
madsage
IRan Education
Research
NETwork
(IRERNET)
شبکه آموزشی-پژوهشی مادسیج
با هدف بهبود پیشرفت علمی
و دسترسی راحت به اطلاعات
برای جامعه بزرگ علمی ایران
ایجاد شده است
مادسیج
شبکه آموزشی -پژوهشی ایران
madsg.com
مادسیج

فصل 10

فایلها

هدف کلی
آشنایی با انواع فایل و توابع مربوط به آنها

هدفهای رفتاری
از دانشجو انتظار مي‌رود پس از خواندن این فصل،
1. فایل را تعریف کند و انواع آن را بشناسد.
2. تفاوت فایلهای متنی و باینری و کاربرد هر یک را بداند.
3. با نحوة ذخیره و بازیابی داده‌ها آشنا شود.
4. با بازکردن فایل و اطلاعاتی که همزمان با بازشدن فایل مشخص مي‌شود آشنایی یابد.
5. فایل ورودی، خروجی، و ورودی-خروجی را بشناسد.
6. با توابع fopen و fclose آشنایی یابد.
7. توابع putc و getc را بشناسد.
8. کاربرد توابع getw و putw را بداند.
9. کاربرد توابع fget و fputs را بداند.
10. نقش فایل را به عنوان وسیله‌ای هم ورودی و هم خروجی بداند و تابع rewind را بشناسد.
11. با کاربرد تابع ferror آشنایی بیابد.
12. با کاربرد تابع remove آشنایی بیابد.
13. توابع fscanf و fprintf را بشناسد.
14. توابع fwrite و fread را بشناسد.
15. با کاربرد تابع fseek آشنایی بیابد.
16. دستگاههای ورودی - خروجی استاندارد را بشناسد.

مقدمه
متغيرهاي معمولي، آرايه‌ها و ساختمانها همگي در حافظة RAM قرار دارند. لذا پس از خاموش شدن كامپيوتر يا خروج از برنامه داده هايي كه در آنها ذخيره شده‌اند از بين مي روند و براي استفادة مجدد بايد دوباره آنها را وارد كرد كه قطعاً اين كار مقرون به صرفه نيست ، زيرا نه‌تنها مستلزم صرف وقت زيادي است، بلكه حوصلة انجام كار را نيز از برنامه نويس سلب مي‌كند. براي رفع اين مشكل از نوعي ساختمان دادة ديگر به نام فايل استفاده می‌شود. اين نوع ساختمان داده روي حافظ ة جانبي مثل ديسك، نوار و جزآن تشكيل مي‌گردد. چون اطلاعات موجود در روي حافظ ة جانبي با قطع جريان برق، قطع اجراي برنامه يا دلايلي از اين قبيل از بين نمي‌روند، به دفعات زيادي مورد استفاده قرار مي‌گيرند.
هر فايل شامل مجموعه‌اي از داده‌هاي مرتبط به هم است ، مانند داده‌هاي مربوط به كلية دانشجويان دانشگاه. داده‌هاي مربوط به هر يك از اجزاي فايل ركورد نام دارد . براى مثال، در دانشگاه ى داده‌هاي مربوط به هر دانشجو تشكيل يك ركورد را مي دهند، لذا مي‌توان گفت كه هر فايل مجموعه‌اي از چند ركورد است . اگر باز هم دقيق تر به فايل دانشجويان دانشگاه بپردازيم ، مشاهده مي‌كنيم كه هر دانشجو ممكن است چند قلم داده داشته باشد ، مثل نام دانشجو، تعداد واحدهايي كه گذرانده، نمر ة هر درس و جز آن. به هريك از اجزاي يك ركورد فيلد گ‌ويند. لذا مي‌توان گفت كه هر ركورد مجموعه‌اي از چند فيلد است.
در زبان C فايل داده ممكن است هر دستگاهي مثل صفح ة نمايش، صفحه كليد، چاپگر، ترمينال، ديسك، نوار و جز آن باشد. داده‌ها ممكن است به چهار روش در فايل ذخيره و سپس بازيابي شوند:
- داده‌ها كاراكتر به كاراكتر در فايل نوشته و سپس كاراكتر به كاراكتر از فايل خوانده شوند .
- داده‌ها به صورت رشته‌اي از كاراكترها در فايل نوشته شوند و سپس به صورت رشته‌اي از كاراكترها در دسترس قرار گيرند.
- داده‌ها در حين نوشتن بر روي فايل با فرمت خاصي نوشته و سپس با همان فرمت خوانده شوند .

- داده‌ها به شكل ساختمان (ركورد) روي فايل نوشته و سپس به صورت ساختمان از فايل خوانده شوند.

براي هريك از موارد فوق توابع خاصي در زبان C منظور شده‌اند كه در اين فصل بررسي مي‌كنيم.

انواع فايل

داده‌ها ممكن است در فايل به دو صورت متني و باينري وجود داشته باشند. اين دو روش ذخيره شدن داده‌ها در موارد زير با يكديگر تفاوت دارند.

- تعيين انتهاي خط؛
- تعيين انتهاي فايل؛
- نحوة ذخيره شدن اعداد روي ديسك.

در فايل متني اعداد به صورت رشته اي از كاراكترها ذخيره مي شوند، ولي در فايل باينري اعداد با همان صورتي كه در حافظه قرار مي‌گيرند روي ديسك ذخيره مي‌شوند. براي مثال، در فايل متني عدد 526، سه بايت را اشغال مي كند، زيرا هر رقم آن به صورت كاراكتر در نظر گرفته مي‌شود، ولي در فايل باينري اين عدد در 2 بايت ذخيره مي‌گردد (چون عدد 526 عدد صحيح است و اعداد صحيح در حافظة كامپيوتر در دو بايت ذخيره مي‌شوند).

در فايل متني، كاراكتري كه پايان خط را مشخص مي كند، در حين ذخيره شدن روي ديسك بايد به كاراكترهاي CR/LF، line feed، carriage تبديل شود و در حين خوانده شدن عكس اين عمل بايد صورت گيرد ؛ يعني كاراكترهاي CR/LF بايد به كاراكتر تعيين‌كنندة پايان خط تبديل شوند. بديهي است كه اين تبديلها مستلزم صرف وقت است، لذا دسترسي به اطلاعات موجود در فايلهاي متني كندتر از فايلهاي باينري است.

اختلاف ديگر فايلهاي متني و باينري در تشخيص انتهاي فايل است . در هر دو روش ذخيرة فايلها، طول فايل را سيستم نگهداري مي‌كند و انتهاي فايل با توجه به اين طول مشخص مي گردد. در حالت متني كاراكتر 1A (در مبناي 16) يا 26 (در مبناي 10) مشخص‌كنندة انتهاي فايل است (كليد Ctrl + z). در حين خواندن داده ها از روي فايل متني، وقتي كنترل به اين كاراكتر رسيد بيانگر اين است كه داده هاي موجود در فايل تمام شده‌اند. در فايل باينري ممكن است عدد 1A (در مبناي 16) يا 26 (در مبناي 10) جزئي از اطلاعات باشند و بيانگر انتهاي فايل نباشند. لذا نحوة تشخيص انتهاي فايل در فايل باينري با فايل متني متفاوت است.

از نظر نحوة ذخيره و بازيابي داده‌ها در فايل دو روش وجود دارد:

- سازمان فايل ترتيبي
- سازمان فايل تصادفي.

در سازمان فايل ترتيبي، ركوردها به همان ترتيبي كه از ورودي خوانده مي شوند در فايل قرار مي گيرند و در هنگام بازيابي، به همان ترتيبي كه در فايل ذخيره شده‌اند در دسترس قرار مي‌گيرند.

در سازمان فايل تصادفي، به هر ركورد يك شماره اختصاص مي يابد. لذا اگر فايل داراي n ركورد باشد، ركوردها از 1 تا n شماره‌گذاري خواهند شد. وقتي كه ركورد در يك فايل با سازمان تصادفي قرار گرفت، محل آن توسط الگوريتم پيداكنندة آدرس كه با فيلد كليد ارتباط دارد مشخص مي شود. در اين صورت دو ركورد با فيلد كليد مساوي نمي توانند در فايل تصادفي وجود داشته باشند . در سازمان فايل تصادفي مستقيماً مي‌توان به هر ركورد دلخواه دسترسي پيدا كرد بدون اينكه ركوردهاي قبل از آن خوانده شوند.

بازكردن و بستن فايل

هر فايل قبل از اينكه بتواند مورد استفاده قرار گيرد بايد باز گردد. مواردي كه در حين بازكردن فايل مشخص مي‌شوند عبارت‌اند از:

- نام فايل
- نوع فايل ازنظر ذخيرة اطلاعات (متني يا باينري)
- نوع فايل ازنظر ورودي ـ خروجي (آيا فايل فقط ورودي است، آيا فقط خروجي است يا هم ورودي است و هم خروجي).

يك فايل ممكن است طور ي باز شود كه فقط عمل نوشتن اطلاعات روي آن مجاز باشد . به چنين فايلي فايل خروجي گويند. اگر فايل طوري باز گردد كه فقط عمل خواند ن اطلاعات از آن امكان پذير باشد به چنين فايلي فايل ورودي گويند. اگر فايل طوري باز شود كه هم عمل نوشتن اطلاعات روي آن مجاز باشد و هم عمل خواندن اطلاعات از آن، به چنين فايلي فايل ورودي - خروجي گويند. اگر فايلي قبلاً وجود نداشته باشد، در حين بازشدن بايد فايل خروجي باز شود. اگر فايلي قبلاً وجود داشته باشد و به‌عنوان خروجي بازگردد، اطلاعات قبلي آن از بين مي‌رود.

تابع fopen براي باز كردن فليل مورد استفاده قرار مي‌گيرد و داراي الگوي زير است.

```
FILE *fopen (char *filename , *mode)
```

در اين الگو كلمة كليدي FILE با حروف بزرگ نوشته مي شود. filename به رشته‌اي اشاره مي كند كه حاوي نام فايل و محل تشكيل يا وجود آن است. نام فايل داده از قانون نامگذار ي فايل برنامه تبعيت مي كند و شامل دو قسمت نام و انشعاب است كه بهتر است انشعاب فايل داده dat انتخاب گردد. محل تشكيل يا وجود فايل شامل نام درايو و يا هر مسير موجود روي ديسك است. mode مشخص مي‌كند كه فايل چگونه بايد باز شود (ورودي، خروجي يا ...). مقاديري كه مي توانند به جاي mode در تابع fopen قرار گيرند، همراه با مفاهيم آنها در جدول 1ـ10 آمده است.

براي باز كردن فايل بايد اشاره گری از نوع فايل تعريف كرد تا به فايلي كه با تابع fopen باز مي‌شود اشاره كند. اگر فايل به دلايلي باز نشود، اين اشاره‌گر برابر با NULL خواهد بود.

ϖ **مثال 1ـ10** دستورهای زير را درنظر بگيريد.

```
FILE *fp ;
fp = fopen ("A:  test" , "w") ;
```

دستور اول متغير fp را از نوع اشاره گر فايل تعريف مي كند و دستور دوم فايلي به نام test را در درايو A ايجاد مي‌كند چون حالت "w" فايل را به صورت خروجي باز مي‌كند.

ϖ

جدول 1ـ10 مقادير معتبر mode در تابع fopen()

mode	مفهوم
r (rt)	فايلي از نوع متني را به عنوان ورودي باز مي‌كند.
w (wt)	فايلي از نوع متني را به عنوان خروجي باز مي‌كند.
a (at)	فايل را طوري باز مي‌كند كه بتوان اطلاعاتي را به انتهاي آن اضافه كرد.
rb	فايلي از نوع باينري را به عنوان ورودي باز مي‌كند.
wb	فايلي از نوع باينري را به عنوان خروجي باز مي‌كند.
ab	فايل موجود از نوع باينري را طوري باز مي كند كه بتوان اطلاعاتي را به انتهاي آن اضافه كرد.

r + (r+t)	فايل موجود از نوع متنی را به عنوان ورودي و خروجي باز مي‌كند.
w + (w+t)	فايلي از نوع متنی را به عنوان ورودي و خروجي باز مي كند.
a + (a+t)	فايل موجود از نوع متنی را به عنوان ورودي و خروجي باز مي‌كند.
r + b	فايل موجود از نوع باينري را به عنوان ورودي و خروجي باز مي كند.
a + b	فايل احتمالاً موجود از نوع باينري را به عنوان ورودي و خروجي باز مي‌كند.
w + b	فايل از نوع باينري را به عنوان ورودي و خروجي باز مي‌كند.

براي تشخيص اينكه آيا فايل با موفقيت باز شده است يا خير مي توان اشاره‌گر فايل را با NULL مقايسه كرد. NULL ماكرويي است كه در فايل stdio.h تعريف شده است و با حروف بزرگ به كار مي‌رود. اگر اشاره‌گر فايل برابر با NULL باشد بدين معني است كه فايل باز نشده است.

```
if ((fp=fopen ("A:  test" , "w"))= NULL)
  {
    printf ("cannot open file \ n") ;
    exit (0) ;
  }
```

پس از اينكه برنامه‌نويس كارش را با فايل تمام كرد، بايد آن را ببندد. بستن فايل با تابع fclose انجام مي‌شود كه داراي الگوي زير است.

```
int fclose (FILE *fp)
```

در اين الگو fp به فايلي اشاره مي كند كه بايد با تابع fclose بسته شود. به عنوان مثال دستور fclose (p) ; موجب بستن فايلي مي شود كه p به آن اشاره مي‌كند.

توابع putc و getc

براي نوشتن كاراكتر در فايلي كه قبلاً باز شده است، از توابع putc و fputc استفاده مي‌شود. طريقة استفاده از اين دو تابع يكسان است . تابعputc در نسخه‌هاي جديد C و نيز fputc در نسخه‌هاي قديمي C وجود داشته است . چون تابع putc به صورت ماكرو تعريف شده است ، سرعت عمل آن بالاست. الگوي تابع putc به صورت زير است.

```
int putc (int ch , FILE *fp)
```

در اين الگو، ch كاراكتري است كه بايد در فايل نوشته شود و fp اشاره‌گري از نوع فايل است كه مشخص مي كند كاراكتر مورد نظر بايد در چه فايلي نوشته شود.

براي خواندن كاراكترها از فايل مي توان از دو تابع getc و fgetc استفاده كرد. نحوة به كارگيري اين دو تابع يكسان است . تابع fgetc در گونه‌هاي قديمي C و نيز getc در گونه‌هاي جديد C وجود دارد. چون تابعgetc به صورت ماكرو پياده‌سازي شده است، از سرعت بيشتري برخوردار است. الگوي اين تابع به صورت زير است.

```
int getc (FILE *fp)
```

در اين الگو، fp اشاره‌گري است كه مشخص مي‌كند كاراكتر مورد نظر از كدام فايل بايد خوانده شود. در مورد خواندن و نوشتن داده‌ها روي فايل بايد به چند نكته توجه داشت.

اول اينكه، وقتي كاراكترهايي روي فايل نوشته مي شوند بايد مكان بعدي‌اي كه كاراكتر بعدي در آنجا قرار مي گيرد مشخص باشد . همچنين وقتي كه كاراكترهايي از فايل خوانده مي شوند بايد مشخص باشد كه تاكنون تا كجاي فايل خوانده شده است و كاراكتر بعدي از كجا بايد خوانده شود. براي برآوردن اين هدف، سيستم از متغير ی به نام موقعيت سنج فايل استفاده مي كند كه با هر دستور خواندن يا نوشتن روي فايل مقدار اين متغير به طور خودكار تغيير مي‌كند تا موقعيت فعلي فايل را مشخص نمايد . لذا عمل نوشتن روي فايل و عمل خواندن از روي آن از جايي شروع مي‌شود كه اين متغير نشان مي‌دهد.

در هنگام خواندن داده ها از فايل بايد بتوان انتهاي فايل را بررسي كرد؛ يعني در برنامه بايد بتوان اين تست را انجام داد كه اگر در حين خواندن داده‌ها از فايل موقعيت سنج فايل به انتهاي فايل رسيد دستور خواندن بعدي صادر نگردد ، چرا كه در غير اين صورت سيستم پيام خطايي را مبني بر نبودن اطلاعات در فايل صادر مي‌كند.

در حين خواندن داده ها از فايل متنی، پس از رسيدن به انتهاي فايل ، تابع getc يا fgetc علامت EOF را برمي‌گرداند. لذا در هنگام خواندن داده‌ها از فايل متنی مي‌توان به عمل خواندن ادامه داد تا اينكه كاراكتر خوانده شده برابر با EOF گردد. در فايل باينري براي تست كردن انتهاي فايل از تابع feof استفاده مي‌گردد. الگوي اين تابع به‌صورت زير است.

```
int feof (FILE *fp)
```

در اين الگو fp اشاره‌گري است كه مشخص مي كند اين تابع بايد روي چه فايلي عمل كند . تابع fopen علاوه بر تشخيص انتهاي فايلهاي باينري براي تشخيص انتهاي فايلهاي متنی نيز استفاده مي‌شود.

مثال 2.10 برنامة زير كاراكترهايي را از ورودي مي‌خواند و در فايل متني قرار مي دهد. سپس داده‌هاي موجود در اين فايل را مي‌خواند و به فايل ديگري منتقل مي‌كند. آخرين كاراكتر ورودي نقطه در نظر گرفته شده است.

```
# include <stdio. h>
# include <stdlib. h>
void main (void)
  {
    FILE *in , *out ;
    char ch ;
    in = fopen ("F1.txt" , "w") ;
    if (in = = NULL)
      { printf ("cannot open F1.txt \n") ;
        exit(1) ;
      }
     do {
            ch = getchar() ;
            putc (ch , in) ;
```

```
    }while (ch ! = '.') ;
    fclose(in) ;
    out = fopen ("F2.txt" , "w") ;
    if (out = = NULL)
      { printf ("cannot open F2.txt ") ;
        exit(1) ;
      }
    int = fopen ("F1.txt " , "r") ;
    if (in = = NULL)
      { printf ("can not open F1.txt ") ;
        exit(1) ;
      }
    ch = getc(in) ;
    while (ch! = EOF)
      { putc(ch , out) ;
        ch = getc (in) ;
      }
    fclose(in) ;
    fclose(out) ;
  }
```

ϖ

ϖ **مثال 3.10** برنامه‌اي بنويسيد كه كاراكترهايي را از صفحه كليد بگيرد و در فايل باينري قرار دهد. سپس كاراكترهاي موجود در اين فايل را بخواند و به فايل باينري ديگر منتقل كند. اسامي فايلهاي ورودي و خروجي به عنوان آرگومان تابع اصلي به برنامه وارد مي‌شوند.

```
# include <stdio. h>
# include <stdlib. h>
void main (int argc , char *argv[ ])
  {
    FILE *in , out ;
    char ch ;
    clrscr() ;
     if (argc!=3)
       { printf ("you forget enter file name \n ") ;
         exit(1) ;
       }
     in = fopen (argv[1] , "wb") ;
     if (in = = NULL)
       { printf ("cannot open (first) output file\n ");
         exit (1) ;
       }
     do { ch = getchar() ;
          putc(ch , in) ;
        } while (ch ! = '.') ;
     fclose(in) ;
     in = fopen (argv[1] , "rb") ;
     if (in = = NULL)
       { printf ("cannot open input file \n") ;
         exit(1) ;
       }
     out = fopen (argv[2] , "wb") ;
     if (out = = NULL)
       { printf ("cannot open output file \n ") ;
         exit(1) ;
       }
     ch = getc(in) ;
     while (!feof (in))
       {
         putc(ch , out) ;
         ch = getc(in) ;
       }
     fclose(in) ;
     fclose(out) ;
  }
```

ϖ

توابع putw و getw

اين دو تابع مشابه getc و putc اند، با اين تفاوت كه براي خواندن و نوشتن مقادير صحيح از يک فايل به يک فايل ديگر ب ه كار می‌روند. برای مثال دستور putw(50 , fp) ; عدد صحيح 50 را در فايلي كه fp به آن اشاره مي‌كند مي‌نويسد.

ϖ **مثال 4.10** برنامۀ زير مقادير صحيح را از فايلي می‌خواند و مجموع آنها را در خروجي چاپ مي‌كند.

```
#include<stdio.h>
#include<stdlib.h>
main()
  {
     FILE *fp ;
     int sum ;
     if (fp = fopen("sample" , "r") = = NULL)
        {
          printf(" can not open this file \n") ;
          exit(1) ;
        }
     while (!feof(fp))
        sum = sum + getw(fp) ;
     printf(" sum = %d" , sum) ;
     fclose(fp) ;
  }
```

در اينجا تابع getw مقادير صحيح را از فايل sample می‌خواند و شاخص موقعيت فايل را پيش می‌برد. براي مشخص ساختن اينكه به پايان فايل رسيده است يا نه، درون حلقه از دستور feof استفاده شده است.

ϖ

توابع fputs و fgets

براي نوشتن رشته‌ها در فايل از تابع fputs و براي خواندن رشته‌ها از فايل از تابع fgets استفاده مي‌گردد. الگوهاي اين دو تابع به صورت زيرند.

```
int fputs (const char *str , FILE *fp)
char *fgets (char *str , int length , FILE *fp)
```

در الگوهاي فوق، fp اشاره‌گري است كه مشخص مي كند اين توابع بايد روي چه فايلهايي عمل كنند . در تابع fgets اشاره‌گر str به رشته‌اي اشاره مي‌كند كه بايد در فايل نوشته شود. اين اشاره‌گر در تابع fputs به رشته‌اي اشاره مي‌كند كه اطلاعات خوانده شده از فايل در آن قرار مي‌گيرند. length طول رشته‌اي را كه بايد از فايل خوانده شود مشخص مي كند. نحوۀ عمل تابع fgets به‌اين صورت است كه از ابتداي فايل شروع به خواندن مي‌كند تا به انتهاي خط برسد يا رشته‌اي به طول length كاراكتر را از فايل بخواند. برخلاف تابع gets، در تابع fgets كاراكتري كه انتهاي خط را مشخص مي‌كند جزء رشته‌اي خواهد بود كه اين تابع از فايل مي‌خواند.

ϖ **مثال 5.10** برنامۀ زير رشته‌هايي را از ورودي (صفحه كليد) می‌خواند و در فايل قرار مي دهد. از آنجايي كه تابع gets كاراكتري كه پايان خط را مشخص مي‌كند به رشته اضافه نمي‌كند، در حين نوشتن روي فايل اين كاراكتر به رشتۀ خوانده شده اضافه مي‌شود. براي خاتمۀ برنامه كافي است به جاي رشته فقط كليد enter وارد شود.

```
# include "stdio. h"
# include "stdlib. h"
void main (void)
  {
     FILE *fp ;
     char str [80] ;
     if ((fp = fopen ("test" , "w")) = = NULL)
        { printf ("cannot open file \n") ;
          exit(1) ;
        }
     printf ("enter a string") ;
     printf ("ENTER to quit. \n") ;
     while(1)
       { gets (str) ;
          if (str[0]) break ;
             strcat (str , "\n") ;
          fputs (str , fp) ;
       }
     fclose (fp) ;
  }
```

ϖ

فايل وسيلة ورودي ـ خروجي

مي‌توان فايل را هم به عنوان وسيلة ورودي و هم به عنوان وسيلة خروجي مورد استفاده قرار داد . براي اين منظور كافي است در تابع fopen به جاي mode از يكي از عبارات r+ يا r+t براي باز كردن فايل متنی موجود به عنوان ورودي و خروجي استفاده كرد. از يكي از عبارات w+ يا w + t براي ايجاد فايل متنی به عنوان ورودي و خروجي استفاده كرد. و نيز از يكي از عبارات a+ يا a + t براي ايجاد فايل متنی يا باز كردن فايل متنی موجود، به عنوان ورودي و خروجي استفاده كرد.

همچنين از عبارت r + b براي باز كردن فايل باينري موجود، به عنوان ورودي و خروجي استفاده كرد. از عبارت w + b براي ايجاد فايل

باينري به عنوان ورودي و خروجي استفاده كرد. از عبارت a + b براي ايجاد يا بازكردن فايل موجود باينري به عنوان ورودي و خروجي استفاده كرد.

ϖ **مثال 6.10** دستورهای زير را در نظر بگيريد.

```
fp1 = fopen ("test. dat" , "w+b") ;
fp2 = fopen ("sample. dat" , "r+b") ;
fp3 = fopen ("test2. dat" , "a+t") ;
```

دستور اول، فايلي به نام test. dat را از نوع باينري و به صورت ورودي و خروجي باز مي كند كه اشاره‌گر fp1 به آن اشاره مي‌كند. اگر اين فايل قبلاً وجود داشته باشد، محتويات قبلي آن از بين خواهند رفت.

دستور دوم، فايلي به نام sample. dat را كه اكنون در درايو جاري وجود دارد از نوع باينري و به صورت ورودي و خروجي باز مي كند. اگر اين فايل بر روي درايو جاري وجود نداشته باشد، پيام خطايي صادر خواهد شد.

دستور سوم، فايلي به نام test 2. dat را از نوع متنی و به صورت ورودي و خروجي باز مي كند. اگر فايل test2. dat قبلاً وجود نداشته باشد، ايجاد خواهد شد و اگر وجود داشته باشد اطلاعات قبلي آن محفوظ خواهد ماند و اطلاعات جديد به انتهاي آن اضافه خواهد شد.

ϖ

باتوجه به مطالبي كه تاكنون در مورد فايلها گف تيم ، در حين كار با فايلها (نوشتن اطلاعات بر روي آنها و يا خواندن اطلاعات از آنها) براي برگشت به ابتداي فايل (تغيير موقعيت سنج فايل طوري كه به ابتداي فايل اشاره كند) بايد فايل را بست و مجدداً آن را ب از كرد. اصولاً شايد در فايلهايي كه فقط به عنوان خروجي يا فقط به عنوان ورودي باز مي شوند، نياز به برگشت به ابتداي فايل (بدون بستن و باز كردن مجدد آن) احساس نشود، ولي اين امر در مورد فايلهاي ورودي و خروجي ضروری است. براي اين منظور از تابعي به نام rewind استفاده مي گردد. الگوي اين تابع در فايل stdio.h قرار دارد و به صورت زير است.

```
void rewind (FILE *fp)
```

در اين الگو fp به فايلي اشاره مي‌كند كه موقعيت‌سنج آن بايد به ابتداي فايل اشاره كند.

ϖ **مثال 7.10** برنامۀ زير رشته‌هايي را از ورودي می‌خواند و در فايل test قرار مي‌دهد. سپس محتويات اين فايل را می‌خواند و به صفحۀ نمايش منتقل مي‌كند.

```
# include <stdio.h>
# include <stdlib.h>
void main (void)
  {
    FILE *fp ;
    char str [80] ;
    if ((fp=fopen ("test" , "w+")) = NULL)
       {
          printf ("cannot open file\n") ;
          exit (1) ;
       }
     printf ("enter a string") ;
     printf ("Enter to quit \n") ;
     while (1)
       { gets (str) ;
         if (!str [0]) break ;
            strcat (str , "\n") ;
         fputs (str , fp) ;
       }
    printf ("\n the content of file is:  \n ") ;
    rewind (fp) ;
    fgets (str , 79 , fp) ;
    while (!feof (fp))
       { printf ("%s" , str) ;
         fclose (str , 79 , fp) ;
       }
     fclose (fp) ;
  }
```

ϖ

تابع ferror

در حين انجام كار با فايلها ممكن است خطايي رخ دهد . برای مثال، عدم وجود فضاي كافي براي ايجاد فايل، آماده نبودن دستگاهي كه فايل بايد در آنجا تشكيل گردد يا مواردي از اين قبيل منجر به بروز خطا مي‌شوند. با استفاده از تابع ferror مي‌توان از بروز چنين خطايي مطلع شد. الگوي تابع ferror در فايل stdio.h قرار دارد و به صورت زير است.

```
int ferror (FILE *fp)
```

در الگوي فوق fp اشاره‌گري است كه مشخص مي كند اين تابع بايد روي چه فايلي عمل كند . اين تابع تابعی منطقي است؛ بدين معني كه اگر خطايي در رابطه با فايلها رخ داده باشد اين تابع ارزش درست و در غير اين صورت ارزش نادرست را برمي‌گرداند. براي تشخيص خطا در كار با فايلها، بلافاصله پس از هر عملي كه روي فايل انجام مي‌شود بايد از اين تابع استفاده كرد.

ϖ **مثال 8_10** برنامۀ زير كاراكترهاي tab را از فايل حذف مي‌كند و به جاي آن به تعداد كافي فضاي خالي يا blank قرار مي‌دهد. اسامي فايلهاي ورودي و خروجي از طريق آرگومان به برنامه وارد مي‌شود.

```
# include "stdio. h"
# include "stdlib. h"
```



```
# define TAB_SIZE 8
```

```
# define OUT 1
# define IN 1
void err (int) ;
void main (int argc , char *argv[ ])
  {
    FILE *in , *out ;
    int tab , i ;
    char ch ;
    if (argc! = 3)
       {
         printf ("\n incorrect number of parameters ") ;
         printf ("\n\t press any key …") ;
         getch () ;
         exit (1) ;
       }
    in = fopen (argv[2] , "wb") ;
    if (in = = NULL)
      {
        printf ("\n cannot open output file ") ;
        printf ("\n\t press a key …") ;
        exit (1) ;
      }
    tab = 0 ;
    do {
         ch = getc(in) ;
         if (ferror (in))
            err (IN) ;
         if (ch = = '\t')
            { for (i = tab ; i<8 ; i + +)
               { putc (' ' , out) ;
                  if (ferror (out))
                     err (OUT) ;
               }
               tab = 0 ;
            }
     else
       { putc(ch , out) ;
          if (ferror (out))
             err (OUT) ;
          tab + + ;
          if (tab = = TAB_SIAE || ch = = '\n' || ch = = '\r')
            tab = 0 ;
       }
    }while (!feof (in)) ;
    fclose (in) ;
    fclose (out) ;
  }

void err (int error)
  {
    if (erro = = IN)
       printf ("\n error on input file. ")
    else
       printf ("\n press any key …") ;
    getch () ;
    exit (1) ;
  }
  ϖ
```

تابع remove

براي حذف فايلهاي غيرضروري مي‌توان از تابع remove استفاده كرد. الگوي اين تابع در فايل stdio.h قرار دارد و به صورت زير است.

int remove (char *filename)

در اين الگو filename به نام فايلي كه بايد حذف شود اشاره مي‌كند. اگر عمل تابع با موفقيت انجام شود ، مقدار صفر و در غير اين صورت مقداري غير از صفر برگردانده خواهد شد.

ϖ **مثال 10_9** برنامة زير نام فايلي را به عنوان آرگومان مي‌پذيرد و آن را حذف مي‌كند.

```
# include "stdio. h"
# include "stdlib. h"
# include "ctype. h"
main (int argc , char *argv[ ])
  {
    char str [80] ;
    if (argc!=2)
       {
         printf ("\n you must type a file name \n") ;
         exit (1) ;
       }
    printf ("Delete %s (y/n):  " , argv[1]);
    gets (str) ;
    if (toupper (*str) = = 'y')
       if (remove (argv[1]))
         {
           printf ("cannot delete file \n") ;
           exit (1) ;
         }
  }
```

ملاحظه مي‌كنيد كه در اين برنامه، براي حذف فايل مورد نظر از تابع remove استفاده شده است.

ϖ

تابع fprintf , fscanf

اگر لازم باشد كه داده‌ها با فرمت خاصي در فايل نوشته يا از آن خوانده شوند مي توان از دو تابع fprintf و fscanf استفاده كرد. اين دو تابع دقيقاً كار توابع printf و scanf را در ورودي - خروجي معمولي (غير از فايل) انجام مي‌دهند. الگوي اين توابع در فايل stdio. h قرار دارد و به صورت زير است.

```
int fprintf (FILE *fp , "*control_string , ..." , char arg , ...)
int fscanf (FILE *fp , "*control_string , ..." , char arg , ...)
```

در اين الگو fp اشاره‌گري است كه مشخص مي كند اعمال اين توابع بايد روي چه فايلي انجام شود . control_string مشخص مي‌كند كه داده‌ها يا args بايد با چه فرمتي نوشته يا خوانده شوند.

ϖ **مثال 10_10** برنامة زير يك رشته و يك عدد صحيح را از ورودي مي‌خواند و آن را در فايل مي نويسد. سپس از اين فايل مي‌خواند و در صفحه نمايش چاپ مي‌كند.

```
# include <stdio. h>
# include <stdlib. h>
# include <io. h>
void main (void)
  {
    FILE *fp ;
    char str[80] , number [10] ;
    int t ;
    if ((fp = fopen ("test" , "w")) = = NULL)
      {
        printf ("cannot open file\n") ;
        exit (1) ;
      }
    printf ("\n enter string:  ") ;
    gets (str) ;
    strcat (str , "\n") ;
    printf ("\n enter a number:  ") ;
    gets (number) ;
    t = atoi (number) ;
    fprintf (fp , "%s%d , str , t) ;
    fclose (fp) ;
    if ((fp = fopen ("test" , "r")) = = NULL)
       {
         printf ("cannot open file \n") ;
         exit (1) ;
       }
    fscanf (fp , "%s%d" , &str , &t) ;
```



```
    printf ("\nstring = %s , digit = %d" , str , t) ;
```

```
}
```

در مورد توابع fprintf و fscanf بايد توجه داشت كه علي‌رغم اينكه ورودي ـ خروجي با اين دو تابع آسان است، اطلاعات به همان صورتي كه در صفحه نمايش ظاهر مي شوند در فايل ذخيره مي گردند. براى مثال، عدد 267 كه در صفحة نمايش 3 بايت را اشغال مي كند، اگر با تابع fprintf روي فايل نوشته شود نيز 3 بايت را اشغال خواهد كرد (توجه داريم كه عدد 267 عدد صحيح است و در دو بايت ذخيره مي‌شود). اين بدين معني است كه هر رقم به صورت كاراكتر تلقي مي گردد. اگر اين عدد با تابع fscanf از روي فايل خوانده شود، بايد عمل تبديل كاراكتر به عدد صورت گيرد كه مستلزم صرف وقت است . براي جلوگيري از بروز اين مشكل از دو تابع fread و fwrite كه در ادامه بررسي خواهند شد استفاده مي‌شود.

توابع fwrite و fread

توابع متعددي براي انجام اعمال ورودي خروجي فايل وجود دارند . دو تابع fprintf و fscanf براي نوشتن و خواندن انوا ع مختلفي از داده ها و با فرمتهاي متفاوت روي فايل به كار مي‌روند. البته اين دو تابع از سرعت كمي برخوردارند كه توصيه مي شود از آنها استفاده نگردد . براي ورودي - خروجي ركورد و همچنين ساير ورودي - خروجيها مي‌توان از دو تابع fread و fwrite استفاده كرد كه از سرعت بالايي برخوردارند . الگوي اين تابع در فايل stdio.h قرار دارد و به صورتهاي زيرند.

```
int fread (void *buffer , int num_byte , int count , FILE *fp)
int fwrite (void *buffer , int num_byte , int count , FILE *fp)
```

در اين دو الگو پارامتر buffer در مورد تابع fread به ساختمان داده يا متغيري اشاره مي‌كند كه داده‌هاي خوانده شده از فايل بايد در آن قرار گيرند و اين پارامتر در تابع fwrite به محلي از حافظه اشاره مي كند كه داده‌هاي موجود در آن محل بايد در فايل نوشته شوند . پارامتر num_byte در هر دو تابع طول داده اي كه بايد خوانده يا نوشته شود را مشخص مي كند. پارامتر count تعداد عناصري است كه طول آن با num_byte مشخص گرديد و بايد در فايل نوشته يا از فايل خوانده شوند. اشاره‌گر fp به فايلي اشاره مي‌كند كه توابع fread و fwrite بايد روي آنها عمل كنند.

مثال 10_11 مجموعه دستوره اى زير را درنظر بگيريد.

```
char student [20] ;
char str [10] ;
fwrite (student , sizeof (char) , 20 , fp) ;
fread (str , sizeof (char) , 10 , fp) ;
```

دستور اول و دوم رشته‌هايي به طولهاي 20 و 10 را تعريف مي‌كنند. دستور سوم، تعداد 20 بايت از اطلاعات موجود در آرايه student را در فايلي كه fp به آن اشاره مي‌كند مي‌نويسد. دستور چهارم تعداد 10 بايت از اطلاعات را از فايلي كه fp به آن اشاره مي‌كند مي‌خواند و در متغير str قرار مي‌دهد. توابع fread و fwrite بيشتر در ورودي - خروجي ركورد استفاده مي‌شوند.

مثال 10_12 به برنامهٔ زير توجه كنيد.

```
#include<stdio.h>
main()
  {
     FILE *fp ;
     float x = 3.14 ;
     if ((fp = fopen("f1" , "wb")) = = NULL)
       {
          printf("can not open file \n") ;
          return ;
       }
     fwrite(&x , sizeof(float) , 1 , fp) ;
     fclose(fp) ;
  }
```

اين برنامه با استفاده از تابع fwrite متغير float را در فايل مي‌نويسد. در اينجا بافر تابع fwrite متغيری ساده است.

تابعfseek

براي خواندن و نوشتن داده‌ها به صورت تصادفي از اين تابع استفاده مي‌شود. اين تابع اجازه مي‌دهد كه برنامه‌نويس روي اشاره‌گر موقعيت فايل، كنترل داشته باشد. از اين رو با استفاده از اين تابع، براي دستيابي به ركوردي از فايل، اشاره‌گر موقعيت فايل را به ابتداي ركورد مورد نظر انتقال مي‌دهيم.

الگوي اين تابع به صورت زير است.

```
int fseek(FILE *fp , long int num_bytes , int origin) ;
```

در اين الگو fp اشاره‌گر فايل است. پارامتر دوم، تعداد بايتهاي مورد جستجو از مبد أ را مشخص مي‌كند و پارامتر سوم يا origin محل جستجو در فايل را مشخص مي‌كند كه ممكن است يكي از ماكروهاي زير باشد.

مبدأ	نام ماكرو	مقدار ماكرو
شروع از ابتداي فايل	SEEK_SET	0
از موقعيت جاري	SEEK_CUR	1
انتهاي فايل	SEEK_END	2

كاربرد اين تابع در مورد فايلهاي باينري است، زيرا ترجمهٔ كاراكترها در فايلهاي متني موجب بروز اشتباه در مكانها مي‌شود.

مثال 10_13 تابع زير بايت شمارهٔ 54 از فايلي به نام sample را مي‌خواند.

readByte()

```
{
   FILE *fp ;
   if ((fp = fopen("sample" , "rb")) = = NULL)
     {
       printf(" can not open this file \n") ;
       exit(1) ;
     }
   fseek(fp , 54L , 0) ;
   return getc(fp) ;
}
```

اين تابع اگر با موفقيت عمل كند مقدار صفر را بر مي‌گرداند در غير اين صورت مقداري غير از صفر را بر مي‌گرداند. همان طور كه ملاحظه مي‌كنيد از توصيف‌كنندة L براي نشان دادن مقدار long int استفاده شده است. دستور fseek اشاره‌گر فايل را در بايت شمار ة 54 قرار مي‌دهد. سپس در دستور خط بعد كاراكتر موجود در اين محل از فايل با دستور getc خوانده و به تابع فراخواننده بازگردانده مي شود. مقدار صفر استفاده شده در دستور fseek نشان‌دهندة ماكروي FSEEK_SET است.

ϖ

دستگاههاي ورودي ـ خروجي استاندارد

وقتي اجراي برنامه به زبان C آغاز مي‌شود، پنج فايل به طور خودكار باز مي‌شوند. اشاره‌گرهاي آنها را در جدول 10_2 مشاهده مي‌كنيد.

جدول 10_2 دستگاههاي ورودي - خروجي استاندارد

اشاره‌گر فايل	نام دستگاه (فايل)
stdin	دستگاه ورودي استاندارد (صفحه كليد)
stdout	دستگاه خروجي استاندارد (صفحه نمايش)
stderr	دستگاه استاندارد جهت ثبت پيامهاي خطا (صفحه نمايش)
stdprn	دستگاه استاندارد چاپ (چاپگر موازي)
stdaux	پورت سري (serial port)

ϖ **مثال 10_14** مجموعه دستورهای زیر را در نظر بگیرید.

```
putc (ch , stdout) ;
printf (stdout , "%d , %d" , a , b) ;
fscanf (stdin , "%d , %d" , &x , &y) ;
```

دستور اول موجب مي‌شود تا كاراكتر ch در صفحه نمايش نوشته شود. دستور دوم موجب مي‌شود تا متغيرهاي a و b در صفحه نمايش نوشته شوند. دستور سوم موجب مي‌شود تا متغيرهاي x و y از صفحه كليد خوانده شوند.

دستگاههاي استاندارد ورودي ـ خروجي همان طور كه به طور خودكار باز مي‌شوند، به طور خودكار نيز بسته خواهند شد و لازم نيست برنامه‌نويس آنها را ببندد.

خودآزمايي 10

1. برنامه‌اي بنويسيد كه عددي از ورودي بخواند و فاكتوريل آن را در فايل بنويسد.
2. برنامه‌اي بنويسيد كه رشته‌اي از ورودي دريافت كند، سپس رشتة ورودي را به همراه معكوس آن رشته در فايلي درج كند.
3. برنامه‌اي بنويسيد كه فايلي متني به حجم 10 بايت ايجاد كند.
4. برنامه‌اي بنويسيد كه برنامة موجود درفايل ديگري را فايل ورودي بپذيرد و تعداد پرانتزهاي باز و بسته و همچنين تعداد آكولادهاي باز و بستة آن را شمارش كند. نام فايل ورودي به عنوان آرگومان تابع اصلي به برنامه وارد شود.
5. برنامه‌اي بنويسيد كه مشخصات شغلي كارمندان سازمان را، كه شامل نام، تعداد ساعت كار و كارمزد ساعتي است، دريافت كند و در فايل قرار دهد. سپس با استفاده از اين اطلاعات، حقوق دريافتي آنها را محاسبه و چاپ كند.
6. برنامه‌اي بنويسيد كه بتواند از فايل دلخواهی كپي تهيه كند.


شبکه آموزشی-پژوهشی مادسیج
با هدف بهبود پیشرفت علمی
و دسترسی راحت به اطلاعات
برای جامعه بزرگ علمی ایران
ایجاد شده است
madsg.com
مادسیج
IRan Education & Research NETwork
(IRERNET)