مبانی کامپیوتر و برنامه نویسی به زبان C

فصل هشتم: پردازش رشته ها

۸-۱ مقدمه

نیاز به پردازش کاراکترها و رشته ها در حل پاره ای از مسائل برنامه نویسی اجتناب ناپذیر است،

۸-۲ رشته های کاراکتر

در زبان C کد عددی معادل هر کاراکتر را می توان در یک متغیر از نوع صحیح ذخیره نمود

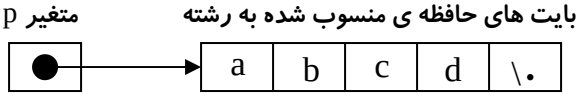
```
x = 'a';
```

کد معادل حرف a در متغیر x ذخیره می گردد و اکنون اگر دستور ; x++ اجرا شود محتویات x که حرف b خواهد بود.

۸-۲-۱ ثابت های رشته ای

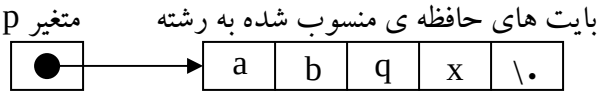
احضار توابع printrf و scanf و کاربرد رشته ها در آنها

در اثر دستور ;"abcd" = *p char در پنج بایت متوالی حافظه نخست کاراکترهای a, b, c و d و بعد از آنها کاراکتر \۰ ذخیره شده و آدرس کاراکتر a در داخل اشاره گر p قرار می گیرد که این وضعیت را می توان به صورت زیر تجسم نمود.



رشته ای که ایجاد می شود در حقیقت آرایه ای از نوع char است کـه آدرس مبنـای آن در متغیـر p قـرار دارد بعد از اجرای دستورهای p[۲] = 'q'; و *(p+۳) = 'x'; وضعیت فوق به صورت زیر تبدیل خواهد شد.

در نمونه های زیر دو دستور سمت راستی با دستورهای سمت چپی معادل می باشند.




```
char  *s = " Total = ٪۳ d \ n";
printf(s, t );
```

```
printf ("Total = ٪۳ d \ n", t) ;
```

```
s [g] =' ۵ ' ;
printf(s, r );
```

```
printf (" Total = ٪۵ d \ n", r) ;
```

۸-۲-۲ تعریف رشته ها در حافظه

برای ذخیره رشته ای با طول n ، از آرایه ای از نوع char با n+۱ خانه استفاده می شود.

در ضمن حفظ سادگی زبان C ، متغیرهای از نوع رشته ی کاراکتر با به کارگیری نوع های مبنایی زبان شبیه سازی شده است.

عمل های مورد نیاز برای پردازش رشته ها از طریق توابع انجام می شود.

توابع مزبور علاوه بر آدرس شروع باید طول رشته را نیز داشته باشند، این طـول را از روی نشـانه ی پایـان رشته که همان کاراکتر \۰ است در می یابند.

در صورت ذخیره ی رشته ای با طول بیش از حد مجاز تعریف شده برای آرایه، به دلیل این که این موضوع توسط کامپایلر C کنترل نمی گردد، نتیجه کار قابل پیش بینی نیست.

```
char test۱[۵۰], test۲[۱۰], test۳[۷] = {'s', 'a', 'm', 'p', 'l', 'e',
'\۰'};
char line۱[۱۰] = {'a', 'b', 'c', '\۰'}, line۲[] = {'a', 'b', 'c', '\۰'};
char str۱[۷] = "sample", str۲[] = "A sample string., str۳[۲۰] = "A test!";
char error۱[۳] = {'a', 'b', 'c'}, error۲[] = {'w', 'r', 'o', 'n', 'g'};

char error۳[۱۰] = "this is an error!", error۴[۱۴] = "Another Error!";
```

۸-۳ عملیات ورودی و خروجی برای رشته ها

۸-۳-۱: خواندن و نوشتن کاراکتر

تابع getchar که به صورت int getchar () تعریف شده و فاقد پارامتر است. برای خواندن یک کاراکتر در صورتی که به پایان داده های ورودی برخورد نماید، مقداری مساوی ثابت نمادی EOF برخواهد گرداند

تابع دیگر putchar نام دارد که تعریف آن به صورتی مشابه int putchar (int c) بوده و پارامتر ورودی آن کد عددی معادل یک کاراکتر می باشد که در خروجی چاپ می شود.

اگر متغیر صحیح n حاوی عدد ۹۷ که کد معادل حرف a است، باشد احضارهای putchar (n);، putchar (n+۲۵۶); و putchar (n+۱۰۲۴); هر سه باعث نمایش حرف a در خروجی خواهد شد (چرا؟). در صورت ارسال کد عددی معادل علائم کنترلی مثل صدای زنگ (کد ۷)، شروع صفحه (کد ۱۲) یا شروع سطر (کد ۱۰)، اثر کنترلی مربوطه در خروجی ظاهر می گردد.

دو تابع بالا پایه ای ترین توابع برای انجام عملیات ورودی و خروجی هستند.

مثال:

```
#include <stdio.h>
main ()
{ int  c;
  While ((c = getchar ()) != EOF)
  if (c >= 'a'  &&  c  <=  'z')
  putchar (c + 'A'  -  'a' );
  else
  putchar (c);
  return  ٠;
}
```

برنامه نمونه ۸-۱: استخراج و چاپ آمار کاراکترها

```
#include <stdio.h>

main()                          /* برنامهٔ شمارش تعداد دفعات تکرار کاراکترهای مختلف در یک متن */
{  int ndigit[۱۰], nletter[۲۶], nwhite = ۰, nother = ۰, c, i, j;

   for (i = ۰; i < ۲۶; i++)              /* حلقهٔ تکرار صفر کردن آرایه‌های شمارنده */
   {  if (i < ۱۰)
         ndigit[i] = ۰;
      nletter[i] = ۰;
   }
   while ((c = getchar()) != EOF)        /* حلقهٔ تکرار خواندن کاراکترهای متن ورودی */
   {  putchar(c);
      if (c >= 'a' && c <= 'z')          /* شمارش حروف کوچک */
         nletter[c - 'a']++;
      else if (c >= 'A' && c <= 'Z') /* شمارش حروف بزرگ */
         nletter[c - 'A']++;
      else if (c >= '۰' && c <= '۹') /* شمارش رقمها */
         ndigit[c - '۰']++;
      else if (c == ' ' || c == '\n' || c == '\t' || c == '\f')
         nwhite++;                       /* شمارش فاصله‌های سفید */
      else
         nother++;                       /* شمارش سایر کاراکترها */
   }
                                         /* چاپ عنوان گزارش آماری */
   printf
      ("\f Frequency of occurrences of characters in a text.\n");
   printf("\nCharacter Occurrences \n--------------------\n");
                                         /* حلقهٔ تکرار چاپ آمار حروف کوچک و بزرگ */
   for (i = 'a'; i <= 'z'; i++)
      printf("  %c or %c      %۳d\n",
             i, i+'A'-'a', nletter[i-'a']);
                                         /* حلقهٔ تکرار چاپ آمار رقمها */
   for (i = '۰'; i <= '۹'; i++)
      printf("    %c          %۳d\n", i, ndigit[i - '۰']);
                                         /* چاپ آمار فاصله‌های سفید */
   printf("White spaces  %۳d\n", nwhite);
                                         /* چاپ آمار سایر کاراکترها */
   printf("Other chars   %۳d\n", nother);
   return ۰;
}
```

شکل ۸-۱۰: متن برنامهٔ ۸-۱، استخراج و چاپ آمار کاراکترهای مختلف در یک متن.

برنامه نمونه ۸-۲: تبدیل یک عدد از حالت رشته ای به معادل عددی

```
#include <stdio.h>

main()                                  /* برنامهٔ تبدیل یک عدد از حالت رشته‌ای به حالت عددی */
{  char c, strnum[۱۱];
   long number = ۰;
   int i = ۰, length, sign;

   while ((c = getchar()) == ' ') /* حلقهٔ تکرار خواندن و صرف‌نظر کردن از فواصل خالی */
   ;
   strnum[۰] = c;                       /* ذخیرهٔ اولین کاراکتر غیرفاصلهٔ خالی */
   i = ۱;
   /* حلقهٔ تکرار خواندن ادامهٔ رشتهٔ ورودی و تعیین طول آن */
   while ((c = getchar()) != '\n')
   {  strnum[i] = c;
   i++;
   }
   length = i;
   sign = strnum[۰] == '-' ? -۱ : ۱;/* بررسی وجود علامت منفی قبل از رقمهای عدد */
   /* صرف‌نظر کردن از علامت در صورت وجود */
   i = (strnum[۰] == '-' || strnum[۰] == '+') ? ۱ : ۰;
   /* بررسی رقم بودن کاراکترها و تبدیل آنها به ارزش عددی معادل */
   for (; i < length && strnum[i] >= '۰' && strnum[i] <= '۹';   i++)
   number = number * ۱۰ + (strnum[i] - '۰');
   printf(" The input string is: ");/* چاپ رشتهٔ ورودی */
   for (i = ۰; i < length; i++)
   putchar(strnum[i]);
   /* چاپ معادل عددی محاسبه شده در برنامه */
   printf("\n The corresponding number is: %ld\n", sign * number);
   return ۰;
}
```

شکل ۸-۱۱: متن برنامهٔ ۸-۲، تبدیل یک عدد از حالت رشته‌ای به معادل عددی.

۸-۳-۲ : خواندن و نوشتن رشته ها

دو تابع دیگر با نام های gets و puts وجود دارد که اولی به شکل Char *gets (char *s) تعریف شده است. این تابع یک سطر کامل را می خواند و به جای کاراکتر پایان سطر یعنی \n کاراکتر پایان رشته یا \۰ قرار می دهد.

تعداد خانه های تعریف شده برای آرایه ی s باید برای خواندن سطر ورودی و علامت پایان رشته کافی باشد.

در صورت برخورد به پایان فایل ورودی یا اشکال در خواندن، مقدار NULL را برمی گرداند.

تابع دوم int puts (const char *s) است که سطر کامل داده شده را چاپ می کند و به جای علامت پایان رشته نیز کارکتر \n ارسال می کند.

اگر در حین انجام عمل خروجی با خطا روبه رو شود مقدار EOF را به عنوان نتیجه برمی گرداند و در صورت انجام عمل مزبور نتیجه ی آن یک مقدار غیر منفی خواهد بود.

دلیل این که قبل از پارامتر آن کلمه ی const گذاشته شده ؟

این دو تابع که خودشان با استفاده از توابع getchar و putchar نوشته شده اند، در نوشتن توابع scanf و printf استفاده شده است.

برنامه نمونه ی ۸-۳:

برای خواندن رشته از دو حلقه داخل هم استفاده شده، حلقه اول برای خواندن و چاپ سطرها است و در حلقه دوم کاراکترهای موجود دو سطر خوانده شده مورد شمارش قرار می گیرد. تفاوت دیگر هم این است که چون تابع gets علامت پایان سطر را به علامت پایان رشته تبدیل می کند این علامت شروع صفحه از فاصله های سفید حذف شده اند. در سایر موارد دو برنامه مشابه می باشند.

```
#include <stdio.h>
#define LENGTH ۸۱                          /* تعریف حداکثر طول خط */

       /* برنامهٔ شمارش تعداد دفعات تکرار کاراکترهای مختلف در یک متن بااستفاده از خواندن و نوشتن رشته */
main()
{  int ndigit[۱۰], nletter[۲۶], nwhite = ۰, nother = ۰, i, j;
   char line[LENGTH];

   for (i = ۰; i < ۲۶; i++)              /* حلقهٔ تکرار صفر کردن آرایه‌های شمارنده */
   {  if (i < ۱۰)
         ndigit[i] = ۰;
      nletter[i] = ۰;
   }
   while (gets(line) != NULL)            /* حلقهٔ تکرار خواندن سطرهای متن ورودی */
   {  puts(line);
      nwhite++;                          /* شمارش علائم پایان سطر */
                                         /* حلقهٔ تکرار بررسی کاراکترهای موجود در یک سطر */
      for (i = ۰; i < LENGTH - ۱ && line[i] != '\۰'; i++)
         if (line[i] >= 'a' && line[i] <= 'z')
            nletter[line[i] - 'a']++;    /* شمارش حروف کوچک */
         else if (line[i] >= 'A' && line[i] <= 'Z')
            nletter[line[i] - 'A']++;    /* شمارش حروف بزرگ */
         else if (line[i] >= '۰' && line[i] <= '۹')
            ndigit[line[i] - '۰']++;     /* شمارش رقمها */
         else if
            (line[i] == ' ' || line[i] == '\t' || line[i] == '\f')
            nwhite++;                    /* شمارش فاصله‌های سفید */
         else
            nother++;                    /* شمارش سایر کاراکترها */
   }
                                         /* چاپ عنوان گزارش آماری */
   printf
      ("\f Frequency of occurrences of characters in a text.\n");
   printf("\nCharacter Occurrences\n--------------------\n");
                                         /* حلقهٔ تکرار چاپ آمار حروف کوچک و بزرگ */
   for (i = 'a'; i <= 'z'; i++)
      printf("  %c or %c      ٪۳d\n",
             i, i + 'A' - 'a', nletter[i - 'a']);
                                         /* حلقهٔ تکرار چاپ آمار رقمها */
   for (i = '۰'; i <= '۹'; i++)
      printf("    %c         ٪۳d\n", i, ndigit[i - '۰']);
                                         /* چاپ آمار فاصله‌های سفید */
   printf("White spaces  ٪۳d\n", nwhite);
                                         /* چاپ آمار سایر کاراکترها */
   printf("Other chars   ٪۳d\n", nother);
   return ۰;
}
```

شکل ۸-۱۲: متن برنامهٔ ۸-۳، استخراج و چاپ آمار کاراکترهای متن از طریق خواندن و نوشتن رشته.

۸-۳-۳ : خواندن رشته ها با فرمت

جدول ۸-۱ : نمونه هایی از خواندن رشته ها از طریق مشخصه ی %s به داخل آرایـه هـای x[۶] و y[۴] از نوع char .

(علامت های < و > در دو طرف رشته ی ورودی فقـط بـرای تعیـین دقیـق آن اسـت و بخشـی از داده ی ورودی نیست).

ردیف	آرگومان های تابع scanf	داده ی ورودی	محتویات x و y	توضیح
۱	("%s" , x)	<abcde>	x:abcde\۰	رشته ی خوانده شده قابل ذخیره است.
۲	("%s" , x)	<ab>	x:ab\۰	رشته ی خوانده شده قابل ذخیره است.
۳	("%s" , x)	<ab cde>	x:ab\۰	فاصله ی خالی جدا کننده می باشد.
۴	("%s" , x)	<abcdefgh>	غیر قابل پیش بینی	طول رشته بزرگتر از گنجایش آرایه.
۵	("%s" , x)	<"abc">	x:"abc"\۰	رشته ی خوانده شده قابل ذخیره است.
۶	("\"%[^\"]\"", x)	<"a b d">	x:a b\nd\۰	علامتهای "\ در فرمت با " در ورودی تطبیـق یافتـه کـل رشته ی بین آن ها از جملـه فاصـله خـالی و شـروع سـطر خوانده می شود
۷	("%s", &x[۱])	<ab>	x:?ab\۰	رشته ی خوانده شده از خانه ی یکم ذخیـره شـده اسـت، خانه ی صفرم تغییر نمی کند.
۸	("%s%s", x, y)	<ab cde>	x:ab\۰ y:cde\۰	فاصله ی خالی جدا کننده می باشد.
۹	('٪۴s٪۳s",x,y)	<abcdefgh>	x:abcd\۰ y:efg\۰	به تعداد طول میدان از رشـته ی ورودی بـرای هـر آرایـه خوانده و ذخیره شده است.
۱۰	("٪۶s٪۳s",x,y)	<abcdefgh>	غیر قابل پیش بینی	در آرایه ی X حداکثر ۵ کاراکتر جا می گیرد.

۸-۳-۴ : نوشتن رشته ها با فرمت

جدول ۸-۲: تأثیر ترکیبات گوناگون طول میدان، دقت و تعداد کاراکتر در نتیجه ی چاپ رشته ها با فرمت

ردیف	تأثیر در نتیجه ی چاپ	ترکیبات طول، طول میدان و دقت
۱	رشته ی ارائه شده عیناً چاپ می شود.	تعداد کاراکتر= دقت= طول میدان
۲	طول میدان بی اثر است و رشته ی ارائه شده عیناً چاپ می شود.	تعداد کاراکتر= دقت< طول میدان
۳	به تعداد طول میدان منهای دقت در سمت مخالف تنظیم فاصله خالی اضافه شده و نتیجه چاپ می شود.	تعداد کاراکتر= دقت< طول میدان
۴	به تعداد دقت کاراکترها از سمت چپ رشته ی ارائه شده برداشته شده و چاپ می شود.	تعداد کاراکتر< دقت= طول میدان
۵	طول میدان بی اثر است و عین مورد قبلی عمل می شود.	تعداد کاراکتر< دقت< طول میدان
۶	به تعداد دقت کاراکترها از سمت چپ رشته ی ارائه شده برداشته شده، به تعداد طول میدان منهای دقت در سمت مخالف تنظیم چاپ فاصله ی خالی اضافه و چاپ می شود.	تعداد کاراکتر< دقت< طول میدان
۷	دقت بی اثر است و به تعداد طول میدان منهای تعداد کاراکترهای ارائه شده در سمت مخالف تنظیم چاپ فاصله ی خالی اضافه و چاپ می شود.	تعداد کاراکتر< دقت= طول میدان
۸	طول میدان و دقت بی اثرند و رشته ی ارائه شده عیناً چاپ می شود.	تعداد کاراکتر< دقت،دقت< طول میدان و تعداد کاراکتر≤طول میدان
۹	دقت بی اثر است و به تعداد طول میدان منهای تعداد کاراکترهای ارائه شده در سمت مخالف تنظیم چاپ فاصله ی خالی اضافه شده و نتیجه چاپ می شود.	تعداد کاراکتر< طول میدان < دقت
۱۰	دقت بی اثر است و به تعداد طول میدان منهای تعداد کاراکترهای ارائه شده در سمت مخالف تنظیم چاپ فاصله ی خالی اضافه شده و نتیجه چاپ می شود.	تعداد کاراکتر< دقت < طول میدان

به عنوان مثال آرایه ی `x` **مورد استفاده در جدول ۸-۳ را می توان به هر دو صورت زیر تعریف نمود. در صورت استفاده از هر کدام از این تعاریف نیاز به هیچگونه تغییری در آرگومانهای ذکر شده برای تابع** `printf` **نخواهد بود.**

```
char  x[۱۱] = "sun & moon";
```

```
char  *x = "sun & moon";
```

جدول ۸-۳: نمونه هایی از چاپ رشته ها از داخل آرایه ی x[۱۱] کـه حـاوی رشـته ی "sun & moon" می باشد، متغیر i حاوی ۱۲ است (علامتهای < و> اطراف رشته ی خروجی فقط برای تعیین دقیق آن بوده و جزء خروجی نیست).

آرگومان های تابع printf	نتیجه ی چاپ	توضیح
("%s", x)	<sun & moon>	رشته ی داده شده عیناً چاپ می‌شود.
("٪۱۰s", x)	<sun & moon>	طول رشته داده شده با طول میدان برابر است، رشته عیناً چاپ می‌شود.
("٪۱۲s", x)	< sun & moon>	طول رشته داده شده دوتا کمتر از طول میدان است پس رشته با دو فاصله خالی اضافی در سمت چپ چاپ می‌شود.
("%*s",i , x)	< sun & moon>	مشابه بالا ولی با طول میدان به صورت آرگومان (i قبل از x).
("٪-۱۲s", x)	<sun & moon >	مشابه مورد بالا ولی به دلیل تنظیم از چپ دو فاصله ی خالی در سمت راست اضافه می‌شود.
("٪۵s", x)	<sun & moon>	طول رشته داده شده بزرگتر از طول میدان است پس بدون توجه به طول میدان، رشته عیناً چاپ می‌شود.
("٪۵s", &x[۲])	<n & moon>	عین سطر قبل با این تفاوت که شروع رشته از خانه ی دوم است.
("٪۲c٪-۳c", x[۰], x[۹])	< sun >	دو کاراکتر از رشته انتخاب شده و با توجه به مشخصه های تبدیل متناظر چاپ شده است.
("٪۱۰.۱۰s", x)	<sun & moon>	نمونه ای از ردیف ۱ جدول ۲-۸ .
("٪۸.۱۰s", x)	<sun & moon>	نمونه ای از ردیف ۲ جدول ۲.
("٪۱۲.۱۰s", x)	< sun & moon>	نمونه ای از ردیف ۳ جدول ۲.
("٪۸.۸s", x)	<sun & mo>	نمونه ای از ردیف ۴ جدول ۲.
("٪۶.۸s", x)	<sun & mo>	نمونه ای از ردیف ۵ جدول ۲.
("٪۱۰.۸s", x)	< sun & mo>	نمونه ای از ردیف ۶ جدول ۲.
("٪-۱۰.۸s", x)	<sun & mo >	مشابه مورد بالا ولی تنظیم از چپ.
("٪۱۲.۱۲s", x)	< sun & moon>	نمونه ای از ردیف ۷ جدول ۲-۸ .
("٪۸.۱۲s", x)	<sun & moon>	نمونه ای از ردیف ۸ جدول ۲-۸ .
("٪-۱۲.۱۴s", x)	<sun & moon >	نمونه ای از ردیف ۹ جدول ۲-۸ . با تنظیم از چپ.
("٪۱۲.۱۱s", x)	< sun & moon>	نمونه ای از ردیف ۱۰ جدول ۲-۸ .
("%*.۱۱s",۱۲, x)	< sun & moon>	مشابه بالا ولی با طول میدان به صورت آرگومان(۱۲ قبل از x).
("%*.*s"i,۱۱, x)	< sun & moon>	مشابه مورد بالا ولی با طول میدان و دقت به صورت آرگومان.
("%.۸s", x)	<sun & mo>	طول میدان حذف شده، به تعداد دقت کاراکترها چاپ می‌شود.
("%.۱۲s", x)	<sun & moon>	طول میدان حذف شده و طول رشته > دقت، کل رشته چاپ می‌شود.

۸-۳-۵ : خواندن با فرمت از داخل یک رشته و نوشتن با فرمت به داخل یک رشته

```
int sprintf (char *string, char *format, exp۱, exp۲, . . . );

int sscanf (char *string, char *format, adr۱, adr۲, . . . );

int day, month, year;
char line[۸۰] , monstr[۱۵];
gets (line);
if (sscanf (line, "%d/%d/%d", &day, &month, &year) == ۳)
     printf ("valid date is: %s\n", line);
else if (sscanf (line, "%d %d %d", &day, &month, &year) == ۳)
     printf ("valid date is: %s\n", line);
else if (sscanf (line, "%d %s %d", &day, &monstr, &year) == ۳)
     printf ("valid date is: %s\n", line);
else printf ("Invalid date: %s\n", line);
```

۸-۴ : عملیات روی رشته ها

۸-۴-۱ جا به جا کردن رشته ها (توابع strcpy و strncpy)

```
char s۱[] = "This is a test .", s۲[۲۰], *s۳ = "This is another test.",
*t;
s۲ = s۱;                    /* دستور غلط است */

s۳ = s۱;        /*دستور درست است ولی آدرس شروع آرایه سمت راست  به متغیر سمت چپ تخصیص داده می شود */


for (i = ۰; i <= ۱۵; i ++)          /* رشته در اثر اجرای حلقه ی تکرار جابه جا می شود */
       s۲[i] = s۱[i];

for (i = ۰, t = s۳; i < ۱۵; i ++, t ++)          /*رشته جا به جا می شود */
      *t = s۱ [i];
```

```
char *strcpy (char *s, const char *t)
char *strncpy (char *s, const char *t, unsigned n )
```

```
char *strcpy(char *s, const char *t)   /* روش اندیس‌دهی */
{  int i = ۰;
   while ((s[i] = t[i]) != '\۰')        /* حلقهٔ تکرار برای بازنویسی رشته */
      i++;
   return s;                            /* برگرداندن آدرس رشتهٔ حاصل */
}

char *strcpy(char *s, const char *t)   /* روش محاسبات روی اشاره‌گرها */
{  char *ts;
   ts = s;
   while ((*s = *t) != '\۰')            /* حلقهٔ تکرار برای بازنویسی رشته */
   {  s++;
      t++;
   }
   return ts;                           /* برگرداندن آدرس رشتهٔ حاصل */
}

char *strncpy(char *s, const char *t, unsigned n)/* روش اندیس‌دهی */
{  unsigned i;
   for (i = ۰; i < n && t[i] != '\۰'; i++)
      s[i] = t[i];                      /* حلقهٔ تکرار برای بازنویسی بخشی از رشته */
   for (; i < n; i++)
      s[i] = '\۰';      /* در صورت نیاز NULL حلقهٔ تکرار برای پرکردن بایتهای خالی با */
   return s;                            /* برگرداندن آدرس رشتهٔ حاصل */
}

char *strncpy                           /* روش محاسبات روی اشاره‌گرها */
   (char *s, const char *t, unsigned n)
{  unsigned i;
   char *ts;
   ts = s;
   for (i = ۰; i < n && *t != '\۰'; i++, s++, t++)
      *s = *t;                          /* حلقهٔ تکرار برای بازنویسی بخشی از رشته */
   for (; i < n; i++, s++)
      *s = '\۰';        /* در صورت نیاز NULL حلقهٔ تکرار برای پرکردن بایتهای خالی با */
   return ts;                           /* برگرداندن آدرس رشتهٔ حاصل */
}
```

شکل ۸-۲: متن توابع strcpy و strncpy برای جابه‌جا کردن رشته‌ها.

```
char  str۱ [۱۲] =  "take  that ", *str۲ = "bring this", str۳ [ ] = "pq";
int  n =  ٤;
strcpy (str۱, str۲);
strcpy (str۲, "not this");
strncpy (str۱, str۲, n);
strncpy (str۱, str۳, n);
strncpy (str۲, str۱, n + ۱);
```

قبل از اولین احضار تابع strcpy:

str۱:	t	a	k	e		t	h	a	t	\۰	\۰	\۰
str۲:	b	r	i	n	g		t	h	i	s	\۰	
str۳:	p	q	\۰									

بعد از اولین احضار تابع:

str۱:	b	r	i	n	g		t	h	i	s	\۰	\۰
str۲:	b	r	i	n	g		t	h	i	s	\۰	

بعد از دومین احضار تابع:

str۱:	b	r	i	n	g		t	h	i	s	\۰	\۰
str۲:	n	o	t		t	h	i	s	\۰	s	\۰	

بعد از سومین احضار تابع:

str۱:	n	o	t		g		t	h	i	s	\۰	\۰
str۲:	n	o	t		t	h	i	s	\۰	s	\۰	

بعد از چهارمین احضار تابع:

str۱:	p	q	\۰	\۰	g		t	h	i	s	\۰	\۰
str۳:	p	q	\۰									

بعد از پنجمین احضار تابع:

Str۱:	p	q	\۰	\۰	g		t	h	i	s	\۰	\۰
Str۲:	p	q	\۰	\۰	\۰	h	i	s	\۰	s	\۰	

شکل ۸-۱ : نمونه هایی از احضار توابع مربوط به جابه جایی رشته ها.

```
void *memcpy (void *a, const void *b,  unsingned n)
```

۸-۴-۲ تعیین طول رشته ها (تابع strlen)

```
unsigned strlen (const char *s)

char s۱[۱۰],  *s۲ = "Is this a test?";

strcpy (s۱, "yes !");

printf ("%u, %u\n", strlen (s۱), strlen (s۲));
```

۴،۱۵ نتیجه ی چاپ:

برنامه های نمونه ی ۸-۴ و ۸-۵

متن تابع strlen

```
unsingned long strlen (const char *s)
{  unsingned long n ;
   for (n = ۰; *s != '\۰'; ++s)
   ++n;
   return n;
}
```

۸-۴-۳ مقایسه ی رشته ها (توابع strncmp و strcmp)

```
int strcmp (const char *s, cost char *t);
```

- اگر دو رشته دقیقاً هم طول بوده و کاراکترهای متناظر مساوی باشند جواب صفر بر می گرداند.

- **اگر یک رشته کوتاه تر باشد و کاراکترهای آن با کاراکترهای متناظر در رشته ی دیگر مساوی باشند رشته ی کوتاه تر کوچک تر فرض می شود. حال اگر رشته ی اول یعنی s کوتاه تر باشد یک عدد منفی وگرنه یک عدد مثبت به عنوان جواب برگردانده می شود.**
- **اگر صرف نظر از طول دو رشته (هم طول یا غیر هم طول) دو کاراکتر (شماره ی p ام) متناظر در دو رشته نامساوی باشند و کاراکتر p ام رشته ی اول در ردیف کاراکترهای جدول کدهای مورد استفاده (مثل ASCII یا EBCDIC) قبل از کاراکتر p ام از رشته ی دوم باشد یک عدد منفی به عنوان جواب برمی گرداند. نهایتاً اگر کاراکتر p ام رشته ی اول در ردیف کاراکترهای جدول کدها بعد از کاراکتر p ام از رشته ی دوم باشد یک عدد مثبت به عنوان جواب بر می گرداند.**

تابع دیگر

```
int strncmp (const char *s, const char *t, unsingned n);
```

۸-۴-۴ چسباندن یا الحاق دو رشته به یکدیگر (توابع strcat و strncat)

تابع دیگر

```
char *strcat (char *s, const char *t)
char *strncat (char *s, const char *t, unsigned n)
```

```
char str۱[۱۲] = "take", *str۲ = " this", str۳[۱۵] = "that is it";
int n = ۴;
strcat  (str۱, str۲);
strcat  (str۱, " x");
strncat (str۳, str۱,  n);
strcpy  (str۱, "buy");
```

قبل از اولین احضار تابع strcat :

str۱:	t	a	k	e	\.	\.	\.	\.	\.	\.	\.	\.
str۲:		t	h	i	s	\.						
str۳:	t	h	a	t		i	s		i	t	\.	\.	\.	\.	\.

بعد از اولین احضار تابع strcat :

str۱:	t	a	k	e		t	h	i	s	\.	\.	\.
str۲:		t	h	i	s	\.						

بعد از دومین احضار تابع strcat :

str۱:	t	a	k	e		t	h	i	s		x	\.

بعد از اولین احضار تابع strncat :

str۳:	t	h	a	t		i	s		i	t	t	a	k	e	\.
str۱:	t	a	k	e		t	h	i	s		x	\.			

بعد از احضار تابع strcpy :

شکل۸-۶: نمونه هایی از احضار توابع مربوط به الحاق رشته ها به یکدیگر.

```
int memcmp (const void *a, const void *b, unsingned n)
```

۸-۴-۵ جستجوی یک رشته یا کاراکتر در یک رشته ی دیگر (توابع strstr، strchr و strrchr)

```
#include <stdio.h>
#include <string.h>

main()
{  char nm۱[] = “mirzahamid”, nm۲[] = “abdolhosein”, *sbs =
“hose”;

   printf(“۱: string in string: %s\n”, strstr(nm۲, sbs));
   printf(“۲: string has char (first): %s\n”, strchr(nm۱, ‘i’));
   printf(“۳: string has char (last): %s\n”, strrchr(nm۱, ‘i’));
   if (strstr(nm۲, “sin”) != NULL)
      printf(“۴: string in string: %s\n”, strstr(nm۲, “sin”));
   else
      printf(“۵: substring not found!\n”);
   return ۰;
}
```

نتیجهٔ خروجی:

```
۱: string in string: hosein
۲: string has char (first): irzahamid
۳: string has char (last): id
۵: substring not found!
```

شکل ۸-۷: نمونه‌هایی از جستجوی زیررشته یا کاراکتر در رشته‌ها.

```
void  *memchr (const  void  *s ,  int  c,  unsigned  n)
```

۸-۴-۶ تبدیل رشته به عدد (توابع atoi ، atol و atof)

```
int  atoi (const  char  *s)
long  atol (const  char  *s)
double  atof (const  char  *s)
```

متن این تابع atoi به عنوان بخشی از برنامه ی شماره ۸-۵ در بخش برنامه های نمونه ارائه گردیده است.

۸-۴-۷ تشخیص گروه بندی کاراکترها

تابع `isalnum (c)` **برای بررسی این که کاراکتر ارائه شده یک حرف یا یک رقم است.**

تابع `isalpha (c)` **برای بررسی این که کاراکتر ارائه شده یک حرف است.**

تابع `iscntrl (c)` **برای بررسی این که کاراکتر ارائه شده یک کاراکتر کنترل است.**

تابع `isdigit (c)` **برای بررسی این که کاراکتر ارائه شده یک رقم مبنای ده است.**

تابع `isgraph (c)` **برای بررسی این که کاراکتر ارائه شده یک کاراکتر قابل چاپ (بجز فاصله ی خالی) است.**

تابع `islower (c)` **برای بررسی این که کاراکتر ارائه شده یک حرف کوچک انگلیسی است.**

تابع `isprint (c)` **برای بررسی این که کاراکتر ارائه شده یک کاراکتر قابل چاپ یا یک فاصله ی خالی است.**

تابع `ispunct (c)` **برای بررسی این که کاراکتر ارائه شده یک کاراکتر قابل چاپ غیر از حرف، رقم یا فاصله ی خالی (یک جدا کننده) است.**

تابع `isspace (c)` **برای بررسی این که کاراکتر ارائه شده یک کاراکتر فاصله ی سفید است.**

تابع `isupper (c)` **برای بررسی این که کاراکتر ارائه شده یک حرف بزرگ انگلیسی است.**

تابع `isxdigit (c)` **برای بررسی این که کاراکتر ارائه شده یک رقم مبنای شانزده است.**

```
int   tolower (int  c);
int   toupper (int  c);
```

۸-۵ برنامه های نمونه

```
#include <stdio.h>
#include <string.h>
#define MAXLEN ۱۰۰

void reverse(char *s, char *t)  /* تابعی برای معکوس کردن ترتیب کاراکترهای یک رشته */
{  int i, j;
   for (i = strlen(s) - ۱, j = ۰; i >= ۰; i--, j++)
      t[j] = s[i];  /* حلقهٔ تکرار بازنویسی کاراکترهای رشتهٔ اصلی به طور معکوس در پارامتر دوم */
   t[j] = '\۰';      /* اضافه نمودن علامت پایان رشته به آخر رشتهٔ معکوس شده */
   return;
}

main()               /* تابع اصلی برای خواندن رشته‌ها و چاپ هر رشته همراه با معکوس آن */
{  char text[MAXLEN], revtext[MAXLEN];
   int done = ۰;
   while (!done)                              /* حلقهٔ تکرار خواندن رشته‌ها */
   {  printf("\n\nEnter a string:\t");        /* اعلام درخواست ورود رشته */
      if (gets(text) != NULL)                 /* بررسی پایان رشته‌های ورودی */
      {  reverse(text, revtext);              /* احضار تابع برای معکوس کردن رشته */
         printf("Original text: %s\nReversed text: %s\n",
                text, revtext);               /* چاپ رشتهٔ ورودی و معکوس آن */
      }
      else
         done = ۱;                            /* اعلام پایان رشته‌های ورودی */
   }
   return ۰;
}
```

شکل ۸-۱۳: متن برنامهٔ ۸-۴، خواندن رشته‌ها و چاپ هر رشته همراه با معکوس آن.

```
#include <stdio.h>
#include <string.h>
#define MAXLEN ۱۲

char *itoa(long n, char *s)       /* تابعی برای تبدیل یک عدد صحیح به یک رشتهٔ معادل */
{  int i, len;
   long sign;
   char tmp;

   n = ((sign = n) < ۰) ? -n : n;    /* ذخیرهٔ عدد اولیه و تبدیل عدد منفی به مثبت */
   i = ۰;
   do                                /* حلقهٔ تکرار جداسازی رقمها از راست به چپ */
   {  s[i] = n % ۱۰ + '۰'; /* تبدیل هر رقم جدا شده به کاراکتر معادل و ذخیره در رشتهٔ جواب */
      i++;
   } while ((n /= ۱۰) > ۰);          /* بررسی تمام شدن رقمها و خاتمهٔ حلقهٔ تکرار */
   if (sign < ۰)                     /* درج علامت در صورت نیاز */
      s[i++] = '-';
   s[i] = '\۰';
   len = strlen(s);
                                     /* حلقهٔ تکرار برعکس کردن ترتیب رقمهای جدا شده */
   for (i = ۰; i < len / ۲; i++)
   {                                 /* تعویض جای یک کاراکتر از اول رشته با یک کاراکتر از آخر رشته */
       tmp = s[i];
      s[i] = s[len - i - ۱];
      s[len - i - ۱] = tmp;
   }
   return s;
}


main()                    /* تابع اصلی برای خواندن عددها و چاپ هر عدد همراه با رشتهٔ معادل آن */
{  char text[MAXLEN];
   int done = ۰;
   long number;
```

```
#include <stdio.h>
#include <string.h>              /* افزودن فایل حاوی تعریف توابع لازم برای پردازش رشته‌ها */
#define MAXNOS ۱۰۰               /* حداکثر تعداد دانشجویان */
#define LINEPPAGE ۲۵             /* حداکثر تعداد سطر مورد چاپ در هر صفحه */

main()                           /* برنامهٔ مرتب‌سازی داده‌های دانشجویان برحسب فامیل و نام */
{ char fams[MAXNOS][۳۱];         /* آرایه برای ذخیره و جابه‌جایی فامیلها */
  char names[MAXNOS][۲۱], tmp[۳۱];/* آرایه برای ذخیره و جابه‌جایی اسامی و آرایهٔ کمکی */
  long stno[MAXNOS], tsn;        /* آرایه و متغیر کمکی برای ذخیره و جابه‌جایی شماره‌های دانشجویی */
  float gpa[MAXNOS], tgp;        /* آرایه و متغیر کمکی برای ذخیره و جابه‌جایی معدلها */
  int i, j, nofst;

  scanf("%d", &nofst);
  for (i = ۰; i < nofst; i++)    /* حلقهٔ تکرار خواندن داده‌ها */
    scanf("٪ld٪۲۰s٪۳۰s٪۵f", &stno[i], names[i], fams[i], &gpa[i]);
  for (i = ۰; i < nofst - ۱; i++) /* حلقه‌های تکرار مرتب‌سازی داده‌ها */
    for (j = i + ۱; j < nofst; j++)
      if (strcmp(fams[i], fams[j]) > ۰ || /* احضار تابع مقایسهٔ رشته‌ها */
          strcmp(fams[i], fams[j]) == ۰ &&
          strcmp(names[i], names[j]) > ۰)
      { tsn = stno[i];           /* جابه‌جا کردن دو شمارهٔ دانشجویی */
        stno[i] = stno[j];
        stno[j] = tsn;
        strcpy(tmp, names[i]);   /* جابه‌جا کردن دو سطر آرایهٔ حاوی دو اسم */
        strcpy(names[i], names[j]);
        strcpy(names[j], tmp);
        strcpy(tmp, fams[i]);    /* جابه‌جا کردن دو سطر آرایهٔ حاوی دو فامیل */
        strcpy(fams[i], fams[j]);
        strcpy(fams[j], tmp);
        tgp = gpa[i];            /* جابه‌جا کردن دو معدل */
        gpa[i] = gpa[j];
        gpa[j] = tgp;
      }
  for (i = ۰; i < nofst; i++)    /* حلقهٔ تکرار چاپ داده‌های مرتب شده */
  { if (!(i % LINEPPAGE))        /* چاپ عنوان و شرح ستونها برای هر صفحه */
```

برنامه ی ۸-۷ : تابعی بنویسید که یک آرایه ی دو بعدی حاوی اسامی حداکثر ۳۰ کاراکتری و یک اسم حداکثر ۳۰ کاراکتری را به عنوان پارامتر گرفته و سپس با فرض اینکه اسامی موجود در آرایه به صورت صعودی مرتب شده باشد، با استفاده از روش جستجوی دوتایی وجود یا عدم وجود اسم داده شده را در بین اسامی موجود در آرایه بررسی نموده در صورت وجود، شماره ی سطر حاوی آن و در غیر این صورت مقدار ۱- را به عنوان نتیجه برگرداند. این تابع پارامتر سومی نیز دارد که تعداد اسمهای موجود در آرایه را نشان می دهد.

```
#include <string.h>
#define MAXLEN ۳۱
#define FAIL   -۱

       /* تابعی برای بررسی وجود یا عدم وجود یک اسم در آرایه‌ای از اسامی به روش جستجوی دوتایی */
int binary_search(char name[], char name_list[][MAXLEN], int rows)
{  int first = ۰, last = rows - ۱, mid, res;

   while (first <= last)      /* حلقهٔ تکرار جستجو تا زمانی که به نصفهٔ تهی برسیم */
   {  mid = (first + last) / ۲;  /* محاسبهٔ شماره سطر وسط */
      res = strcmp(name, name_list[mid]);/* مقایسه با اسم موجود در سطر وسط */
      if (res == ۰)              /* تطبیق اسم مورد جستجو با اسم وسط و برگرداندن شمارهٔ سطر وسط */
         return mid;
      else if (res > ۰)          /* نیاز به جستجو در نصفهٔ پایین آرایه */
         first = mid + ۱;/* تغییر نقطهٔ شروع جستجو به شروع نصفهٔ پایینی برای دور بعدی */
      else                       /* نیاز به جستجو در نصفهٔ بالای آرایه */
         last = mid -۱;          /* تغییر نقطهٔ خاتمهٔ جستجو به آخر نصفهٔ بالایی برای دور بعدی */
   }
   return FAIL;          /* یافت نشدن اسم مورد جستجو و برگرداندن مقدار ۱- */
}
```

برنامه ی ۸-۸ : یک تابع اصلی بنویسید که در آن نخست تعدادی اسم در یک آرایه ی دو بعدی از نوع char ذخیره گردد و سپس متناوباً یک اسم را از ورودی خوانده و در صورت وجود آن در بین اسامی موجود در آرایه، اندیس آن را همراه با توضیح مناسب چاپ نماید. در صورت عدم وجود اسم خوانده شده در لیست اسامی نیز پیغام مناسبی چاپ کند.

```
#include <stdio.h>
#include <string.h>
#define MAXLEN   ۳۱
#define MAXROWS ۲۵
#define FAIL    -۱
                /*. . . . . binary_search محل قرارگرفتن تابع . . . . .*/
main()          /* تابعی اصلی  برای بررسی وجود یا عدم وجود یک اسم در آرایه‌ای از اسامی */
{  char s[MAXROWS][MAXLEN] =        /* مقداردهی اولیهٔ آرایهٔ حاوی اسامی */
      {"ahmad", "ali",    "beti", "bidar", "coolman", "donia",
       "hasan", "hosein", "jack",  "jail",   "james",   "jina",
       "jini",  "keivan", "leila", "maryam", "neda",    "pirooz",
       "reza",  "saeed",  "taban", "taghi",  "vega",    "yalda",
       "ziba"};
   int res;
   char t[MAXLEN];

   while (gets(t) != NULL)    /* خواندن اسم مورد جستجو، بررسی وجود آن در آرایه و چاپ نتیجه */
      if ((res = binary_search(t, s, MAXROWS)) < ۰)
         printf("<%s> was not found in the list!\n", t);
      else
         printf("<%s> was found in row %d of the list.\n", t, res);
   return ۰;
}
ahmad                    داده‌های ورودی:
nemat
ziba
jini
james
```

برنامه ی ۸-۹ : یکی از مشکلاتی که در ارتباط با مرتب سازی رکوردها مطرح است این است که در هنگام نیاز به جابه جا کردن داده ها وقت زیادی تلف می شود. برای رفع این مشکل و سرعت بخشیدن به عملیات مرتب سازی روش مرتب سازی tag-sort ارائه شده است. در این روش در کنار داده های ذخیره شده در آرایه یا آرایه هایی که برای مرتب سازی آماده شده اند یک آرایه ی صحیح هم اندازه با آرایه های مزبور تعریف شده و در داخل آن اعداد متوالی از صفر به بعد قرار داده می شود، یعنی سطر k ام آرایه حاوی عدد k است. حال برای رجوع به عناصر آرایه های اصلی و مقایسه ی آنها از محتویات خانه های این آرایه به عنوان اندیس استفاده می گردد و هر بار لازم باشد جای داده های اصلی مثلاً در سطرهای i و j عوض شود، جای اندیس های متناظر در این آرایه تعویض می گردد. این روش مشکل اتلاف وقت در اثر جابه جا کردن رشته های طولانی را حل می کند ولی یک مشکل دیگر هم وجود دارد و آن این است که چون اسامی یا متن های مورد مرتب سازی دارای طول های متفاوتی هستند، برای ذخیره ی آن ها درخانه های متوالی یک آرایه باید اندازه ی خانه های آرایه را مساوی طول حداکثر در نظر گرفت که باعث اتلاف فضای حافظه می گردد. برای رفع این مشکل راه حل این است که متن های مورد مرتب سازی را پشت سر هم در یک آرایه ی بزرگ از نوع char ذخیره کنیم به این ترتیب جدا کننده ی آن ها علامت پایان رشته یعنی \۰ خواهد بود. در کنار این آرایه یک آرایه از جنس اشاره گر ناظر بر char با طولی به اندازه ی حداکثر تعداد اسمها تعریف کرده و آدرس شروع اسمهای ذخیره شده در آرایه ی اصلی را به ترتیب در خانه های این آرایه قرار می دهیم و نهایتاً از این آرایه به جای آرایه ی حاوی اندیس ها برای مرتب

سازی به روش فوق استفاده می کنیم. با توجه به این توضیحات تابعی بنویسید که آرایه ی حاوی آدرس ها را همراه با تعداد اسامی گرفته، اسامی مزبور را به روش فوق مرتب نموده برگرداند.

```
void tag_sort(char *tag[], int n)   /* tag sort تابع مرتب‌سازی داده‌ها به روش */
{  int i, j;
   char *temp;

   for (i = ۰; i < n - ۱; i++) /* حلقه‌های تکرار مرتب‌سازی داده‌ها */
      for (j = i + ۱; j < n; j++)
         if (strcmp(tag[i], tag[j]) > ۰) /* مقایسهٔ دو اسم از طریق آدرس آنها */
         {  temp = tag[i];       /* جابه‌جا کردن فرضی دو اسم از طریق جابه‌جایی آدرسها */
            tag[i] = tag[j];
            tag[j] = temp;
         }
   return;
}
```

شکل ۸-۱۸: متن برنامهٔ ۸-۲، تابعی برای مرتب سازی اسامی به روش tag sort.

برنامه ز ورودی بخواند و سپس به‌این تعداد اسامی با طول های متفاوت (حداکثر ۱۰۰ کاراکتر) را خوانده و در یک آرایه به طول ۱۰۱۰۰ پشت سر هم ذخیره نماید. لازم است این تابع همزمان آرایه ی حاوی آدرس ها را نیز با توجه به توضیحات برنامه ی قبل بسازد. در مرحله ی بعد با احضار تابع tag-sort که در آن برنامه نوشته شد (شکل ۸-۱۷)، را مرتب نموده چاپ نماید.

```
#include <stdio.h>
#include <string.h>
#define MAXNAMES  ۱۰۰     /* حداکثر تعداد اسمها */
#define NAMELEN   ۱۰۰     /* حداکثر طول یک اسم */
#define MAXLEN    ۱۰۱۰۰   /* حداکثر حافظهٔ لازم برای همهٔ اسمها */
#define LINEPPAGE ۲۵            /* حداکثر تعداد سطر مورد چاپ در هر صفحه */
#define LINEWIDTH ۵۰      /* عرض سطر مورد چاپ */
                  /* . . . . . محل قرار دادن تابع مرتب‌سازی . . . . . */

main()            /* تابع اصلی برای خواندن اسامی، احضار تابع مرتب‌سازی و چاپ اسامی مرتب شده */
{ char names[MAXLEN], *tag[MAXNAMES];  /* آرایه‌ها برای ذخیرهٔ اسامی و آدرسها */
  char *nextaddr, name[NAMELEN];/* آرایه‌ها برای ذخیرهٔ هر اسم و اشاره‌گر آدرس خالی بعدی */
  char dash[LINEWIDTH + ۱];          /* آرایهٔ حاوی خط افقی */
  int i, nofnames;

  scanf("%d\n", &nofnames);          /* خواندن تعداد اسمها */
  nextaddr = names;
  for (i = ۰; i < nofnames; i++)     /* حلقهٔ تکرار خواندن اسمها */
  { if (gets(name) == NULL)          /* بررسی ناکافی بودن اسمها */
    { printf("Insufficient number of names!\n");
      return -۱;
    }
                          /* بررسی ناکافی بودن فضای در نظر گرفته شده برای ذخیرهٔ اسمها */
    if (nextaddr + strlen(name) > names + MAXLEN - ۱)
    { printf("Insufficient memory to store names!\n");
      return -۱;
    }
    tag[i] = nextaddr;               /* ذخیرهٔ آدرس شروع اسم خوانده شده */
    strcpy(nextaddr, name);          /* بازنویسی اسم خوانده شده در آرایهٔ اصلی */
    nextaddr += strlen(nextaddr)+۱; /* اصلاح آدرس شروع فضای خالی برای اسم بعدی */
  }
  tag_sort(tag, nofnames);          /* احضار برنامهٔ مرتب‌سازی */
  memset(dash, '-', LINEWIDTH);     /* ساختن خط افقی در حافظه */
  dash[LINEWIDTH] = '\۰';
  for (i = ۰; i < nofnames; i++     /* حلقهٔ تکرار چاپ اسمهای مرتب شده */
  { if (!(i % LINEPPAGE))           /* چاپ عنوان صفحه و عناوین ستونها برای هر ۲۵ نفر */
    { printf("\f%۱۸s\n\n", "LIST OF NAMES");
      printf(" Seq  Name\n");
```