COM267

Bölüm 4: Dizeler

C Kullanarak Veri Yapıları, İkinci Baskı
Reema Thareja

- giriiş
- Dizelerde İşlemler
- Dize Dizileri
- İşaretçiler ve Dizeler

giriiş

- Bilgisayarlar, metinsel verilerin oluşturulması, eklenmesi, güncellenmesi ve değiştirilmesi gibi kelime işlem uygulamalarında yaygın olarak kullanılmaktadır.
- Bunun dışında metin içerisinde belirli bir örüntüyü aramamız, silmemiz veya başka bir örüntüyle değiştirmemiz gerekebilir.
- Yani kullanıcılar olarak metinsel verileri manipüle etmek için çok fazla şey yapıyoruz.
- C'de bir dize, null ile sonlandırılmış bir karakter dizisidir.
- Bu, son karakterden sonra karakter dizisinin sonunu belirtmek için boş bir karakterin ('\0') depolandığı anlamına gelir.
- Örneğin, char str[] = "HELLO"; yazarsak, H, E, L, L ve O olmak üzere beş karakterden oluşan bir dizi bildirmiş oluruz.
- Bu karakterlerin dışında dizenin sonunda bir null karakteri ('\0') saklanır.
- Yani, dizenin dahili gösterimi HELLO'\0' olur. 5 uzunluğunda bir dizeyi depolamak için 5 + 1 konuma ihtiyacımız var (boş karakter için 1 ekstra).
- Karakter dizisinin (veya dizenin) adı, dizenin başlangıcına işaret eden bir işaretçidir.

giriiş

- Şekil 4.1 karakter depolama ile dize depolama arasındaki farkı göstermektedir.
- Eğer str'yi char str[5] = "HELLO"; olarak bildirmiş olsaydık, karakter dizisine otomatik olarak boş karakter eklenmeyecekti.
- Bunun sebebi str'nin sadece 5 karakter tutabilmesi ve HELLO'daki karakterlerin kendisine ayrılan alanı zaten doldurmuş olmasıdır.

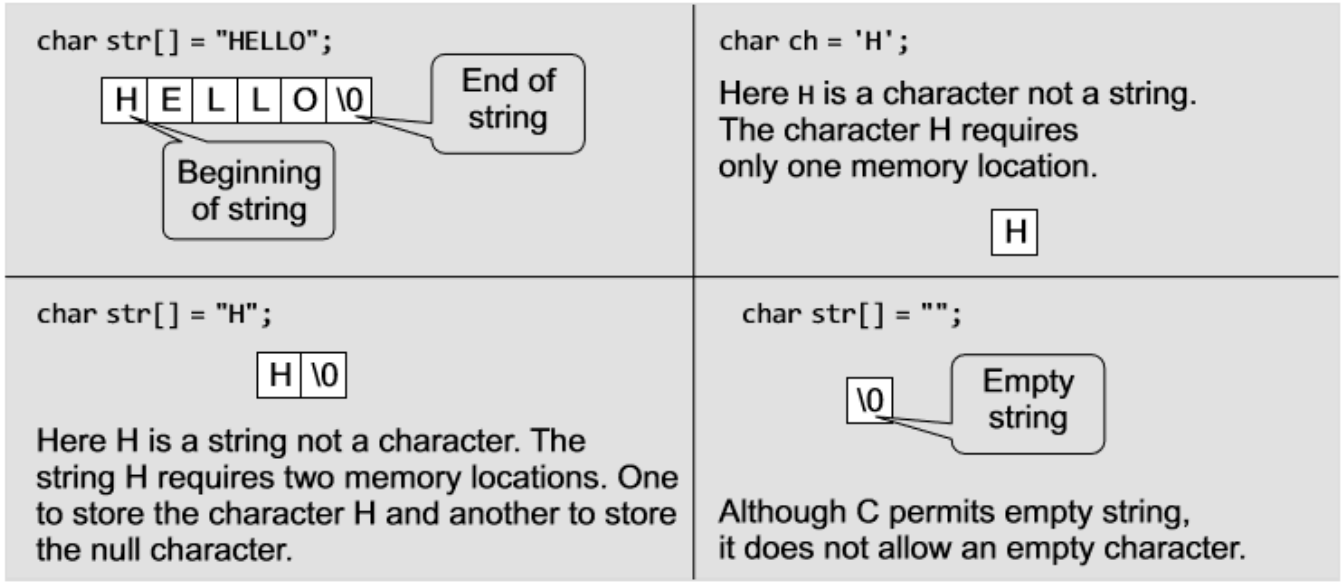

Figure 4.1 Difference between character storage and string storage

giriiş

- Bir dizinin elemanlarına erişmek için dizinleri kullandığımız gibi, bir dizenin elemanlarına erişmek için de dizinleri kullanabiliriz.
- Alt dizin sıfır (0) ile başlar. Bir dizenin tüm karakterleri ardışık bellek konumlarında saklanır.
- Şekil 4.2, str[]'nin bellekte nasıl saklandığını gösterir. Bu nedenle, basit bir ifadeyle, bir dize bir karakter dizisidir.
- Şekil 4.2'de 1000, 1001, 1002 vb., bireysel karakterlerin bellek adresleridir.
- Basitleştirmek için, şekilde H'nin 1000 numaralı bellek konumunda saklandığı gösterilmektedir ancak gerçekte bir karakterin ASCII kodu bellekte saklanır, karakterin kendisi saklanmaz.
- Yani 1000 adresinde H'nin ASCII kodu 72 olduğundan 72 saklanacaktır.

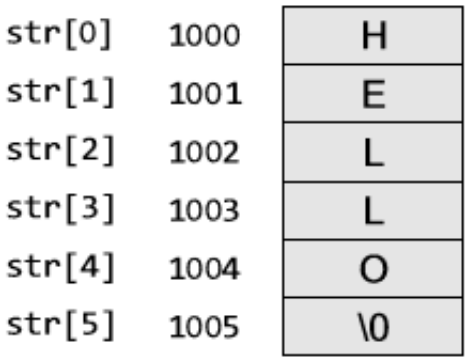

Figure 4.2 Memory representation of a character array

Programming Tip

When allocating memory space for a string, reserve space to hold the null character also.

giriiş

- char str[] = "HELLO"; ifadesi, dizeyi bildirirken ona bir değer atadığımız için sabit bir dize bildirir.
- Ancak, bir dizeyi bildirmenin genel biçimi char str[size] şeklindedir;
- Diziyi bu şekilde bildirdiğimizde, dizide boyut-1 karakter saklayabiliriz çünkü son karakter null karakter olacaktır.
- Örneğin, char mesg[100]; en fazla 99 karakter depolayabilir.
- Şimdiye kadar dizeleri başlatmanın yalnızca bir yolunu gördük.
- Bir dizgeyi başlatmanın diğer yolu onu bir karakter dizisi olarak başlatmaktır.
- Örneğin, char str[] = {'H', 'E', 'L', 'L', 'O', '\0'}; Bu örnekte, boş karakteri açıkça ekledik.
- Ayrıca, dizenin boyutundan bahsetmediğimize dikkat edin.
- Burada derleyici karakter sayısına göre boyutu otomatik olarak hesaplayacaktır.
- Yani bu örnekte, dize değişkeni str'yi depolamak için altı bellek konumu ayrılacaktır.

giriiş

- Ayrıca, başlatılan öğe sayısından çok daha büyük boyutta bir dize de bildirebiliriz. Örneğin, aşağıdaki ifadeyi ele alalım. char str [10] = "HELLO";
- Bu gibi durumlarda derleyici 10 büyüklüğünde bir dizi oluşturur; içine "HELLO" ifadesini depolar ve son olarak dizeyi null karakteriyle sonlandırır.
- Dizideki diğer elemanlar otomatik olarak NULL olarak başlatılır.
- Şimdi aşağıdaki ifadeleri ele alalım: char str[3]; str = "HELLO";
- Yukarıdaki başlatma ifadesi C'de geçersizdir ve iki nedenden dolayı derleme zamanı hatasına neden olur.
- Öncelikle dizi, depolayabileceğinden daha fazla elemanla başlatılı
- İkincisi, başlatma bildirimden ayrılamaz.

giriiş

- **Dizelerin okunması**
- Eğer bir stringi char str[100] yazarak bildirirsek;
- Str kullanıcı tarafından üç şekilde okunabilir:
  1. scanf fonksiyonunu kullanarak,
  2. gets() fonksiyonunu kullanarak ve
  3. getchar() fonksiyonunu tekrar tekrar kullanmak.
- Dizeler scanf() kullanılarak scanf("%s", str); yazılarak okunabilir.

giriiş

- **Dizelerin okunması**
- scanf() fonksiyonunun söz dizimi iyi bilindiği ve kullanımı kolay olduğu halde, bu fonksiyonu kullanmanın en büyük dezavantajı, fonksiyonun boş bir alan bulduğu anda sonlanmasıdır.
- Örneğin kullanıcı Hello World girerse str sadece Hello kelimesini içerecektir.
- Çünkü boşlukla karşılaşıldığı anda string scanf() fonksiyonu tarafından sonlandırılır.
- Ayrıca okunabilecek maksimum karakter sayısını belirtmek için bir alan genişliği de belirtebilirsiniz.
- Giriş tamponunda fazladan karakterlerin tüketilmeden bırakıldığını unutmayın.
- int, float ve char değerlerinin aksine, %s biçimi str değişkeninden önce ve işaretinin kullanılmasını gerektirmez.
- Bir dizeyi okumanın bir diğer yöntemi gets() fonksiyonunu kullanmaktır.
- String gets(str) yazılarak okunabilir; gets(), scanf() fonksiyonunun dezavantajlarını aşan basit bir fonksiyondur.
- gets() fonksiyonu girdiyi tutacak olan dizenin başlangıç adresini alır.
- gets() kullanılarak girilen string otomatik olarak null karakteriyle sonlandırılır.

giriiş

- Dizelerin okunması
- Dizeler ayrıca getchar() fonksiyonunu tekrar tekrar çağırarak tek karakterlerden oluşan bir diziyi okumak (sonlandırıcı bir karakter girilmediği sürece) ve aynı anda aşağıda gösterildiği gibi bir karakter dizisinde saklanmak suretiyle de okunabilir.

```
ben=0;
ch = getchar;// Bir karakter al
while(ch != '*')
{
   str[i] = ch;// Okunan karakteri str'de sakla
   ben++;
   ch = getchar();// Başka bir karakter al
}
str[i] = '\0';// Str'yi null karakterle sonlandır
```

- Bu yöntemde, dizeye bilerek null karakteri eklemeniz gerektiğini unutmayın.
- Diğer iki fonksiyon bunu otomatik olarak yapar.

giriiş

- Dizeleri Yazmak
- Dizeler ekranda aşağıdaki üç şekilde görüntülenebilir:
  1. printf() fonksiyonunu kullanarak,
  2. puts() fonksiyonunu kullanarak ve
  3. putchar() fonksiyonunu tekrar

  tekrar kullanmak.
- Dizeler printf() fonksiyonu kullanılarak printf("%s", str); yazılarak görüntülenebilir.

giriiş

- Bir dize çıktısı almak için %s biçim belirtecini kullanırız.
- Dize değişkeninde '&' karakterinin kullanılmadığına dikkatlice dikkat edin.
- Ayrıca %s ile birlikte genişlik ve hassasiyet özelliklerini de kullanabiliriz.
- Genişlik, minimum çıktı alanı genişliğini belirtir. Dize kısaysa, ekstra alan ya soldan ya da sağdan doldurulur.
- Negatif genişlikte sol, varsayılan sağa hizalama yerine kısa dizeyi destekler.
- Hassasiyet, dizenin kesildiği, görüntülenecek maksimum karakter sayısını belirtir. Örneğin, printf ("%5.3s", str);
- Yukarıdaki ifade, toplam beş karakterlik bir alanda yalnızca ilk üç karakteri yazdıracaktır.
- Ayrıca bu karakterler, ayrılan genişlikte sağa yaslanmış olacaktır.
- Dizeyi sola yaslamak için eksi işareti kullanmalıyız. Örneğin, printf ("%–5.3s", str);

giriiş

- Alan genişliği dizenin uzunluğundan az olduğunda dizenin tamamı yazdırılacaktır.
- Eğer yazdırılacak karakter sayısı sıfır olarak belirtilirse ekrana hiçbir şey yazdırılmaz.
- Bir string yazmanın bir diğer yöntemi ise puts() fonksiyonunu kullanmaktır.
- Bir string, puts(str) yazılarak görüntülenebilir; puts(), printf() fonksiyonunun dezavantajlarını aşan basit bir fonksiyondur.
- Ayrıca, tek karakterlerden oluşan bir diziyi yazdırmak için putchar() fonksiyonunu tekrar tekrar çağırarak da dizeler yazılabilir.

```
ben=0;
str[i] sırasında != '\0')
 {
putchar(str[i]);// Karakteri ekrana yazdır
ben++;
 }
```

Dizelerde İşlemler

- Bu bölümde dizeler üzerinde yapılabilecek farklı işlemleri öğreneceğiz.
- Bir Dizgenin Uzunluğunu Bulma
- Bir dizedeki karakter sayısı dizenin uzunluğunu belirler.
- Örneğin, LENGTH("C PROGRAMLAMA EĞLENCELİDİR") 20 değerini döndürecektir.
- Dizedeki boşlukların bile karakter olarak sayıldığını unutmayın.
- Şekil 4.3'te bir dizenin uzunluğunu hesaplayan bir algoritma gösterilmektedir.
- Bu algoritmada, I, STR dizisini dolaşmak için bir indeks olarak kullanılır.
- STR'nin her bir karakterini dolaşmak için I değerini artırırız.
- Null karakteriyle karşılaştığımızda, kontrol while döngüsünden çıkar ve length değeri I ile başlatılır.
- Not string.h'de tanımlanan strlen(s1) kütüphane fonksiyonu s1 dizisinin uzunluğunu döndürür.

Dizelerde İşlemler

```
Step 1: [INITIALIZE] SET I = 0
Step 2: Repeat Step 3 while STR[I] != NULL
Step 3:    SET I = I + 1
        [END OF LOOP]
Step 4: SET LENGTH = I
Step 5: END
```

Figure 4.3 Algorithm to calculate the length of a string

```
#include <stdio.h>
#include <conio.h>
int main()
{
        char str[100], i = 0, length;
        clrscr();
        printf("\n Enter the string : ");
        gets(str)
        while(str[i] != '\0')
                i++;
        length = i;
        printf("\n The length of the string is : %d", length);
        getch()
        return 0;
}
```

Output

Dizelerde İşlemler

- Bir Dizgenin Karakterlerini Büyük/Küçük Harfe Dönüştürme
- Bellekte gerçek değerler yerine ASCII kodlarının saklandığını daha önce tartışmıştık.
- A–Z'nin ASCII kodu 65 ile 91 arasında değişirken, a–z'nin ASCII kodu 97 ile 123 arasında değişmektedir.
- Yani küçük harfli bir karakteri büyük harfe çevirmek istediğimizde karakterin ASCII değerinden 32'yi çıkarmamız yeterli olacaktır.
- Ve eğer büyük harfli bir karakteri küçük harfe çevirmemiz gerekiyorsa, karakterin ASCII değerine 32 eklememiz gerekir.
- Şekil 4.4, bir dizenin küçük harflerini büyük harfe dönüştüren bir algoritmayı göstermektedir.
- Not: ctype.h'de tanımlanan toupper() ve tolower() kütüphane fonksiyonları bir karakteri sırasıyla büyük ve küçük harfe çevirir.

Dizelerde İşlemler

- Algoritmada I'yi sıfıra başlatıyoruz.
- STR'nin indeksi olarak I'yi kullanarak, STR'nin her bir karakterini 2. Adımdan 3. Adıma kadar geçiyoruz.
- Eğer karakter küçük harfli ise ASCII değerinden 32 çıkarılarak büyük harfe dönüştürülür.
- Ancak karakter zaten büyük harfle yazılmışsa, UPPERSTR dizisine kopyalanır.
- Son olarak, tüm karakterler dolaşıldıktan sonra UPPERSTR'ye boş bir karakter eklenir (Adım 4'te yapıldığı gibi).

```
Step 1: [INITIALIZE] SET I=0
Step 2: Repeat Step 3 while STR[I] != NULL
Step 3:   IF STR[I] >= 'a' AND STR[I] <= '2'
            SET UPPERSTR[I] = STR[I] -32
          ELSE
            SET UPPERSTR[I] = STR[I]
          [END OF IF]
          SET I = I + 1
        [END OF LOOP]
Step 4: SET UPPERSTR[I] = NULL
Step 5: EXIT
```

Figure 4.4 Algorithm to convert characters of a string into upper case

Dizelerde İşlemler

2. Write a program to convert the lower case characters of a string into upper case.

```
#include <stdio.h>
#include <conio.h>
int main()
{
        char str[100], upper_str[100];
        int i=0;
        clrscr();
        printf("\n Enter the string :");
        gets(str);
        while(str[i] != '\0')
        {
                if(str[i]>='a' && str[i]<='z')
                        upper_str[i] = str[i] - 32;
                else
                        upper_str[i] = str[i];
                i++;
        }
        upper_str[i] = '\0';
        printf("\n The string converted into upper case is : ");
        puts(upper_str);
        return 0;
}
```

Output

```
Enter the string : Hello
The string converted into upper case is : HELLO
```

Dizelerde İşlemler

- Bir Dizgeyi Başka Bir Dizeye Ekleme
- Bir dizeyi başka bir dizeye eklemek, kaynak dizenin içeriğinin hedef dizenin sonuna kopyalanmasını içerir.
- Örneğin, S1 ve S2 iki dizeyse, S1'i S2'ye eklemek, S1'in içeriğini S2'ye eklememiz gerektiği anlamına gelir.
- Yani S1 kaynak dizesi, S2 ise hedef dizesidir.
- Ekleme işlemi kaynak dize S1'i değişmeden bırakır ve hedef dize S2 = S2 + S1 olur.
- Şekil 4.5 iki dizeyi birleştiren bir algoritmayı göstermektedir.
- Not: string.h'de tanımlanan strcat(s1, s2) kütüphane fonksiyonu s2 dizesini s1'e birleştirir.

Dizelerde İşlemler

- Bu algoritmada öncelikle hedef dizgeyi dolaşarak sonuna, yani null karakterin bulunduğu konuma ulaşıyoruz.
- Kaynak dizenin karakterleri daha sonra bu konumdan başlanarak hedef dizeye kopyalanır.
- Son olarak hedef dizeyi sonlandırmak için bir null karakteri eklenir.

```
Step 1: [INITIALIZE] SET I=0 and J=0
Step 2: Repeat Step 3 while DEST_STR[I] != NULL
Step 3:         SET I = I + 1
        [END OF LOOP]
Step 4: Repeat Steps 5 to 7 while SOURCE_STR[J] != NULL
Step 5:         DEST_STR[I] = SOURCE_STR[J]
Step 6:         SET I = I + 1
Step 7:         SET J = J + 1
        [END OF LOOP]
Step 8: SET DEST_STR[I] = NULL
Step 9: EXIT
```

Figure 4.5 Algorithm to append a string to another string

Dizelerde İşlemler

```
#include <stdio.h>
#include <conio.h>
int main()
{
        char Dest_Str[100], Source_Str[50];
        int i=0, j=0;
        clrscr();
        printf("\n Enter the source string : ");
        gets(Source_Str);
        printf("\n Enter the destination string : ");
        gets(Dest_Str);
        while(Dest_Str[i] != '\0')
                i++;
        while(Source_Str[j] != '\0')
        {
                Dest_Str[i] = Source_Str[j];
                i++;
                j++;
        }
        Dest_Str[i] = '\0';
        printf("\n After appending, the destination string is : ");
        puts(Dest_Str);
        getch();
        return 0;
}
```

Output

```
Enter the source string : How are you?
```

Dizelerde İşlemler

- İki Dizeyi Karşılaştırma
- Eğer S1 ve S2 iki dizeyse, bu iki dizeyi karşılaştırmak aşağıdaki sonuçlardan birini verecektir:
  (a) S1 ve S2 eşittir
  (b) S1>S2, sözlük sırasına göre S1, S2'den sonra gelir
  (c) S1<S2, sözlük sırasına göre S1, S2'den önce gelir
- İki dizeyi karşılaştırmak için her iki dizedeki her karakter karşılaştırılır.
- Eğer tüm karakterler aynıysa, o zaman iki dizenin eşit olduğu söylenir.
- Şekil 4.6 iki dizeyi karşılaştıran bir algoritmayı göstermektedir.
- Not: string.h'de tanımlanan strcmp(s1, s2) kütüphane fonksiyonu s1 dizesini s2 ile karşılaştırır.

Dizelerde İşlemler

- İki Dizeyi Karşılaştırma
- Bu algoritmada öncelikle iki dizenin aynı uzunlukta olup olmadığını kontrol ediyoruz.
- Aksi takdirde ilerlemenin bir anlamı yoktur, zira bu, iki dizenin aynı olmadığı anlamına gelir.
- Ancak iki dize aynı uzunluktaysa, tüm karakterlerin aynı olup olmadığını kontrol etmek için karakter karakter karşılaştırırız.
- Eğer evet ise, SAME değişkeni 1 olarak ayarlanır. Aksi takdirde, SAME = 0 ise, sözlük sırasındaki hangi dizenin diğerinden önce geldiğini kontrol ederiz ve karşılık gelen mesajı yazdırırız.

Dizelerde İşlemler

```
Step 1: [INITIALIZE] SET I=0, SAME =0
Step 2: SET LEN1 = Length(STR1), LEN2 = Length(STR2)
Step 3: IF LEN1 != LEN2
            Write "Strings Are Not Equal"
        ELSE
            Repeat while I<LEN1
                  IF STR1[I] == STR2[I]
                        SET I = I + 1
                  ELSE
                        Go to Step 4
                  [END OF IF]
            [END OF LOOP]
            IF I = LEN1
                  SET SAME =1
                  Write "Strings are Equal"
            [END OF IF]
Step 4: IF SAME = 0,
            IF STR1[I] > STR2[I]
                  Write "String1 is greater than String2"
            ELSE IF STR1[I] < STR2[I]
                  Write "String2 is greater than String1"
            [END OF IF]
        [END OF IF]
Step 5: EXIT
```

Figure 4.6 Algorithm to compare two strings

Dizelerde İşlemler

- Bir Dizgeyi Tersine Çevirme
- Eğer S1 = "HELLO" ise, S1 = "OLLEH" ifadesinin tersi geçerlidir.
- Bir dizgeyi tersine çevirmek için ilk karakteri son karakterle, ikinci karakteri sondan ikinci karakterle, vb. değiştirmemiz yeterlidir.
- Şekil 4.7 bir dizgeyi tersine çeviren bir algoritmayı göstermektedir.
- Not string.h'de tanımlanan strrev(s1) kütüphane fonksiyonu, dizgedeki null karakteri hariç tüm karakterleri ters çevirir.
- Adım 1'de I sıfıra başlatılır ve J, dizenin uzunluğu olan -1'e başlatılır.
- 2. Adımda, dizenin tüm karakterlerine erişilene kadar while döngüsü yürütülür.
- 4. Adımda STR'nin i'inci karakterini j'inci karakteriyle değiştiriyoruz.
- Sonuç olarak STR'nin ilk karakteri son karakteriyle, ikinci karakteri STR'nin sondan ikinci karakteriyle, vb. değiştirilecektir.
- 4. Adımda, STR'yi ileri ve geri yönlerde sırasıyla geçmek için I değeri artırılır ve J değeri azaltılır.

```
Step 1: [INITIALIZE] SET I=0, J= Length(STR)-1
Step 2: Repeat Steps 3 and 4 while I < J
Step 3:     SWAP(STR(I), STR(J))
Step 4:     SET I = I + 1, J = J - 1
        [END OF LOOP]
Step 5: EXIT
```

Figure 4.7 Algorithm to reverse a string

Dizelerde İşlemler

5. Write a program to reverse a given string.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
        char str[100], reverse_str[100], temp;
        int i=0, j=0;
        clrscr();
        printf("\n Enter the string : ");
        gets(str);
        j=strlen(str)-1;
        while(i<j)
        {
                temp = str[j];

              str[j] = str[i];
              str[i] = temp;
              i++;
              j--;
       }
       printf("\n The reversed string is : ");
       puts(str);
       getch();
       return 0;
}
```

Output

```
Enter the string: Hi there
The reversed string is: ereht iH
```

Dizelerde İşlemler

- Bir Dizgeden Alt Dize Çıkarma
- Verilen bir dizeden bir alt dize çıkarmak için aşağıdaki üç parametreye ihtiyacımız var:
  1. ana dize,
  2. verilen dizedeki alt dizenin ilk karakterinin konumu ve
  3. Alt dizenin maksimum karakter sayısı/uzunluğu.
- Örneğin, str[] = "Programlama dünyasına hoş geldiniz" dizemiz varsa;
- Daha sonra, SUBSTRING(str, 15, 5) = dünya

Dizelerde İşlemler

- Şekil 4.8 bir dizenin ortasından bir alt dize çıkaran bir algoritmayı göstermektedir.
- Bu algoritmada, karakterlerin kopyalanması gereken konum olan I'den M'ye kadar bir döngü sayacı başlatıyoruz.
- 3 ila 6. adımlar N karakter kopyalanana kadar tekrarlanır.
- Kopyalanan her karakterle birlikte N değerini azaltıyoruz.
- Dizenin karakterleri SUBSTR adı verilen başka bir dizeye kopyalanır.
- Son olarak SUBSTR'ye bir null karakteri eklenerek dize sonlandırılır.

```
Step 1: [INITIALIZE] Set I=M, J=0
Step 2: Repeat Steps 3 to 6
           while STR[I] != NULL and N>0
Step 3:  SET SUBSTR[J] = STR[I]
Step 4:  SET I = I + 1
Step 5:  SET J = J + 1
Step 6:  SET N = N - 1
        [END OF LOOP]
Step 7: SET SUBSTR[J] = NULL
Step 8: EXIT
```

Figure 4.8 Algorithm to extract a substring from the middle of a string

Dizelerde İşlemler

- Ana Dizeye Bir Dize Ekleme
- Ekleme işlemi, ana metin T'nin k'ıncı konumuna S dizesini ekler.
- Bu işlemin genel sözdizimi INSERT(metin, konum, dize) şeklindedir.
- Örneğin, INSERT("XYZXYZ", 3, "AAA") = "XYZAAAXYZ"
- Şekil 4.9, belirtilen bir metinde belirtilen konuma bir dize eklemek için bir algoritmayı göstermektedir.
- Bu algoritma ilk önce dizgedeki indeksleri sıfıra başlatır.
- 3. Adımdan 5. Adıma kadar NEW_STR'nin içerikleri oluşturulur.
- Eğer I, alt dizenin eklenmesi gereken konuma tam olarak eşitse, iç döngü alt dizenin içeriğini NEW_STR'a kopyalar.
- Aksi halde metnin içeriği kopyalanır.

Dizelerde İşlemler

```
Step 1: [INITIALIZE] SET I=0, J=0 and K=0
Step 2: Repeat Steps 3 to 4 while TEXT[I] != NULL
Step 3: IF I = pos
          Repeat while Str[K] != NULL
              new_str[J] = Str[K]
              SET J=J+1
              SET K = K+1
          [END OF INNER LOOP]
        ELSE
          new_str[J] = TEXT[I]
          set J = J+1
        [END OF IF]
Step 4: set I = I+1
        [END OF OUTER LOOP]
Step 5: SET new_str[J] = NULL
Step 6: EXIT
```

Figure 4.9 Algorithm to insert a string in a given text at the specified position

Dizelerde İşlemler

- Desen Eşleştirme
- Bu işlem, dize deseninin ilk kez oluştuğu dizgedeki konumu döndürür.
- Örneğin, INDEX("Programlama dünyasına hoş geldiniz", "world") = 15
- Ancak eğer desen dizgede yoksa INDEX fonksiyonu 0 değerini döndürür.
- Şekil 4.10, belirli bir metin içerisinde bir dizenin ilk oluşumunun dizinini bulmak için bir algoritmayı göstermektedir.
- Bu algoritmada MAX, length(TEXT) – Length(STR) + 1 olarak başlatılır.
- Örneğin, bir metinde 'Programlamaya Hoş Geldiniz' yazıyorsa ve string'de 'Dünya' yazıyorsa, ana metinde en fazla 22 – 5 + 1 = 18 karakter ararız, çünkü bundan sonra string'in metinde bulunması için bir alan kalmaz.

Dizelerde İşlemler

```
Step 1: [INITIALIZE] SET I=0 and MAX = Length(TEXT)-Length(STR)+1
Step 2: Repeat Steps 3 to 6 while I < MAX
Step 3:   Repeat Step 4 for K = 0 To Length(STR)
Step 4:          IF STR[K] != TEXT[I + K], then Goto step 6
          [END OF INNER LOOP]
Step 5:   SET INDEX = I. Goto Step 8
Step 6:   SET I = I+1
        [END OF OUTER LOOP]
Step 7: SET INDEX = -1
Step 8: EXIT
```

Figure 4.10 Algorithm to find the index of the first occurrence of a string within a given text

Dizelerde İşlemler

- 3 ila 6. adımlar, metnin her bir karakterinin içinde dizenin bulunup bulunmadığı kontrol edilene kadar tekrarlanır.
- 3. Adımdaki iç döngüde, n karakterlik string değerini n karakterlik text değeriyle kontrol ederek karakterlerin aynı olup olmadığını bulmaya çalışıyoruz.
- Eğer durum böyle değilse o zaman 6. Adıma geçilir ve burada I arttırılır.
- Eğer dize bulunursa, indeks I ile başlatılır, aksi takdirde -1 olarak ayarlanır.
- Örneğin, TEXT = WELCOME TO THE WORLD ise STRING = COME İç döngünün ilk geçişinde, COME ile WELC'i karakter karakter karşılaştıracağız.
- W ve C eşleşmediğinden, kontrol Adım 6'ya geçecek ve ardından ELCO, COME ile karşılaştırılacak. Dördüncü geçişte, COME, COME ile karşılaştırılacak.

Dizelerde İşlemler

- Ana Dizeden Bir Alt Dizeyi Silme
- Silme işlemi, verilen metinden bir alt dizeyi siler.
- Bunu DELETE(metin, konum, uzunluk) olarak yazabiliriz. Örneğin, DELETE("ABCDXXXABCD", 4, 3) = "ABCDABCD"
- Şekil 4.11, verilen bir metinden bir alt dizeyi silmek için bir algoritmayı göstermektedir. Bu algoritmada, önce endeksler sıfıra başlatırız.
- 3'ten 6'ya kadar olan adımlar metnin tüm karakterleri taranana kadar tekrarlanır.
- Eğer I, M'ye (silme işleminin yapılacağı konum) tam olarak eşitse, metnin indeksi artırılır ve N azaltılır.
- N, M pozisyonundan başlayarak silinmesi gereken karakter sayısıdır.
- Ancak eğer I, M'ye eşit değilse, metnin karakterleri basitçe NEW_STR'ye kopyalanır.

Dizelerde İşlemler

```
Step 1: [INITIALIZE] SET I=0 and J=0
Step 2: Repeat Steps 3 to 6 while TEXT[I] != NULL
Step 3:  IF I=M
                Repeat while N>0
                        SET I = I+1
                        SET N = N - 1
                [END OF INNER LOOP]
         [END OF IF]
Step 4: SET NEW_STR[J] = TEXT[I]
Step 5: SET J = J + 1
Step 6: SET I = I + 1
        [END OF OUTER LOOP]
Step 7: SET NEW_STR[J] = NULL
Step 8: EXIT
```

Figure 4.11 Algorithm to delete a substring from a text

Dizelerde İşlemler

- Bir Dizgedeki Deseni Başka Bir Desenle Değiştirme
- Değiştirme işlemi, P1 deseninin başka bir P2 deseniyle değiştirilmesi için kullanılır.
- Bu, REPLACE(text, pattern1, pattern2) yazılarak yapılır. Örneğin, ("AAABBBCCC", "BBB", "X") = AAAXCCC ("AAABBBCCC", "X", "YYY")= AAABBBCC
- İkinci örnekte ise X metinde yer almadığı için bir değişiklik söz konusu değildir.
- Şekil 4.12, metinde bir P1 örüntüsünü başka bir P2 örüntüsüyle değiştirmek için bir algoritmayı göstermektedir.
- Algoritma çok basittir; önce metinde desenin bulunduğu POS konumunu buluyoruz, sonra mevcut deseni o konumdan siliyoruz ve oraya yeni bir desen ekliyoruz.

```
Step 1: [INITIALIZE] SET POS = INDEX(TEXT, P1)
Step 2: SET TEXT = DELETE(TEXT, POS, LENGTH(P1))
Step 3: INSERT(TEXT, POS, P2)
Step 4: EXIT
```

Figure 4.12 Algorithm to replace a pattern P_1 with another pattern P_2 in the text

Dize Dizileri

- Şimdiye kadar bir stringin karakter dizisi olduğunu gördük.
- Örneğin, char name[] = "Mohan" dersek, isim beş karakterden oluşan bir dizedir (karakter dizisi).
- Şimdi, bir sınıfta 20 öğrenci olduğunu ve 20 öğrencinin isimlerini saklayan bir dizeye ihtiyacımız olduğunu varsayalım.
- Bu nasıl yapılabilir? Burada bir string dizisine veya string dizisine ihtiyacımız var.
- Böyle bir dize dizisi 20 ayrı dizeyi depolayabilir.
- Bir dizi dize, char names[20][30] olarak bildirilir;
- Burada ilk indeks kaç adet string'e ihtiyaç olduğunu, ikinci indeks ise her bir string'in uzunluğunu belirleyecektir.
- Burada her bir ismin maksimum 30 karakter uzunluğunda olabileceği 20 isim için yer ayıracağız.

Dize Dizileri

- Dizelerden oluşan bir dizinin bellekteki gösterimine bakalım.
- char name[5][10] = {"Ram", "Mohan", "Shyam", "Hari", "Gopal"} şeklinde tanımlanmış bir dizimiz varsa;
- Daha sonra dizi hafızada Şekil 4.13'te gösterildiği gibi saklanacaktır.

name[0]	R	A	M	'\0'					
name[1]	M	O	H	A	N	'\0'			
name[2]	S	H	Y	A	M	'\0'			
name[3]	H	A	R	I	'\0'				
name[4]	G	O	P	A	L	'\0'			

Figure 4.13 Memory representation of a 2D character array

Dize Dizileri

- Dizi isimlerini tanımlayarak 50 bayt ayırıyoruz.
- Ancak gerçekte işgal edilen bellek 27 bayttır.
- Dolayısıyla ayrılan belleğin yaklaşık yarısının boşa gittiğini görüyoruz.
- Şekil 4.14, bir dizi dizeden bireysel dizeleri işlemek için bir algoritmayı göstermektedir.
- Adım 1'de endeks değişkeni I'i sıfıra başlatıyoruz.
- Adım 2'de dizideki tüm dizelere erişilene kadar while döngüsü yürütülür.
- 3. Adımda her bir dize ayrı ayrı işlenir.

```
Step 1: [INITIALIZE] SET I=0
Step 2: Repeat Step 3 while I< N
Step 3:      Apply Process to NAMES[I]
          [END OF LOOP]
Step 4: EXIT
```

Figure 4.14 Algorithm to process individual string from an array of strings

İşaretçiler ve Dizeler

- C'de dizeler, ikili sıfır karakteriyle ('\0' olarak yazılır) sonlanan karakter dizileri olarak ele alınır.
- Örneğin, char str[10]; str[0] = 'H'; str[1] = 'i'; str[2] = '!': str[3] = '\0';
- C, bir dizeyi bildirmenin ve başlatmanın iki alternatif yolunu sağlar.
- İlk olarak char str[10] = {'H', 'i', '!', '\0'}; yazabilirsiniz.
- Ancak bu da kullanışlı olandan daha fazla yazmayı gerektirir.
- Yani, C char str[10] = "Merhaba!";
- Çift tırnak işareti kullanıldığında dizenin sonuna otomatik olarak boş bir karakter ('\0') eklenir.

İşaretçiler ve Dizeler

- Bir dize bu şekilde bildirildiğinde, derleyici, karakterleri tutmak için belleğin bitişik bir bloğunu (yani 10 bayt uzunluğunda) ayırır ve ilk dört karakterini Hi!\0 olarak başlatır.
- Şimdi, bir metni yazdıran aşağıdaki programı ele alalım.

```
#include <stdio.h>
int ana()
{
   char str[] = "Merhaba";
   karakter *pstr;
   pstr = dizi;
   printf("\n Dize şudur : ");
   while(*pstr != '\0')
   {
      printf("%c", *pstr);
      pstr++;
   }
   0 döndür;
}
```

Çıktı Dize şudur: Merhaba

İşaretçiler ve Dizeler

- Bu programda, ekranda stringi göstermek için *pstr adında bir karakter işaretçisi tanımlıyoruz.
- Daha sonra pstr işaretçisini str'ye yönlendiriyoruz.
- Daha sonra while döngüsünü kullanarak stringin her bir karakterini yazdırıyoruz.
- While döngüsünü kullanmak yerine, aşağıda gösterildiği gibi puts() fonksiyonunu doğrudan kullanabiliriz puts(pstr);
- puts() fonksiyonunun prototipi aşağıdaki gibidir: int puts(const char *s);
- Burada const değiştiricisi, fonksiyonun kaynak işaretçisi tarafından işaret edilen içerikleri değiştirmemesini sağlamak için kullanılır.
- Dizinin adresi fonksiyona argüman olarak

İşaretçiler ve Dizeler

- puts() fonksiyonuna geçirilen parametre, işaret ettiği adresin dışında başka bir şey olmayan bir işaretçi veya sadece bir adrestir.
- Bu nedenle, puts(str) yazmak, str[0] adresini geçirmek anlamına gelir.
- Benzer şekilde puts(pstr); yazdığımızda da aynı adresi geçiriyoruz, çünkü pstr = str; yazdık.
- Bir dizeyi okuyan ve daha sonra girilen büyük ve küçük harf sayısını saymak için her karakteri tarayan başka bir programı düşünün.

İşaretçiler ve Dizeler

```
#include <stdio.h>
int main()
{
        char str[100], *pstr;
        int upper = 0, lower = 0;
        printf("\n Enter the string : ");
        gets(str);
        pstr = str;
        while(*pstr != '\0')
        {
                if(*pstr >= 'A' && *pstr <= 'Z')
                        upper++;
                else if(*pstr >= 'a' && *pstr <= 'z')
                        lower++;
                pstr++;
        }
        printf("\n Total number of upper case characters = %d", upper);
        printf("\n Total number of lower case characters = %d", lower);
        return 0;
}
```

Output

```
Enter the string : How are you
Total number of upper case characters = 1
Total number of lower case characters = 8
```