**Arrays(Diziler)**

Diziler, en kısa tanımla elemanları aynı türden bellek bloklarında ardışıl bir şekilde tutulan veri yapılarıdır. Şüphesiz bu tanım bilinmelidir. Diziler çoğu programlama dilinde de vardır ve bu şekilde tanımlanmaktadır. Diziler diğer veri yapılarına göre oldukça basit bir mantığa dizilerin dinamik olarak oluşturulması dinamik bellek yönetimi konusunda ele alınacaktır. Bu döküman içerisinde yalnızca derleme zamanında oluşturulan dizileri ele almaktadır. Bu kısımlar ileride mutlaka ele alınacaktır. Otomatik ömürlü dizilerle ilgili bilinmesi gereken en önemli şeylerden biri bu dizilerin boyutlarının derlenme zamanı içerisinde belirmesidir. Örneğin aşağıdaki gibi bir bildirim bir sentaks hatasına neden olur:

int arr[];//error

Şüphesiz dizinin boyutu bir makro ile de bildirilebilir. Ancak bu makronun sabit ifadesi belirtmesi de şart değildir. Ama genellikle aşağıdaki gibi sentaks kullanılır:

#define SIZE 100

int main(void)

{

int arr[SIZE];//geçerli

}

**Anahtar Notlar:** Bir fonksiyonun parametre değişkeni dizi olamaz. Örneğin aşağıdaki aşağıdaki örnekteki fonksiyonun her ne kadar arr parametresi bir dizi türünden gibi görülse de bu int türden bir pointer türüdür. Buna göre aşağıdaki örneği dikkatli bir şekilde inceleyiniz:

void func(int arr[],int size)

{

//kod

}

İlerde dizilerin pointerlarla birlikte kullanımına değineceğiz.

**Anahtar Notlar :** Bilinmesi gereken önemli bilgilerden birisi de bir dizi türünün örneğin int türden bir diziyle int[] (int köşeli parantez) türünden oluşur. Pek çok kişinin ilk tahmini gibi int türünden değildir.

Dizilerde en önemli yere sahip operatörlerden bir tanesi olan ve operatör tablosunun birinci öncelik seviyesinde bulunan [](index of) operatörüdür. Operatör öncelik tablosunun birinci öncelik seviyesindeki diğer operatörleri(fonksiyon çağırma operatörü hariç) yapılar(structures) konusunda değineceğiz. Dizilerinin indisi sıfırdan başlar yani dizinin birinci elemanı aslında sıfır indisli elemanıdır. Hatırlarsak bu yazılarda da böyleydi. Çünkü onlar da hatırlarsak char dizilerdi. Orda da olduğu gibi dizinin en son elemanının indisi aslında dizinin size – 1 indisli elemanıdır. Yani size – 1 indisli elemanı aslında size. elemanıdır.

Bu operatör kullanılarak dizinin elemanlarına erişim sağlanabilir. Diziye ilk değer de verilebilir sonrasında keyfi olarak rastgele set de edilebilir. İşte bazı örnek kodlar aşağıdaki gibidir:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define SIZE 20

#define randomize() srand(time(NULL))

void setRandomArray(int arr[],int size)

{

for(int i = 0;i < size;++i)

{

arr[i] = rand() % 100; // 0 – 100 arası rastgele sayı üret.

}

}

void printArray(int arr[], int size)

{

for(int i = 0;i < size;++i)

}

int main(void)

{

randomize();

int a[] = {1,2,3}; //ilk değer verme

int b[SIZE];

setRandomArray(b,SIZE);

}

Bir diziye ilk değer verildiği takdirde boyutunun belirtilmesi zorunluluğu yoktur. Ancak belirtilmesi de geçerlidir. Örneğin:

int arr[3] = {1,2,3};//legal

Diziler default olarak hayatına çöp değerle başlarlar. Ancak sıfırla başlatmak da mümkündür. Dizi 100 elemanlı bile olsa aşağıdaki gibi bir sentaks ile tüm elemanlara 0 değeri verilebilir:

int arr[100] = {0}; //tüm elemanlara 0 atamakla aynı anlama gelir.

Peki ardışıl atama yapmak yerine istediğimiz elemana da değer verebilir miyiz?

Cevap : Evet, aşağıdaki örnekte gösterildiği gibi bir sentaks özelliğiyle dizinin istenilen elemanına değer atanabilir:

int arr[] = {[4] = 15, [10] = 97};

Bu örnekte aslında belirtilen ilk değer verme şu şekildedir : 4 indisli elemana 15 ata, 10 indisli elemana 97 ata diğer elemanlara da 0 ata, ayrı dizi boyutu da maksimum indis ne ise ona göre belirleneceğinden dizi boyutu 11 olarak belirlenir.

Örnek Soru: Kendisine gönderilen dizinin en büyük elemanını geri döndüren getMax isimli fonksiyonu yazınız ve test ediniz:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define randomize() srand(time(NULL))

#define SIZE 20

void printArray(int arr[],int size)

{

for(int i = 0;i < size;++i)

{

printf("%d ",arr[i]);

}

printf("\n");

}

void setRandomArray(int arr[],int size)

{

for(int i = 0;i < size;++i)

{

arr[i] = rand() % 100;

}

}

int getMax(int arr[],int size)

{

int max = arr[0];

for(int i = 1;i < size;++i)

{

if(max < arr[i])

{

max = arr[i];

}

}

return max;

}

int main(void)

{

randomize();

int arr[SIZE];

setRandomArray(arr,SIZE);

printArray(arr,SIZE);

printf("Max : %d\n", getMax(arr,SIZE));

}

Örnek Soru: Rastgele üretilmiş bir diziyi tersten yazdıran reverseArray isimli fonksiyonu yazınız ve test ediniz.

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define randomize() srand(time(NULL))

#define SIZE 20

void printArray(int arr[],int size)

{

for(int i = 0;i < size;++i)

{

printf("%d ",arr[i]);

}

printf("\n");

}

void setRandomArray(int arr[],int size)

{

for(int i = 0;i < size;++i)

{

arr[i] = rand() % 100;

}

}

void reverseArray(int arr[],int size)

{

for(int i = 0;i < size / 2;++i)

{

int temp = arr[size - 1 - i];

arr[size - 1 - i] = arr[i];

arr[i] = temp;

}

}

int main(void)

{

randomize();

int arr[SIZE];

setRandomArray(arr,SIZE);

printArray(arr,SIZE);

reverseArray(arr,SIZE);

printArray(arr,SIZE);

}

**Dizilerde Sıralama Algoritmaları**

**Anahtar Notlar:** Algoritmaların karmaşıklıkları için en popüler notasyonlardan biri Big (O) notasyonudur. Bu notasyon örneğin kodda hiçbir döngü yoksa bu durumda algoritmanın karmaşıklığı O (1) karmaşıklığı olacaktır. Eğer programda tek bir döngü varsa bu durumda algoritmanın karmaşıklığı O (n) karmaşıklığı olacaktır. Eğer programda iç içe döngü varsa ama her dönüşte bir döngünün dönüşü yarıya düşüyorsa bu durumda algoritmanın karmaşıklığı O (n log n) karmaşıklığı olacaktır. Eğer dönüş sayısı yarıya düşmeyen iç içe iki döngü olduğu zaman algoritmanın karmaşıklığı O (n^2) karmaşıklığı olacaktır. Son olarak iç içe üç döngü varsa bu durumda algoritmanın karmaşıklığı O (n ^ 3) karmaşıklığıdır.

Sıralama algoritmaları, bir dizi veriyi belirli bir düzen veya kritere göre sıralamak için kullanılan önemli hesaplama işlemleridir. Dökümanımızın son kısmında geçişli sıralama algoritmalarını inceleyeceğiz. Mülakatlarda junior düzeyde en sık sorulan bubble sort algoritmasının kodunu da inceleyeceğiz. İşte bazı sıralama algoritmaları:

1-)Bubble Sort (Kabarcık Sıralaması): Bu algoritmada iç içe iki döngü vardır ve döngünün dönme sayısı her dönüşte yarıya düşmemektedir. Bu yüzden bu algoritmanın karmaşıklığı O (n ^ 2) karmaşıklığına sahiptir.

2-)Quick Sort (Hızlı Sıralama) : Bu algoritmada da iç içe iki döngü bulunmaktadır. Ancak döngünün dönme sayısı her seferinde yarıya düşmektedir. Bu yüzden bu algoritmanın karmaşıklığı O (n log n) karmaşıklığına sahiptir.

3-)Selection Sort (Seçmeli Sıralama) : Seçmeli sıralama, bir dizideki en büyük veya en küçük elemanı bulup bu elemanı sıralanan elemanların başına veya sonuna yerleştirerek çalışan bir algoritmadır. Bu algoritmada iç içe iki döngü vardır ve döngünün dönme sayısı her dönüşte yarıya düşmemektedir. Bu yüzden bu algoritmanın karmaşıklığı O (n ^ 2) karmaşıklığına sahiptir.

4-)Insertion Sort (Eklemeli Sıralama) : Eklemeli sıralama, bir elemanın uygun konuma eklenmesiyle sıralama yapar. İnsanların kart oyunları sırasında kullandığı bir yaklaşım gibi çalışır. Bu algoritmada iç içe iki döngü vardır ve döngünün dönme sayısı her dönüşte yarıya düşmemektedir. Bu yüzden bu algoritmanın karmaşıklığı O (n ^ 2) karmaşıklığına sahiptir.

5-)Merge Sort (Birleştirme Sıralaması) : Birleştirme sıralaması, bölme ve fethet stratejisini kullanır ve büyük bir problemi küçük alt problemlere böler. Bu alt problemleri sıralar ve sonunda birleştirir. Bu algoritmada da iç içe iki döngü bulunmaktadır. Ancak döngünün dönme sayısı her seferinde yarıya düşmektedir. Bu yüzden bu algoritmanın karmaşıklığı O (n log n) karmaşıklığına sahiptir.

6-)Counting Sort (Sayma Sıralaması) : Sayma sıralama, sayıların sıklığını sayarak çalışır. Genellikle sınırlı aralıkta sayılarla çalışan büyük veriler için uygundur. Bu algoritmada da iç içe iki döngü bulunmaktadır. Ancak döngünün dönme sayısı her seferinde yarıya düşmektedir. Bu yüzden bu algoritmanın karmaşıklığı O (n log n) karmaşıklığına sahiptir.

Örnek Soru : setRandomArray fonksiyonuyla rastgele set edilmiş bir diziyi kabarcık sıralaması algoritması kullanarak sıralayan programı yazınız.

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define randomize() srand(time(NULL))

#define SIZE 20

void printArray(int arr[],int size)

{

for(int i = 0;i < size;++i)

{

printf("%d ",arr[i]);

}

printf("\n");

}

void setRandomArray(int arr[],int size)

{

for(int i = 0;i < size;++i)

{

arr[i] = rand() % 100;

}

}

void bubbleSort(int arr[],int size)

{

for(int i = 0;i < size - 1;++i){

for(int k = 0;k < size - i - 1;++k){

if(arr[k] > arr[k + 1])

{

int temp = arr[k];

arr[k] = arr[k + 1];

arr[k + 1] = temp;

}

}

}

}

int main(void)

{

randomize();

int arr[SIZE];

setRandomArray(arr,SIZE);

printArray(arr,SIZE);

bubbleSort(arr,SIZE);

printArray(arr,SIZE);

}

Dizilerle ilgili detaylar ilerde ele alınacaktır.