirli adımların bir dizisidir. Bu adımların her biri, bilgisayar programları veya insanlar gibi bir işlemci tarafından uygulanabilir. Algoritmalar, hesaplama, veri işleme, otomasyon ve problem çözme gibi birçok farklı alanın temelini oluşturur.

**2)Program yazılmadan önce algoritması mı hazırlanmalı yoksa akış diyagramı mı çizilmelidir?**

Bir program yazmadan önce çalıştırmadan önce veya diyagram diyagramını çizmeden, yazılım geliştirme sisteminin önemli bir parçası ve hangi yöntemi kullanarak projenizin özelliklerine ve kişisel tercihlerinize bağlıdır. Her iki yöntem de faydalı olabilir, ancak projenizin karmaşıklığına ve özelliklerine göre tercih edilebilecek yöntemler değişebilir.

Algoritma Hazırlama:

Algoritma, bir programın çalışmasının adımlarını metin tabanlı bir şekilde gösterir. Bu, programın ne devam etmesi, adım adım anlaşılmasına yardımcı olur.

Algoritma yazmak, genel insan anlamanıza ve tasarım kararlarınızı vermenize yardımcı olabilir.

Daha küçük paketler veya daha basit problemler için parçaları ayırmak yeterli olabilir.

Algoritma, daha fazla teknik detay içerebilir ve kod yazmaya geçiş öncesinde bir üretim tasarımının geliştirilmesine yardımcı olur.

Akış Diyagramı Çizme:

Akış diyagramları, bir program mantığı ve dağıtılan grafiksel bir şekilde gösterilir. İşlem sırasını ve karar noktalarını görsel olarak temsil eder.

Büyük ve karmaşık yapılar için akış diyagramları daha anlaşılır ve yönetilebilir olabilir. Projenin daha net bir şekilde görmeye yardımcı olması mümkündür.

Özellikle işbirliği gerektiren büyük ekiplerle birlikte akış diyagramları, herkesin genel anlamda anlamasına yardımcı olabilir.

Projenizin büyüklüğü, karmaşıklığı ve ekibinizin yapısı gibi faktörler, hangi yaklaşım seçeneğini belirlemeyi belirler şemaları yazmak, hem de boru hattı diyagramını çizmek, daha iyi anlaşılmasına ve daha iyi tasarlanmasına yardımcı olur. İdeal olarak, bu iki yöntemi bir arada kullanabilirsiniz. Algoritma hazırlandıktan sonra akış diyagramı çizmek veya akış diyagramı üzerinde çalışanlarınızı göz önünde bulundurmak, programın çalıştırılması sırasında daha fazla netlik sağlar.

**3)Algoritma hazırlanırken dikkat edilmesi gereken hususları açıklayınız.**

1. Netlik ve Basitlik: Algoritmanız mümkün olduğunca net ve basit olmalıdır. Karmaşık bir algoritma, hem yazım hem de anlama açısından daha zor olabilir.

2. Adım Adım İşlemler: Algoritma, belirli adımların açıkça tanımlanmış olduğu bir işlem sırasını takip etmelidir. Her adımın ne yaptığı açık olmalıdır.

3. Başlangıç ve Bitiş: Algoritmanın başlangıcı ve bitişi belirtilmelidir. Hangi koşullarda algoritma başlar ve hangi koşullarda sona erer?

4. Kontrol Yapıları: Kontrol yapıları, algoritmanın kararlarını ve tekrarlamalarını yönetir. İf-else ifadeleri, döngüler ve diğer kontrol yapıları, algoritmanın akışını kontrol eder.

5. Veri Girişi ve Çıkışı: Algoritmanın verileri nasıl alacağı ve sonuçları nasıl üreteceği net bir şekilde belirtilmelidir.

6. Mantıksal Doğruluk: Algoritma mantıklı olmalıdır. Yani her adım, amacınıza ve probleminize uygun olmalıdır.

7. Modülerlik: Algoritmayı parçalara bölmek ve her parçayı ayrı ayrı ele almak daha büyük ve karmaşık algoritmaların daha kolay anlaşılmasını sağlar.

8. Döngülerin ve Yinelemelerin Sınırları: Eğer bir algoritma içinde döngüler veya yinelemeler kullanılıyorsa, bunların kaç kez çalışacağı veya hangi koşullar altında sona ereceği belirtilmelidir.

9. Performans ve Verimlilik: Algoritmanın işlemci ve hafıza kullanımı açısından verimli olması önemlidir. Fazla karmaşık veya gereksiz işlemlerden kaçınılmalıdır.

10. Hata Kontrolleri: Algoritma, hatalı girişlere veya beklenmeyen durumlara nasıl tepki vereceğini içermelidir. Bu, hata kontrol mekanizmalarının eklenmesini gerektirebilir.

11. Güvenlik: Eğer algoritma hassas verileri işliyorsa veya güvenlik endişeleri varsa, güvenlik önlemleri alınmalıdır.

12. Belgelendirme: Algoritmanın ne yaptığını ve nasıl kullanılması gerektiğini anlatan belgeler oluşturulmalıdır.

13. Test ve Hata Düzeltme: Algoritma yazıldıktan sonra, test edilmesi ve hataların düzeltilmesi gereklidir.

Algoritma, bir programı geliştirme sürecinin önemli bir başlangıç adımıdır ve programın doğru çalışmasını sağlamak için titizlikle oluşturulmalıdır. İyi bir algoritma, kodlama ve hata düzeltme süreçlerini daha verimli hale getirebilir.

**4)Değişken nedir? Programlarda neden değişkenlere ihtiyaç duyulmaktadır?**

Değişken, programlama dillerinde en temel kavramlardan biridir ve verileri saklamak, işlemek ve yönetmek için kullanılan bir tür isimlendirmeyi ifade eder. Değişkenler, programlarda verileri geçici olarak saklamak ve bu verilere erişmek için kullanılır. İşte değişkenlerin ne olduğu ve neden programlarda ihtiyaç duyulduğu konularında daha fazla açıklama:

1. Veri Saklama: Programlar, birçok farklı türde veriyle çalışır, örneğin sayılar, metinler, diziler, yapılar (structs), sınıflar (classes) gibi veri türleri. Bu verileri geçici olarak saklamak

ve işlemek için değişkenlere ihtiyaç vardır. Bir değişken, bir değeri veya veriyi saklamak için kullanılır.

2. Değer Atama: Değişkenlere değer atama işlemi, programların verileri saklamasını sağlar. Örneğin, bir sayıyı veya metni bir değişkene atayabilirsiniz ve daha sonra bu değişkeni kullanarak o veriye erişebilirsiniz.

*Örnek (Python programlama dili):*

*python*

*sayi = 5 # "sayi" adlı değişkene 5 değerini atar*

*metin = "Merhaba, dünya!" # "metin" adlı değişkene metin değeri atar*

3. Veri İşleme: Programlar, değişkenler aracılığıyla verileri işler. Değişkenlerin içeriğini değiştirerek matematiksel işlemler yapabilir, metinleri birleştirebilir, dizilerde gezinebilir ve daha fazlasını yapabilirsiniz.

*Örnek (Python programlama dili):*

*python*

*sayi1 = 10*

*sayi2 = 5*

*toplam = sayi1 + sayi2 # sayi1 ve sayi2'nin toplamını "toplam" adlı değişkene atar*

4. Veri Yönetimi: Değişkenler, programların verileri düzenli bir şekilde saklamasını ve yönetmesini sağlar. Bu, verilerin program içinde daha kolay erişilebilir ve anlaşılır olmasını sağlar.

5. Koşullu İfadeler ve Döngüler: Programlar, değişkenlerin değerlerine göre kararlar alabilir ve tekrarlayan işlemler gerçekleştirebilir. Değişkenler, koşullu ifadeler (if-else) ve döngüler (for, while) gibi kontrol yapılarıyla birleştirildiğinde programların akışını kontrol etmek için kullanılır.

*Örnek (Python programlama dili):*

*python*

*notu = 85*

*if notu >= 60:*

*print("Geçtiniz.")*

*else:*

*print("Kaldınız.")*

Bu nedenlerle değişkenler, programlama dillerinde temel bir yapı taşıdır ve programların verileri saklaması, işlemesi ve yönetmesi için önemlidir. Her programlama dilinin değişken kullanımı ve kuralları farklı olabilir, ancak temel kavramlar genellikle benzerdir.

**5)Sayaçlar, nerelerde ve niçin kullanılır?**

Sayaçlar (counters), programlamada sıklıkla kullanılan bir yapı veya değişken türüdür. Sayaçlar, belirli koşullar altında veya döngülerde tekrar eden bir olayın sayısını, durumunu veya ilerlemesini izlemek ve kaydetmek için kullanılır. Aşağıda sayaçların nerelerde ve niçin kullanıldığına dair bazı örnekler verilmiştir:

1. Döngülerde İlerleme Takibi: Sayaçlar, döngülerin ilerlemesini izlemek ve kontrol etmek için kullanılır. Örneğin, bir for döngüsü kullanırken her döngü adımında sayaç bir birim artırılabilir ve belirli bir sayıda işlem yapılmasını sağlayabilir.

*Örnek (Python programlama dili):*

*python*

*for i in range(5):*

*print(f"Adım {i}")*

2. Liste Elemanlarına Erişim: Sayaçlar, bir liste, dizi veya dizi benzeri veri yapısındaki belirli elemanlara erişmek için kullanılabilir. Her bir elemanın bir indeksi vardır ve sayaçlar, belirli bir indekse sahip elemana erişmek için kullanılabilir.

*Örnek (Python programlama dili):*

*python*

*liste = [10, 20, 30, 40, 50]*

*sayaç = 2*

*print(liste[sayaç]) # liste içindeki 2. elemanı (30) yazdırır*

3. Koşullu İfadelerde Sayma: Sayaçlar, belirli koşulların kaç kez sağlandığını saymak için kullanılabilir. Örneğin, belirli bir koşulun ne kadar süreyle geçerli olduğunu izlemek amacıyla kullanılabilirler.

*Örnek (Python programlama dili):*

*python*

*sayac = 0*

*for sayi in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:*

*if sayi % 2 == 0:*

*sayac += 1*

*print(f"Çift sayıların sayısı: {sayac}")*

4. Hata Takibi: Sayaçlar, programlar hata ayıklanırken veya hataların izlenmesi gereken durumlarda kullanılabilir. Özellikle büyük veri işleme işlemlerinde, hata takibi için kullanılırlar.Sayaçlar, programların daha karmaşık işlemleri gerçekleştirmesini ve belirli durumları izlemesini sağlayan önemli bir araçtır. İşlem sırasında kaç kez belirli bir olayın gerçekleştiğini veya belirli bir durumun ne kadar süreyle mevcut olduğunu izlemek gibi birçok kullanım alanı vardır.

**6)Aşağıdaki algoritmanın sonucu nedir?**

1. Başla

2. T=0

3. S=0

4. Eğer S>10 ise Git 8

- S = 0 (Şu anki değer)

- S <= 10 olduğu için Git 5'e geç.

5. T=T+2\*S

- T = 0 + 2 \* 0 = 0

- S = 0 + 2 = 2

6. S=S+2

- S = 2 + 2 = 4

7. Git 4

- 4. adıma geri dön (S>10 kontrolü)

- S = 4 (Şu anki değer)

- S <= 10 olduğu için Git 5'e geç.

8. T=T+2\*S

- T = 0 + 2 \* 4 = 8

- S = 4 + 2 = 6

9. S=S+2

- S = 6 + 2 = 8

10. Git 4

- 4. adıma geri dön (S>10 kontrolü)

- S = 8 (Şu anki değer)

- S <= 10 olduğu için Git 5'e geç.

11. T=T+2\*S

- T = 8 + 2 \* 8 = 24

- S = 8 + 2 = 10

12. Eğer S>10 ise Git 8

-S = 10 (Şu anki değer)

- S > 10 olduğu için 8. adıma git.

13. Yaz T  
- T = 24 yazdırılır.

14. Dur

Sonuç olarak, bu algoritmanın sonucu T'nin değeri olan 24'tür.

**7)Aşağıdaki algoritmanın sonucunu hesaplayınız?**

1. Başla

2. F=1: F değişkenine 1 değeri atanır.

3. S=20: S değişkenine 20 değeri atanır.

4. Eğer S<1 ise Git 9: S değişkeni 1'den büyük olduğu için 5. adıma geçilir.

5. S= S-3: S değişkeninden 3 çıkarılır, yani S = 17 olur.

6. F= F+S: F değişkeni, S değişkeni ile toplanır, yani F = 1 + 17 = 18 olur.

7. F= F+2: F değişkenine 2 eklenir, yani F = 18 + 2 = 20 olur.

8. Git 4: 4. adıma geri dönülür.

9. Yaz F: F değişkeninin değeri ekrana yazdırılır. Bu durumda F'nin değeri 20 olduğu için ekrana "20" yazılır.

10. Dur: Algoritma sona erer.

Sonuç olarak, verilen algoritma çalıştırıldığında ekrana "20" yazdırılır.

**8)Girilen üç sayıdan en büyüğünü bulan algoritmayı yazınız.**

1. Başla

2. Sayıları a, b ve c olarak tanımla.

3. EnBüyük değişkenini a olarak başlat.

4. Eğer b > EnBüyük ise EnBüyük = b

5. Eğer c > EnBüyük ise EnBüyük = c

6. EnBüyük'ü ekrana yazdır.

7. Dur

Bu algoritma, ilk olarak a'nın en büyük olduğunu varsayar ve daha sonra b ve c'yi karşılaştırarak en büyük değeri belirler. Eğer b veya c, EnBüyük'ten daha büyükse, EnBüyük değeri güncellenir. Son olarak, EnBüyük değeri ekrana yazdırılır.

Bu algoritma, üç sayının en büyüğünü bulmak için kullanılabilir.

**9)Girilen üç sayıyı küçükten büyüğe doğru sıralayan algoritmayı hazırlayınız.**

1. Başla

2. Sayıları A, B ve C olarak al

3. EnKucuk = A // Başlangıçta A'yı en küçük sayı olarak kabul ediyoruz

4. Orta = B

5. EnBuyuk = C

6. Eğer B < EnKucuk ise

7. Orta = EnKucuk

8. EnKucuk = B // Eğer B, şu ana kadar en küçük sayıdan daha küçükse, B'yi en küçük sayı olarak güncelle

9. Eğer C < EnKucuk ise

10. Orta = EnKucuk

11. EnKucuk = C // Eğer C, şu ana kadar en küçük sayıdan daha küçükse, C'yi en küçük sayı olarak güncelle

12. Eğer C > EnBuyuk ise

13. Orta = EnBuyuk

14. EnBuyuk = C // Eğer C, şu ana kadar en büyük sayıdan daha büyükse, C'yi en büyük sayı olarak güncelle

15. Eğer B > EnBuyuk ise

16. Orta = EnBuyuk

17. EnBuyuk = B // Eğer B, şu ana kadar en büyük sayıdan daha büyükse, B'yi en büyük sayı olarak güncelle

18. Sonuç = "Sıralama: " + EnKucuk + ", " + Orta + ", " + EnBuyuk

19. Yaz Sonuç

20. Dur

Bu algoritma, kullanıcının üç farklı sayıyı girmesini bekler, bu sayıları A, B ve C değişkenlerine atar ve ardından bu sayıları küçükten büyüğe doğru sıralar. Sonuç olarak, sıralanmış sayılar Sonuç değişkenine yazdırılır.

**10)1-99 arasındaki tek ve çift sayıların toplamları ve çarpımlarını ayrı ayrı hesaplayan programın algoritmasını hazırlayınız.**

1. Toplam değişkenlerini başlatın: tekToplam = 0 ve çiftToplam = 0 olarak başlatın.

2. Çarpım değişkenlerini başlatın: tekÇarpım = 1 ve çiftÇarpım = 1 olarak başlatın.

3. 1'den 99'a kadar olan sayılar için bir döngü oluşturun.

4. Her sayıyı kontrol edin:

a. Eğer sayı tekse:

- Tek toplamı güncelleyin: tekToplam += sayı

- Tek çarpımı güncelleyin: tekÇarpım \*= sayı

b. Eğer sayı çiftse:

- Çift toplamı güncelleyin: çiftToplam += sayı

- Çift çarpımı güncelleyin: çiftÇarpım \*= sayı

5. Döngü bittikten sonra, tekToplam, çiftToplam, tekÇarpım ve çiftÇarpım değişkenlerini yazdırın.

İşte bu algoritmanın Python programı:

python

tekToplam = 0

çiftToplam = 0

tekÇarpım = 1

çiftÇarpım = 1

for sayı in range(1, 100):

if sayı % 2 == 0: # Çift sayı

çiftToplam += sayı

çiftÇarpım \*= sayı

else: # Tek sayı

tekToplam += sayı

tekÇarpım \*= sayı

print("Tek sayıların toplamı:", tekToplam)

print("Çift sayıların toplamı:", çiftToplam)

print("Tek sayıların çarpımı:", tekÇarpım)

print("Çift sayıların çarpımı:", çiftÇarpım)

Bu program, 1'den 99'a kadar olan tek ve çift sayıların toplamını ve çarpımını hesaplayacaktır.