۱۱ رشته ها در برنامه نویسی C

رشته ها يكي از مهمترين انواع داده ها در زبانهاي برنامه سازي هستند. يك رشته به يك توالي از صفر يا چند كاراكتر گفته مي شود. بعنوان مثال Ali يك رشته كاراكتري است. از رشته ها در موارد بسياري همچون ذخيره اسم و آدرس استفاده مي شود.
در زبان C نوع داده مشخصي براي رشته وجود ندارد، بلكه يك رشته بصورت آرايه اي از كاراكترها تعريف مي گردد. بعنوان مثال:

```
char  name[۱۰];
```

در مثال فوق متغير name بعنوان يك آرايه ۱۰ عضوي از كاراكترها تعريف شده است، بنابراين مي تواند يك رشته با حداكثر ۱۰ كاراكتر را در خود نگاه دارد. اما فرض كنيد قصد داريم رشته اي مانند Ali را در اين متغير ذخيره نماييم كه كمتر از ۱۰ كاراكتر دارد. دراينصورت زبان C چگونه دريابد كه در هنگام انجام عمليات مختلف برروي اين رشته، مثلا در هنگام چاپ آن، فقط بايد ۳ حرف اول رشته را چاپ نمايد؟ براي حل اين مشكل، طراحان زبان C از يك كاراكتر خاص بنام null استفاده كردند. كليه رشته ها در زبان C بايد به كاراكتر null ختم گردند. در حقيقت در زبان C يك رشته هنگامي كه به null برسد، خاتمه مي يابد و نه زمانيكه به انتهاي آرايه برسد. كاراكتر null، داراي كد اسكي ۰ مي باشد و با '\۰' نشان داده مي شود. بعنوان مثال به نمونه زير دقت كنيد:

```
char  name[۱۰] = {'A', 'l', 'i', '\۰' };
```

در مثال فوق، متغير name بصورت آرايه اي ۱۰ عضوي از كاراكتر تعريف شده و مقدار اوليه Ali به آن نسبت داده شده است. دقت كنيد كه در حقيقت ٤ عضو از آرايه پر شده است و عضو آخر به null تخصص داده شده است. مقادير عناصر بعدي آرايه مهم نيست، چرا كه در هنگام انجام عمليات برروي رشته name از آنها صرفنظر مي گردد. بنابراين دقت كنيد كه در هنگام تعريف يك رشته، يك عنصر اضافه براي كاراكتر null در نظر بگيريد.
همانطور كه در فصول قبل گفتيم، در زبان C امكان تعريف ثابت رشته اي نيز وجود دارد. يك ثابت رشته اي، دنباله اي از كاراكترها است كه در داخل " قرار مي گيرد. بعنوان مثال "Ali" نشاندهنده يك ثابت رشته اي است. توجه كنيد كه هنگامي كه از " براي يك ثابت رشته اي استفاده مي كنيد، كامپايلر يك علامت null در انتهاي رشته اضافه مي كند. از ثوابت رشته اي براي مقداردهي اوليه به متغيرهاي رشته اي نيز مي توان استفاده كرد. بعنوان مثال:

```
char  name[۱۰] = "Ahmad" ;
char  address[۵۰] = "No. ۲۰  Azadi Street" ;
```

آرايه name در مثال بالا به اين صورت مقدار مي گيرد:

name	A	h	m	a	d	\۰				

مقادير موجود در ٤ عضو آخر name، هر مقداري مي تواند باشد و از نظر برنامه نويس مهم نيستند، چرا كه توابع رشته اي آنها را پردازش نمي كنند.
نكته آخر اينكه به جز در هنگام مقداردهي اوليه، نمي توانيد در برنامه از عملگر = براي مقداردهي به يك رشته استفاده نماييد. بعنوان مثال دستور زير خطاي نحوي محسوب مي گردد:

```
char  name[۱۰];

name = "Ali" ;
```

فراموش نكنيد كه رشته ها در حقيقت يك آرايه هستند و نمي توان كل يك آرايه را با يك دستور مقداردهي كرد. براي مقداردهي به يك رشته، بايد به تك تك عناصر آن را جداگانه مقدار داد و يا از توابع كتابخانه اي C استفاده كرد.

۱۱-۱ خواندن و نوشتن رشته ها

براي خواندن و نوشتن رشته ها مي توان از توابع scanf و printf استفاده كرد. تنها نكته اين است كه در داخل رشته كنترلي بايد از مشخصه تبديل %s استفاده نماييد. بعنوان مثال به نمونه زير دقت كنيد:

```
#include <stdio.h>

void main() {
  char  name[۲۰];

  printf("what is your name ? ");
  scanf("%s" ,name);
  printf("Hello %s !", name);
}
```

```
what is your name ?  Ali
Hello Ali !
```

همانطور كه مي بينيد در هنگام ارسال متغير name به توابع scanf و printf، تنها نام آن بدون [] استفاده شده است. نكته ديگر اينكه در هنگام ارسال name به تابع scanf از علامت آدرس & استفاده نشده است. در فصلهاي بعدي خواهيد ديد كه نام يك آرايه به تنهايي، برابر با آدرس اولين عنصر آن آرايه است. در نتيجه name به تنهايي خود يك آدرس است.
اما متاسفانه برنامه فوق مشكلي دارد. به اجراي ديگيري از همين برنامه توجه كنيد:

```
what is your name ? Mohamad reza
Hello Mohamad !
```

همانطور كه مي بينيد با وجود اينكه فرد كلمه Mohamad reza را وارد كرده است، اما تابع scanf فقط قسمت اول يعني Mohamad را در متغير name قرار داده است. دليل اين مسئله آن است كه تابع scanf به محض رسيدن به كاراكتر فضاي خالي (space)، گمان مي كند كه رشته خاتمه يافته است و از بقيه آن صرفنظر مي كند. براي رفع اين مشكل مي توان از تابع ديگري بنام gets استفاده كرد. اين تابع يك متغير رشته اي را بعنوان پارامتر ورودي دريافت و پس از خواندن يك رشته از صفحه كليد، آن را در پارامتر ورودي قرار داده و باز مي گرداند. نكته مهم اين است كه تابع gets تا زمانيكه كليد Enter فشرده نشده است، به خواندن داده ها از صفحه كليد ادامه مي دهد. بنابراين رشته مي تواند داراي فضاي خالي (space)نيز باشد. به بازنويسي مثال قبل با تابع gets دقت كنيد:

```
#include <stdio.h>

void main() {
  char  name[۲۰];

  printf("what is your name ? ");
  gets(name);
  printf("Hello %s !", name);
}
```

```
what is your name ?  Mohamad reza
Hello Mohamad reza !
```

لازم به ذكر است كه براي چاپ رشته ها نيز تابعي بنام puts وجود دارد. بعنوان مثال بجاي آخرين تابع printf در مثال بالا مي توانستيد به شكل زير عمل كنيد:

```
puts("Hello ");
puts(name) ;
```

اما معمولا از اين تابع كمتر استفاده مي شود.

۱۱-۲ توابع كتابخانه اي رشته اي

زبان C داراي يك كتابخانه غني از توابع كار با رشته ها است. پيش تعريف اين توابع در فايل سرآمد string.h آمده است. در اين قسمت چندين تابع از كتابخانه C به همراه الگوريتم آنها بررسي مي گردند. هدف از بررسي الگوريتم اين توابع، آشنايي بيشتر شما با نحوه كار با رشته ها مي باشد. دقت كنيد كه ممكن است تعريف دقيق تابع كتابخانه اي در C با آنچه كه ما در اينجا آورده ايم، كمي متفاوت باشد.
نكته بسيار مهمي كه در هنگام بررسي اين توابع بايد بدان توجه داشته باشيد، اين است كه هيچيك از آنها حدود آرايه (اندازه آرايه) را بررسي نمي كنند. بنابراين اين وظيفه خود برنامه نويس است كه آرايه هايي با اندازه مناسب را به توابع ارسال كند.

۱۱-۲-۱ تابع strlen

نام اين تابع مخفف string length مي باشد. اين تابع يك رشته را دريافت كرده و طول آن را باز مي گرداند. مسلما منظور از طول رشته، تعداد كاراكترهاي آن تا رسيدن به null است. تعريف اين تابع در زير آمده است:

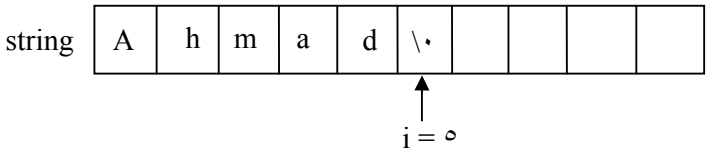
```
int strlen(const char string[]) {
  int i;

  for (i=۰; string[i]; i++) ;
  return(i);
}
```

دقت در تابع بالا نكات جالب را به ما نشان مي دهد. متغير string بصورت ثابت ارسال شده است تا در داخل تابع بطور ناخواسته تغيير داده نشود. دقت كنيد كه ; پس از حلقه for نشان مي دهد كه بدنه اين حلقه، خالي است. به اين معنا كه در هربار اجراي حلقه، فقط عمليات مربوط به خود حلقه يعني افزايش شمارنده و بررسي شرط صورت مي پذيرد. علاوه براين شرط ادامه حلقه فقط string[i] قرار داده شده است كه توسط C اينگونه تفسير مي شود : "تا زمانيكه string[i] درست است". مي دانيد كه هر عدد بجز ۰ درست محسوب مي گردد، بنابراين تا هنگامي كه string[i] برابر null (كه همانطور كه گفته شد برابر كد اسكي ۰ مي باشد) نباشد، درست محسوب مي گردد. به محض اينكه string[i] به null برسد، نادرست ارزيابي شده و حلقه خاتمه خواهد يافت. مسلما در اين حالت مقدار متغير i، طول آرايه را نشان مي دهد. مي توانستيم حلقه for را بصورت زير نيز بنويسيم:

```
for (i=۰; string[i] != '\۰'; i++);
```

براي روشن شدن موضوع به شكل زير دقت كنيد:



همانطور كه مي بينيد، هنگامي كه i برابر ٥ شود، string[i] برابر null شده و شرط برقرار نخواهد بود. در نتيجه مقدار i يعني ٥ نشاندهنده طول رشته است.
برنامه زير نحوه استفاده از strlen را نشان مي دهد:

```
void main() {
  char  text[١٠٠];

  printf("enter a text : ");
  gets(text);
  len = strlen(text);
  printf("length of your text is %d",len);
}
```

```
enter a text : Hello
length of your text is ٥
```

٢-٢-١١ تابع strcpy

نام اين تابع مخفف sring copy مي باشد. اين تابع دو رشته را دريافت و رشته دوم را در رشته اول كپي مي كند. تعريف اين تابع در زير آمده است:

```
void strcpy(char dest[], const char source[]) {
  int i;
  for (i=٠; source[i]; i++)
      dest[i] = source[i] ;
  dest[i] = '\٠' ;
}
```

پارامتر dest، رشته مقصد را نشان مي دهد كه رشته source در آن كپي خواهد شد. از آنجا كه پارامتر source نبايد تغيير كند، بصورت ثابت ارسال شده است. حلقه for تا زمانيكه source[i] به null نرسيده تكرار مي شود و عناصر رشته source را تك به تك در رشته dest كپي مي نمايد. دقت كنيد كه به محض اينكه source[i] به null برسد، حلقه خاتمه مي يابد، در نتيجه خود كاراكتر '\٠' در dest كپي نخواهد شد. به همين دليل در انتها، آخرين عنصر آرايه dest برابر '\٠' قرار داده شده است. برنامه زير نحوه استفاده از اين تابع را نشان مي دهد:

```
void main() {
  char string١[٢٠], string٢[٢٠] ;

  printf("Please enter string١ : ");
  gets(string١) ;

  strcpy(string٢, string١) ;
  printf("copy string١ into string٢\n");
```

```
  printf("now string١ = %s and string٢ = %s\n", string١, string٢);

  strcpy(string١,"new") ;
  printf("copy new into string١\n");
  printf("now string١ = %s", string١);
}
```

```
Please enter string١ : Hello
copy string١ into string٢
now string١ = Hello and string٢ = Hello
copy new into string١
now string١ = new
```

نكته جالبي كه در مثال فوق ديده مي شود اين است كه چنانچه بخواهيم يك ثابت رشته اي را در يك متغير رشته اي قرار دهيم، مي توانيم از تابع strcpy استفاده كنيم.

٣-٢-١١ تابع strcat

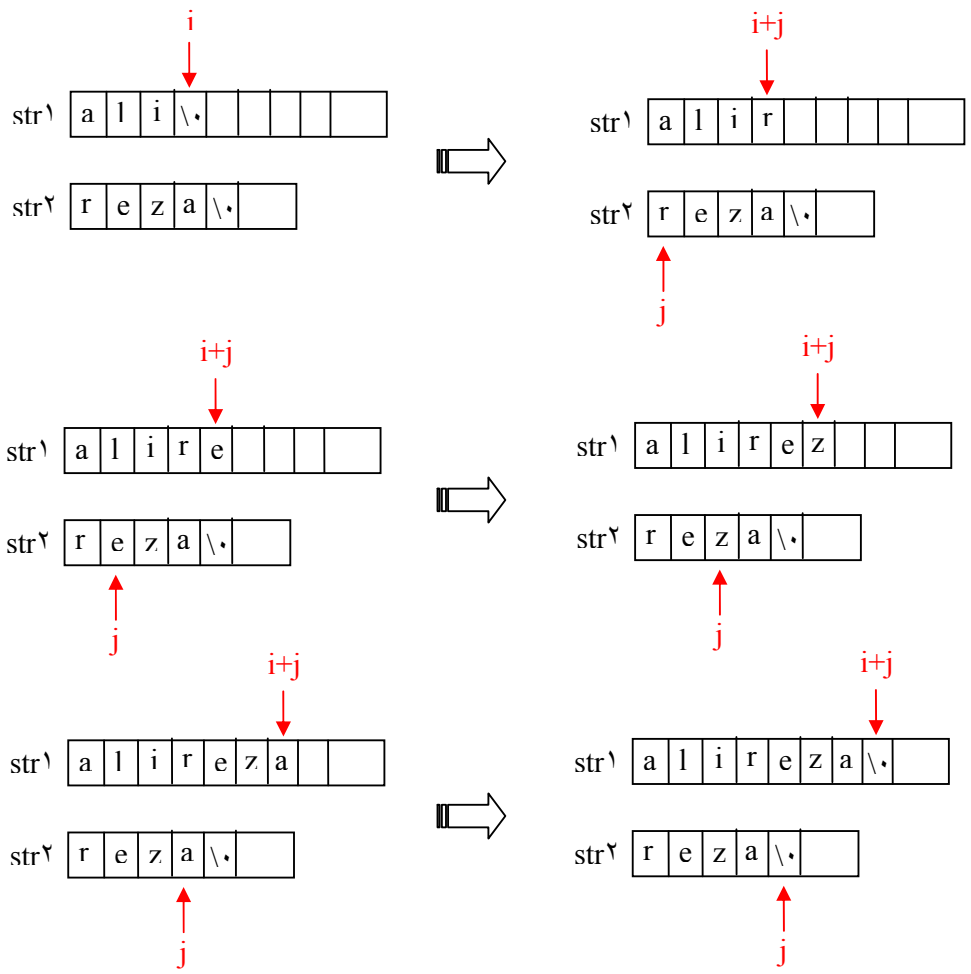
نام اين تابع مخفف string concat است. اين تابع دو رشته را دريافت و رشته دوم را به انتهاي رشته اول الحاق مي كند. به تعريف اين تابع دقت كنيد:

```
void strcat(char str١[], const char str٢[]) {
  int i, j ;

  for (i=٠; str١[i]; i++) ;

  for (j=٠; str٢[j]; j++)
      str١[i+j] = str٢[j] ;
  str١[i+j] = '\٠' ;
}
```

در تابع فوق، قصد داريم رشته str٢ را به انتهاي رشته str١ اضافه كنيم. حلقه for اول، دقيقا همانند تابع strlen است و طول رشته str١ محاسبه و در متغير i قرار مي دهد. سپس در حلقه for دوم، از ابتداي رشته str٢ شروع كرده و هر كاراكتر را به انتهاي رشته str١ اضافه مي كنيم. در پايان كاراكتر '\٠' نيز به انتهاي رشته str١ الحاق مي گردد.
بعنوان مثال فرض كنيد str١="ali" و str٢="reza" باشند. شكل زير مراحل كار حلقه دوم را نشان مي دهد:



براي آشنايي با نحوه فراخواني اين تابع، به برنامه زير دقت كنيد:

```
void main() {
  char string۱[۲۰], string۲[۲۰] ;

  printf("please enter string۱ : ");
  gets(string۱);
  printf("please enter string۲ : ");
  gets(string۲);

  strcat(string۱,string۲);
  printf("concatenate of string۱ and string۲ is : %s", string۱) ;
}
```

```
please enter string۱ : Hello
please enter string۲ : everybody!
concatenate of string۱ and string۲ is : Helo everybody !
```

نام اين تابع مخفف string compare است. اين تابع دو رشته را دريافت و پس از مقايسه آنها يكي از ۳ مقدار زير را باز مي گرداند:

- در صورتيكه مساوي باشند : ۰
- در صورتيكه رشته اول بزرگتر باشد : ۱+
- درصورتيكه رشته اول كوچكتر باشد : ۱-

نحوه مقايسه دو رشته، به همان ترتيبي است كه يك لغتنامه كلمات را مرتب مي كند. يعني ابتدا حروف اول دو رشته مقايسه مي شود، اگر يكي از آنها بزرگتر بود كه نتيجه بازگردانده مي شود. اما در صورتيكه حروف اول دو رشته يكسان بود، حروف دوم با يكديگر مقايسه مي شوند و اين عمل تا زمانيكه يك اختلاف بين دو رشته پيدا شود ادامه مي يابد. در صورتيكه هيچ اختلافي بين دو رشته پيدا نشد، مقدار ۰ باز گردانده مي شود. پياده سازي اين تابع در زير آمده است:

```
int strcmp(const char str۱[], const char str۲[]) {
  int i;

  i = -۱;
  do {
    i ++;
    if (str۱[i] > str۲[i]) return(۱);
    if (str۱[i] < str۲[i]) return(-۱) ;
  } while (str۱[i]) ;
  return(۰);
}
```

تابع فوق دو رشته str۱ و str۲ را با يكديگر مقايسه مي كند. از آنجا كه هيچيك از اين دو نبايد تغيير داده شوند، هردو بصورت ثابت به تابع ارسال شده اند. حلقه do-while عناصر اين دو رشته را با يكديگر مقايسه ميكند. توجه كنيد كه عمل مقايسه براي كاراكترها در زبان تعريف شده و در حقيقت كد اسكي آنها را با يكديگر مقايسه مي كند. در هربار اجراي حلقه، چنانچه يكي از كاراكترها بزرگتر بود، بلافاصله حاصل بازگردانده مي شود، اما درصورتيكه هردو مساوي باشند، حلقه دور زده و عمليات تكرار مي شود. به محض اينكه str[i] به null برسد (كه در اينصورت str۲[i] هم قطعا به null رسيده است، چراكه هر دو مساوي بوده اند) از حلقه خارج شده و مقدار ۰ را به نشانه تساوي دو رشته باز مي گرداند.
براي آشنايي بيشتر به برنامه زير و اجراهاي مختلف آن توجه كنيد:

```
void main() {
  char string۱[۲۰], string۲[۲۰] ;
  int result;

  printf("please enter string۱ : ");
  gets(string۱);
  printf("please enter string۲ : ");
  gets(string۲);

  result = strcmp(string۱,string۲);
  if (result == ۰)
     printf("%s equals %s\n", string۱, string۲) ;
  else if (result == ۱)
          printf("%s is grater than %s\n", string۱, string۲) ;
       else  printf("%s is less than %s\n", string۱, string۲) ;
```

```
}
```

```
please enter string١ : ali
please enter string٢ : ahmad
ali is grater than ahmad
```

```
please enter string١ : ali
please enter string٢ : alireza
ali is less than alireza
```

```
please enter string١ : ali
please enter string٢ : ali
ali equals ali
```

١١-٢-٥ تابع strstr

اين تابع دو رشته را دريافت و در رشته اول به دنبال رشته دوم جستجو مي كند و در صورت پيدا شدن، مكان اولين كاراكتر آن را باز مي گرداند. در صورتيكه در رشته اول چندين نمونه از رشته دوم وجود داشته باشد، مكان اولين نمونه را باز مي گرداند. پياده سازي اين تابع در زير آمده است:

```
void strstr(const char str١[], const char str٢[]) {
  int i, j ;

  for (i=٠; str١[i]; i++)
    if (str١[i] == str٢[٠]) {
      for (j=١; str٢[j] && str١[i+j] == str٢[j] ; j++) ;
      if (!str٢[j]) return(i);
    }

  return(-١) ;
}
```

درك نحوه كار اين تابع نياز به دقت دارد. حلقه for اول تك تك عناصر str١ را با اولين عنصر str٢ مقايسه مي كند. چنانچه يكي از اين عناصر با str٢[٠] برابر بود، آنگاه حلقه for دوم شروع به مقايسه عناصر بعدي str١ (از محل i به بعد) و str٢ (از محل ٠ به بعد) مي نمايد. به شرط حلقه for دوم دقت كنيد. اين حلقه تا زمانيكه str٢ به انتها نرسيده و همچنين عنصر بعدي str٢ و str١ با يكديگر برابر باشند، تكرار مي گردد. دقت كنيد كه بدنه حلقه خالي است. بعد از پايان حلقه بررسي مي گردد كه چنانچه دليل خروج از حلقه به پايان رسيدن str٢ بوده است (يعني str٢[j] برابر null شده است)، بنابراين رشته مورد نظر پيدا شده و مكان شروع آن يعني i بازگردانده مي شود. در غير اينصورت، دليل خروج مساوي نبودن دو كاراكتر از دو رشته بوده است، بنابراين عمليات جستجو ادامه مي يابد. البته تابع فوق بهينه نيست. چرا كه هنگامي كه در رشته اول، شمارنده به مكاني برسد كه تعداد كاراكترهاي باقيمانده تا پايان رشته كمتر از اندازه رشته دوم باشد، قطعا ادامه جستجو لزومي ندارد، چرا كه رشته دوم پيدا نخواهد شد. اعمال اين تغيير بعنوان يك تمرين به خودتان واگذار مي گردد.
براي روشن شدن موضوع، به برنامه زير دقت كنيد:

```
void main() {
  char text[١٠٠], word[٢٠];
  int i, n, result ;
```

```
  printf("enter a text : ");
  gets(text) ;
  printf("how many words do you have : ");
  scanf("%d",&n) ;

  for (i=٠; i<n; i++) {
    printf("enter a word to search : ") ;
    gets(word);
    result = strstr(text, word) ;
    if (result == -١)
       printf("(%s) not found\n",word);
    else printf("(%s) is founded in position %d\n", word, result);
  }
}
```

```
entr a text : this is a sample text!
how many words do you have : ٣
enter a word to search : sample
(sample) is founded in position ١٠
enter a word to search : is
(is) is founded in position ٢
enter a word to search : test
(test) not found
```

٦-٢-١١ تابع strrev

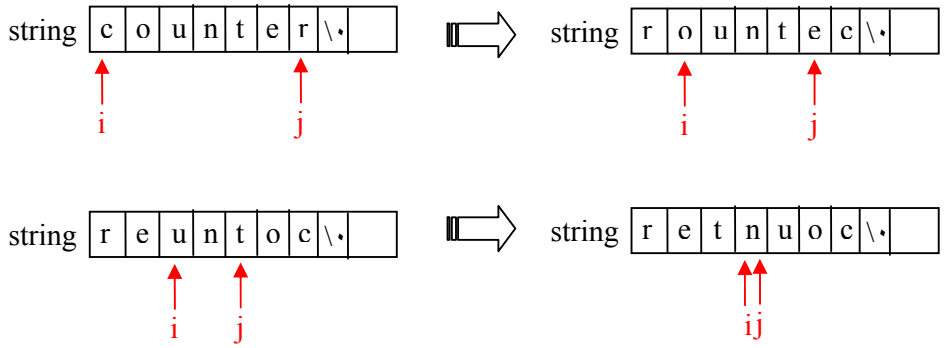
نام اين تابع مخفف string reverse است. اين تابع يك رشته را دريافت و آن را معكوس مي نمايد. پياده سازي اين تابع در زير آمده است.

```
void strrev(char string[]) {
  int i, j, l;
  char temp;

  for (l=٠; string[l]; l++) ;

  for (i=٠, j=l-١; i<j; i++, j--) {
    temp = string[i] ;
    string[i] = string[j] ;
    string[j] = temp ;
  }
}
```

تابع فوق يك رشته را از طريق پارامتر string دريافت و آن را معكوس مي نمايد. حلقه for اول طول رشته را محاسبه و در متغير l قرار مي دهد. سپس در حلقه for دوم، متغير i از ابتداي رشته شروع به حركت رو به جلو، و متغير j از انتهاي رشته شروع به حركت رو به عقب مي نمايند و در حين حركت جاي عناصر متناظر خود را با يكديگر عوض مي كنند. اينكار تا زمانيكه i و j به يكديگر برسند تكرار مي شود. شكل زير نحوه انجام كار را براي رشته "counter" نشان مي دهد.



برنامه زير نحوه كار اين تابع را نشان مي دهد:

```
void main() {
  char word[٢٠] ;

  printf("enter a word : ");
  gets(word) ;
  printf("reverse of %s is ", word);
  strrev(word);
  printf("%s",word) ;
}
```

```
enter a word : hello
reverse of hello is olleh
```

٧-٢-١١ تابع atoi

نام اين تابع مخفف array to integer مي باشد و در فايل سرآمد stdlib.h تعريف شده است. اين تابع يك رشته را دريافت و آن را تبديل به عدد مي كند. البته مسلما رشته بايد منحصرا از ارقام و علامت + يا – تشكيل شده باشد. بعنوان مثال چنانچه **رشته** "٣٤٧٨" را به آن بدهيم، **عدد** ٣٤٧٨ را باز خواهد گرداند. چنانچه رشته ورودي قابل تبديل به عدد نباشد (بدليل وجود كاراكترهاي غيرمجاز مانند حروف)، تابع مقدار ٠ باز خواهد گرداند. پياده سازي اين تابع در زير آمده است:

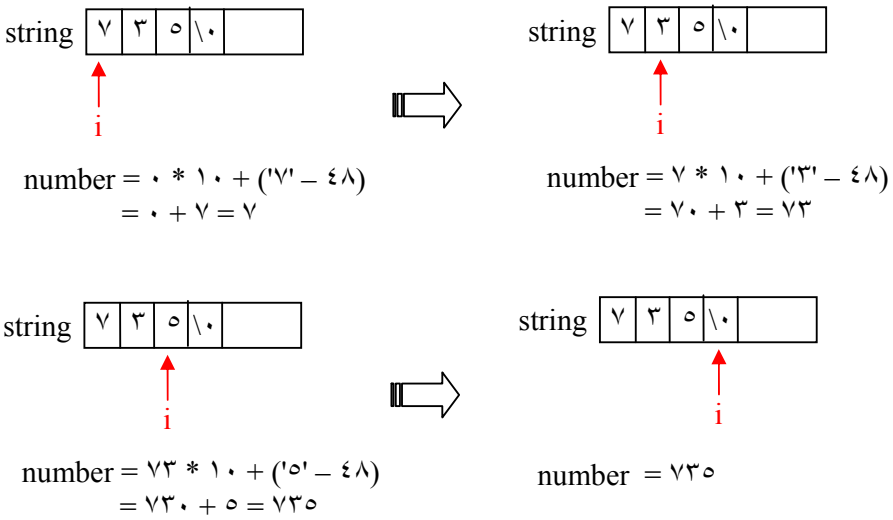
```
int atoi(const char string[]) {
  int i=٠ , sign=١, number=٠;

  if (string[٠] == '-') {
      sign = -١;
      i = ١;
  }
  else if (string[٠] == '+')
         i = ١;

  for (; string[i]; i++)
      if (string[i]>'٠' && string[i]<'٩')
         number = number * ١٠ + (string[i] – ٤٨) ; // or (string[i] – '٠')
      else return(٠);
```

```
  return(sign * number);
}
```

در تابع فوق، string رشته اي است كه قصد تبديل آن به عدد را داريم. متغير i شمارنده رشته است كه از ۰ آغاز مي گردد و sign علامت عدد است كه بطور پيش فرض ۱+ درنظر گرفته مي شود. متغير number نيز مقدار عدد حاصل را در خود نگاه مي دارد. ابتدا بررسي مي شود اگر اولين عنصر آرايه string برابر '-' است، علامت عدد به ۱- تغيير يافته و شمارنده رشته نيز يكي جلو مي رود. چنانچه اولين عنصر برابر '+' باشد، نيازي به تغيير علامت نيست ولي شمارنده رشته بايد جلو برود. سپس در داخل حلقه for، عناصر رشته string به ترتيب بررسي شده و درصورتيكه يك رقم باشند، به حاصل اضافه مي گردند و در غير اينصورت نيز مقدار ۰ بازگردانده مي شود. دقت كنيد كه براي اضافه كردن يك كاراكتر رقمي به number، ابتدا از آن ٤٨، كه كد اسكي كاراكتر ۰ است، را كم مي كنيم تا مقدار عددي آن كاراكتر بدست آيد. بعنوان مثال كد اسكي '۱' برابر ٤٩ است و پس از كم كردن ٤٨ از آن به رقم ۱ مي رسيم. مي توان در برنامه بجاي عدد ٤٨، كاراكتر '۰' را از string[i] كم كرد. در ضمن number در ابتدا در ۱۰ضرب مي شود و سپس با رقم جديد جمع مي شود تا رقم جديد در مكان يكان آن اضافه گردد.شكل زير نحوه انجام كار براي "۷۳٥" نشان مي دهد:

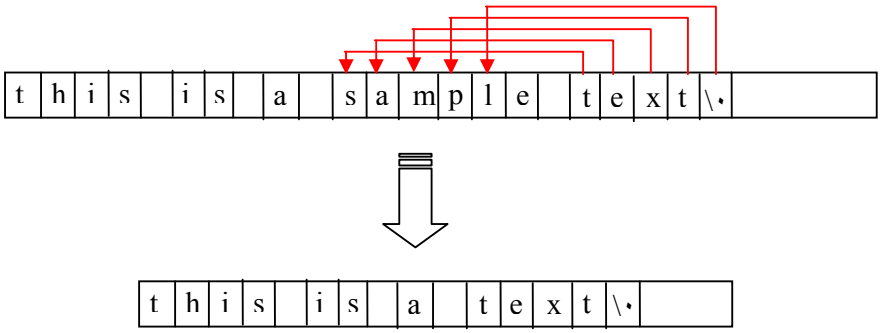


تابع ديگری بنام itoa نيز وجود دارد كه يک عدد را دريافت و آن را تبديل به يک رشته می كند. می توانيد اين تابع را بعنوان تمرين بنويسيد.
از همين دسته توابع، توابع مفيد ديگري همچون atof برای تبديل يک رشته به يک عدد اعشاری و atol برای تبديل يک رشته به يک عدد صحيح بزرگ نيز وجود دارند كه براي آشنايي با آنها مي توانيد به مستندات كامپايلر خود مراجعه كنيد.
توابعي كه مورد بحث قرار گرفتند، ازجمله مهمترين توابعي بودند كه براي كار با رشته ها مورد نياز هستند. اما در فايل سرآمد string.h توابع مفيد ديگري نيز وجود دارند كه براي آشنايي با آنها به مستندات كامپايلر خود رجوع كنيد.

۳-۱۱ چند تابع رشته اي مفيد ديگر

همانطور كه گفته شد، توابع بالا همگي در كتابخانه C موجود هستند و نيازي به نوشتن آنها نيست. اما بعضي اعمال مفيد ديگر نيز بر روي رشته ها وجود دارند كه در توابع كتابخانه اي تعريف نشده اند. در اينجا چند نمونه از اين توابع براي آشنايي بيشتر شما آمده است. در نوشتن اين توابع، از توابع كتابخانه اي استفاده شده است.

مثال ۱) تابعي بنويسيد كه دو رشته را دريافت و در رشته اول بدنبال رشته دوم جستجو كند، در صورت پيدا شدن آن را حذف نمايد.
حل) براي نوشتن اين تابع، ابتدا بايد رشته دوم را در رشته اول پيدا كنيم. براي اين كار مي توان از تابع strstr استفاده كرد. در صورت پيدا شدن رشته دوم، براي حذف آن بايد تمام عناصر پس از آن در رشته اول را به اندازه طول رشته دوم به عقب شيفت دهيم. بعنوان مثال فرض كنيد قصد داريم از رشته "this is a sample text"، رشته "sample " را حذف نماييم. شكل زير نحوه كار را نشان مي دهد. دقت كنيد كه فلشهاي با ارتفاع پايينتر، زودتر انجام شده اند.



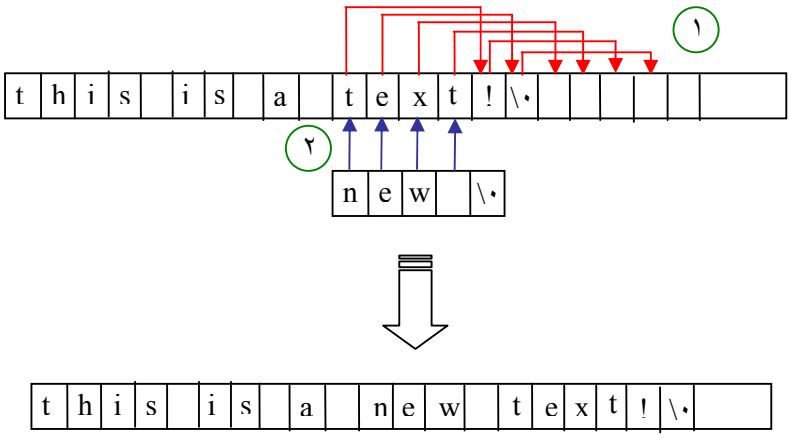
تابع زير اين روال را پياده سازي مي نمايد. توجه نماييد كه مقدار بازگشتي تابع درصورت موفقيت ۱، و در صورت عدم موفقيت (پيدا نشدن رشته دوم در رشته اول) برابر ۰ است.

```
int strDelete(char str۱[], const char str۲[]) {
   int i, position, length;

   position = strstr(str۱, str۲);
   if (position == -۱) return(۰);

   length = strlen(str۲) ;
   for (i= position+length ; str۱[i]; i++)
        str۱[i – length] = str۱[i] ;
   str۱[i-length] = '\۰' ;
   return(۱);
}
```

مثال ۲) تابعي بنويسيد كه دو رشته و عدد k را دريافت و رشته دوم را در مكان kام از رشته اول درج نمايد.
حل) براي حل اين مسئله، ابتدا بايد در مكان k از رشته اول، به اندازه طول رشته دوم فضا باز كنيم. اينكار نياز به شيفت دادن كاراكترها از مكان kام به بعد دارد، اما نكته مهم اينجا است كه بايد عمل شيفت را از انتهاي رشته شروع نماييم. چنانچه شيفت را از مكان kام شروع كنيم، داده موجود در اين مكان برروي يكي از داده هاي ديگر رشته نوشته خواهد شد و باعث ايجاد جواب نادرست خواهد گرديد. سپس رشته دوم را در فضاي ايجاد شده كپي مي نماييم. بعنوان مثال فرض كنيد قصد داريم رشته "new " را در مكان ۱۰ از رشته "this is a text!" درج نماييم. شكل زير نحوه انجام اين كار را نشان مي دهد. دقت كنيد كه فلشهاي با ارتفاع پايينتر، زودتر انجام شده اند.



به پياده سازي اين تابع دقت كنيد:

```
void strInsert(char str١[], const char str٢[], int k) {
  int i, length ;

  length = strlen(str٢);

  for (i=strlen(str١); i >= k ; i --)
     str١[i+length] = str١[i] ;

  for (i=٠ ;i<length; i++)
    str١[k + i] = str٢[i] ;
}
```

در تابع فوق، ابتدا در حلقه for اول عناصر رشته str١ از آخرين عنصر تا عنصر kام هريك به اندازه طول رشته str٢ (كه در متغير length قرار دارد) به سمت جلو شيفت پيدا مي كنند؛ سپس در حلقه for دوم عناصر رشته str٢ تك به تك در مكان kام رشته str١ كپي مي گردند.

مثال ٣) تابعي بنويسيد كه يك رشته و دو عدد start و count را دريافت و سپس يك زير رشته از مكان pام رشته به طول c كاراكتر جدا كرده و بازگرداند.
حل) به تابع زير دقت كنيد:

```
void substring(const char string[], int start, int counr, char result[]) {
  int i;

  for (i=٠; i<count; i++)
    result[i] = string[start + i] ;
  result[i] = '\٠' ;
}
```

پارامتر string رشته اوليه است. پارامتر place مكان شروع زير رشته و count تعداد كاراكترهاي موردنظر است. پارامتر خروجي result نيز زير رشته حاصل مي باشد. همانطور كه مي بينيد با استفاده از يك حلقه for، تك تك عناصر string از مكان place به بعد، در داخل result كپي شده اند.

مثال ٤) تابعي بنويسيد كه يك رشته و يك عدد count را دريافت و سپس يك پيشوند از رشته به طول count را بازگرداند.
حل) به تابع زير دقت كنيد:

```
void prestring(const char string[], int count, char result[]) {
  int i;

  for (i=٠; i<count; i++)
     result[i] = string[i] ;
  result[i] = '\٠' ;
}
```

پارامتر string رشته اوليه است. پارامتر count تعداد كاراكترهاي موردنظر است و پارامتر خروجي result نيز زير رشته حاصل مي باشد.

مثال ٥) تابعي بنويسيد كه يك رشته و يك عدد count را دريافت و سپس يك پسوند از رشته به طول count را بازگرداند.
حل) به تابع زير دقت كنيد:

```
void posttring(const char string[], int count, char result[]) {
  int i ,len, start;

  place = strlen(string) – count ;

  for (i=٠; i<count; i++)
     result[i] = string[start + i] ;
  result[i] = '\٠' ;
}
```

پارامتر string رشته اوليه است. پارامتر count تعداد كاراكترهاي موردنظر است و پارامتر خروجي result نيز زير رشته حاصل مي باشد. مقدار متغير محلي start، مكان شروع عمل كپي را نشان مي دهد كه برابر با مكان انتهاي رشته (طول رشته) منهاي تعداد كاراكترهاي مورد نظر مي باشد.

مثال ٦) تابعي بنويسيد كه سه رشته را دريافت، و سپس در رشته اول به دنبال رشته دوم جستجو نمايد و درصورت پيدا شدن، آن را با رشته سوم جايگزين نمايد.
حل) يك روش ساده حل اين مسئله، به اين صورت است كه ابتدا با استفاده از تابع strDelete، رشته دوم را از رشته اول حذف نموده و سپس با استفاده از strInsert رشته سوم را در رشته اول درج نماييم. به تابع زير دقت كنيد:

```
int  strReplace(char str١[], char str٢[], char str٣[]) {

  position = strstr(str١, str٢) ;
  if (position == -١) return(٠) ;

  strDelete(str١, str٢) ;
  strInsert(str١, str٣, position);
  return(١);
}
```

اما اين روش بهينه نيست، چرا كه در حقيقت عمل حذف داده ها را به عقب شيفت مي دهد و در هنگام درج بايد مجددا داده ها به جلو شيفت داده شوند.
روش بهتر اين است كه پس از پيدا كردن رشته دوم در رشته اول ابتدا تفاضل طول رشته دوم از طول رشته سوم را بدست آوريم و در متغيري مانند difference قرار دهيم. اكنون سه حالت رخ مي دهد:

- چنانچه difference برابر صفر است، بدون نياز به هيچ عملي مي توان براحتي رشته سوم را بر روي رشته دوم كپي كرد.
- چنانچه difference مثبت است، پس طول رشته دوم بيشتر است. در نتيجه داده هاي پس از رشته دوم بايد به اندازه difference به سمت چپ شيفت داده شده و سپس رشته سوم كپي گردد.
- چنانچه difference منفي است، پس طول رشته سوم بيشتر است. در نتيجه داده هاي پس از رشته دوم بايد به اندازه difference به سمت راست شفت داده شده و سپس رشته سوم كپي گردد.

پياده سازي عمليات فوق بعنوان تمرين به خودتان واگذار مي گردد.

مثال ٧) تابعي بنويسيد كه دو رشته را دريافت و تعداد تكرارهاي رشته اول در رشته دوم را پيدا كند.
حل) حل اين مسئله دقيقا مانند strstr است، با اين تفاوت كه پس از پيدا شدن رشته، يك واحد به شمارنده اضافه كرده و سپس جستجو را ادامه مي دهد. به تابع زير دقت نماييد:

```
int strCount(const char str١[], const char str٢[]) {
  int i, j , count=٠ ;

  for (i=٠; str١[i]; i++)
    if (str١[i] == str٢[٠]) {
      for (j=١; str٢[j] && str١[i+j] == str٢[j] ; j++) ;
      if (!str٢[j]) {
        count ++;
        i += (j-١);
      }
    }

  return(count) ;
}
```

تفاوت اين تابع با تابع strstr فقط در دستور زير است:

```
if (!str٢[j]) {
  count ++;
  i += (j-١) ;
}
```

يعني در صورتيكه str٢[j] به null رسيده بود، كه به معناي اين است كه رشته دوم پيدا شده است، ابتدا يك واحد به count اضافه مي كند و سپس متغير i را به اندازه طول رشته str٢ يعني j، جلو مي برد تا در تكرار بعدي حلقه، ادامه رشته str١ بررسي گردد. از ياد نبريد كه در تكرار بعدي حلقه، به i يك واحد اضافه مي شود، در نتيجه آن را به اندازه j-١ واحد جلو برده ايم.

١١-٤ آرايه اى از رشته ها

در بعضي موارد لازم است كه تعدادي رشته كاراكتري را ذخيره نماييم. بعنوان مثال فرض كنيد قصد ذخيره سازي نام ٥ تن از دانشجويان را داريم. اگر براي هر نام ١٠ كاراكتر در نظر بگيريم، احتياج به يك آرايه ٥ تايي از رشته هاي ١٠ كاراكتري خواهيم داشت. براي تعريف آرايه اي از رشته ها، بايد از يك آرايه دو بعدي از كاراكترها استفاده كرد. دستور زير، آرايه مورد نظر براي نگاهداري نام دانشجويان را ايجاد و به آن مقدار اوليه مي دهد:

```
char nameList[٥][١٠] = {"Ali", "Reza", "Ahmad", "Babak", "Hamid" };
```

شكل زير مقادير آرايه فوق را نشان مي دهد:

nameList[٠]	A	l	i	\٠						
nameList[١]	R	e	z	a	\٠					
nameList[٢]	A	h	m	a	d	\٠				
nameList[٣]	B	a	b	a	k	\٠				
nameList[٤]	H	a	m	i	d	\٠				

آرايه nameList

آرايه nameList، يك آرايه دو بعدي از نوع كاراكتر است و همانند تمام آرايه هاي دو بعدي ديگر مي توان از طريق دو انديس به اعضاي آن دسترسي پيدا كرد. بعنوان مثال عنصر nameList[٠][٠] در مثال فوق برابر 'A' مي باشد. اما در يك آرايه از رشته ها، معمولا برنامه نويس عمليات خود را بر روي رشته هاي آن آرايه انجام مي دهد و بطور مستقيم با تك تك كاراكترها كار نمي كند. بعنوان مثال ممكن است بخواهد كه رشته اول يعني "Ali" را چاپ نمايد. براي اينكار كافي است از نام آرايه بعلاوه يك انديس استفاده نماييم. بعنوان مثال nameList[٠] يك متغير رشته اي است كه مقدار "Ali" را در خود جاي داده است. براي دسترسي به ساير رشته ها نيز كافي است از نام آرايه بعلاوه انديس رديفي كه رشته در آن قرار دارد، استفاده نماييم. حتي مي توان با اين روش، يك رشته را به يك تابع نيز ارسال كرد. بعنوان مثال دستور زير مقداري را براي سومين رشته از كاربر دريافت مي نمايد :

```
gets(nameList[٢]) ;
```

براي نمونه به مثال زير دقت كنيد:
مثال) برنامه اي بنويسيد كه ابتدا شماره دانشجويي، نام، معدل تعدادي دانشجو را دريافت و ذخيره نمايد. سپس با دريافت شماره دانشجويي هر فرد به دنبال اطلاعات وي جستجو كرده و در صورت پيدا شدن، مشخصات وي را چاپ نمايد. برنامه با در يافت شماره دانشجويي ٠ پايان مي يابد.

```
#include <stdio.h>

const int maxStudent = ١٠٠;
const int maxNameLen = ٢٠;

void main() {
  long int idList[maxStudent] ;
  float averageList[maxStudent] ;
  char nameList[maxStudent][maxNameLength] ;
  long int searchId;
  int i, n;


  printf("enter student number : ");
  scanf("%d",&n);
  for (i=٠; i<n; i++) {
      printf("student #%d :\n", i+١) ;
      printf("enter id : ");
```

```
        scanf("%ld", &idList[i]);
        printf("enter name : ");
        gets(nameList[i]);
        printf("enter average : ");
        scanf("%f", &averageList[i]);
    }

    // now searching
    while(١) {
      printf("enter student id (٠ to exit) : ");
      scanf("%ld", &searchId);
      if (!serachId) break ;

      for (i=٠ ; i<n; i++)
         if (idList[i] == searchId) break;

       if (i<n) {
          printf("Student specifications : \n");
          printf("id=%ld  name=%s  average=%f\n", idList[i], nameList[i],
                                                    averageList[i]);
       }
       else printf("student not found !\n");
    }
}
```

در برنامه فوق، ابتدا در حلقه for اول مشخصات دانشجويان دريافت مي گردد. سپس در حلقه while، عمليات جستجو آغاز مي گردد. دقت كنيد كه شرط تكرار حلقه بصورت while(١) نوشته شده است. مي دانيد كه ١ در زبان C به معناي درست است، بنابراين حلقه تا ابد تكرار خواهد شد. اما در داخل حلقه و پس از دريافت شماره دانشجويي (searchId)، مقدار آن بررسي شده و در صورت ٠ بودن، با استفاده از دستور break از حلقه خارج مي شود. اين كار توسط دستور زير انجام مي شود:

```
if (!searchId) break;
```

توجه كنيد كه چنانچه مقدار searchId برابر ٠ (نادرست) باشد، بنابراين نقيض آن درست خواهد بود و شرط برقرار مي گردد.
اين عمل، يعني ايجاد يك حلقه بي نهايت توسط while(١) و سپس خروج از حلقه توسط دستور break در صورت بروز شرايط خاص، عملي متداول در برنامه نويسي است. بخصوص در مواديكه شرط خروج از حلقه در وسط آن بررسي مي گردد.
پس از دريافت شماره دانشجويي، در آرايه idList به دنبال آن جستجو كرده و در صورت پيدا شدن، مقادير متناظر با آن در آرايه nameList و averageList چاپ مي گردند.

سعيد ابريشمي
www.prdev.com