فایل ها

اکثر برنامه ها از فایل به منظورذخیره دائمی داده بر روی دیسک استفاده می کنند. فایل ها در C++ در شکل یک جریان (stream) ورودی/خروجی هستند. در این بخش انواع فایل ها، نحوه بازکردن و بستن فایل، خواندن و نوشتن توضیح داده خواهد شد.

فایل مسطح
جریان ها و فایل
اشیای ifstream و ofstream
فایل های متنی
فایل های باینری
دسترسی تصادفی فایل

فایل مسطح

فایل های پایگاه داده دارای ساختار مشخصی هستند. روابط بین داده ها/ رکوردها و فایل های مختلف به روشنی در پایگاه داده تعریف می شود. برخلاف فایل های پایگاه داده، فایل مسطح (flat) ساختاری ندارد. داده در فایل به سادگی بدون ارتباط با داده های دیگر درون فایل یا فایل های دیگر ذخیره می شود. ممکن است تصور بشود که فایل های رابطه ای به دلیل داشتن ساختار معین مناسب تر هستند اما حالت هایی وجود دارد که بهتر است داده در فایل های مسطح ذخیره شود تا پایگاه داده های رابطه ای پیچیده (مانند نوشتن log یا event یا exception).

نکته. فایل هائی که در notepad می سازید فایل مسطح هستند.

جریان ها و فایل

C++ کلیه عملیات ورودی و خروجی و فایل را به صورت جریانی از بایت ها انجام می دهد. یک جریان (stream) یک دنباله از بایت ها است که هر بایت نشان دهنده یک کاراکتر است. جریان ورودی بایت هایی را از دستگاه ورودی، معمولا صفحه کلید یا یک فایل روی دیسک دریافت می کند. جریان خروجی بایت هائی را به صفحه نمایش، چاپگر یا فایل می فرستد.

برنامه باید ارتباطی بین جریان و یک فایل معین روی دیسک برقرار کند. به عبارت دیگر یک جریان مربوط به فایل باید قبل از استفاده باز شود. بعد از باز کردن فایل می توان اطلاعات را از فایل خواند یا درون فایل نوشت. هر فایل باز شده یک اشاره گر فایل دارد که بر اساس تعداد بایت هائی که از ابتدای فایل خوانده یا نوشته شده است موقعیت درون فایل را مشخص می کند. دسترسی به فایل باعث می شود این اشاره گر بهنگام شود.

دو نوع دسترسی به فایل وجود دارد: ترتیبی و تصادفی. در دسترسی ترتیبی (sequential access) داده ها از فایل به ترتیب از ابتدا تا انتها خوانده می شوند. فایل با دسترسی مستقیم یا تصادفی (random access) اجازه می دهد اشاره گر فایل به هر نقطه مورد نظر در فایل پرش کند.

یک جریان فایل می تواند در دو مد متن (text) یا باینری (binary) باز شود. یک فایل متن شامل مجموعه ای از خطوط است. هر خط شامل مجموعه ای از کاراکترهاست که به کاراکتر انتهای خط (newline) ختم می شود (کاراکترهای با کد اسکی 10 و 13). ماکزیمم طول هر خط ۲۵۵ کاراکتر است.

نکته. بخاطر داشته باشید هر خط یک رشته منتهی به کاراکتر NULL نیست بلکه به کاراکتر انتهای خط ختم می شود.

فایل های مسطح متنی نوعی فایل ترتیبی هستند. C++ با این فایل ها به صورت دنباله ای از بایت ها برخورد می کند، فایل های ترتیبی ساختار اضافی ندارند هرساختار اضافی باید توسط برنامه تحمیل شود. راهی برای گردش در فایل های ترتیبی وجود ندارد، فایل همیشه باید از ابتدا شروع شود.

فایل باینری داده ها را به همان فرمتی که در حافظه اصلی نمایش داده می شوند روی دیسک ذخیره می کند. بنابراین مانند فایل های متن اعداد به کاراکتر تبدیل نمی شوند. اگر فایل باینری در یک ادیتور متن باز شود اعداد به صورت کاراکترهای نامفهوم نمایش داده می شوند. فایل های باینری از هیچ نشانه ای برای جدا کردن خطوط داده ای استفاده نمی کنند.

اشیای ifstream و ofstream

در C++ برای کار کردن با فایل از کلاس های ifstream، ofstream و fstream استفاده می شود که در کتابخانه fstream.h تعریف شده اند. اشیای ifstream و ofstream مشابه cin و cout هستند. این اشیا می تواند در برنامه برای نمایش فایل های مسطح و دستکاری آنها بکار رود.

کلاس ifstream برای فایل های ورودی استفاده می شود. اگر می خواهید فایلی را به منظور خواندن از آن باز کنید یک نمونه از این کلاس را مانند زیر ایجاد کنید.

```
ifstream fin();
```

برای باز کردن فایلی به منظور نوشتن در آن باید یک شیء ofstream ایجاد کرد.

```
ofstream fout();
```

کلاس ifstream شامل گروهی از توابع مورد استفاده روی فایل های ورودی است و کلاس ofstream توابع خروجی در فایل را دارد. عملکرد هردو در fsream ترکیب می شود. متدهای اصلی در جدول زیر نشان داده شده اند.

تابع	عملکرد
attach	به stream جاری فایل موجود می چسباند
clear	محتوای بافر را پاک می کند
close	فایل باز متنی یا باینری را می بندد
eof	آیا در انتهای فایل باز شده می باشد یا خیر
get	یک کاراکتر از فایل می گیرد
getline	یک خط کامل از فایل می گیرد
is_open	اگر فایل باز متنی یا بلینری شده باشد true بر می گرداند
open	یک فایل متنی یا باینری را باز می کند
read	از فایل باینری می خواند
seekg	در مد باینری می تواند به مکان مشخصی از فایل برود
seekp	محل اشاره گر فایل متن را تنظیم می کند
tellg	موقعیت جاری اشاره گر فایل باینری را می دهد
tellp	اشاره گر جاری فایل متن را بازیابی می کند
write	یک دنباله از بایت ها را در فایل باینری می نویسد

نام فایل

هر فایل روی دیسک دارای نامی است که هنگام کار کردن با آن مورد استفاده قرار می گیرد. نام فایل به صورت یک رشته کاراکتری است. قواعد نامگذاری فایل بسته به سیستم عامل دارد. در C++ نام فایل می تواند شامل اطلاعات مسیر هم باشد. منظور از مسیر دراویو و فولدری است که فایل در آنجا قرار دارد. اگر نام فایل بدون مسیر مشخص شود محل فایل در موقعیت جاری درنظر گرفته می شود.

مثال. فایل test.txt از مسیر جاری را به منظور خواندن باز می کند.

```
ifstream myfile("test.txt");
```

مثال. اگر فایل در فهرست دیگری باشد باید مسیر فایل کامل ذکر شود.

```
ifstream myfile ("c:\\myfolder\\test.txt");
```

در رشته ها کاراکتر \ بیان کننده کاراکتر escape است و دارای معنی خاص است به همین دلیل برای جداکردن فولدر ها به جای \ باید از \\ استفاده شود. اگر نام فایل از ورودی دریافت می شود یک علامت \ کافی است.

بازکردن فایل

فرآیند ارتباط بین جریان با فایل را باز کردن فایل می نامند. هنگامی که فایلی باز می شود برای خواندن و نوشتن آماده است. متد open یک فایل متن یا باینری را باز می کند. باید یک متغیر از نوع ofstream/ifstream بسته به احتیاج تعریف شود.

```
ifstream infile;
ofstream outfile;
fstream myfile;
```

سپس فایل را با استفاده از open باز شود. اسم/مسیر فایل و مد مورد نظر را تعیین کنید. مد مشخص می کند فایل ورودی، خروجی یا هردو است.فرم کلی متد به صورت زیر است:

```
myfile.open(filename, mode)
```

filename رشته ای است که نام خارجی فایل، یعنی نامی که توسط دیسک شناخته شده است، را مشخص می کند. mode مقداری است که توسط ios تعریف می شود و مشخص می کند فایل با چه مدی (متن/باینری) و به چه منظوری (خواندن/نوشتن/ایجاد) باز شود. با استفاده از عملگر | می توان چند مقدار را با هم تلفیق کرد. پیش فرض فایل در مد متن باز می شود.

مثال. فایل متن myfile.txt را به منظور خواندن باز می کند.
```
infile.open("myfile.txt",ios::in);
```
مثال. فایل باینری myfile.txt را به منظور خواندن باز می کند.
```
infile.open("myfile.txt",ios::in|ios::binary);
```
مثال. فایل متن myfile.txt را به منظور نوشتن باز می کند.
```
outfile.open("myfile.txt",ios::out);
```
مثال. فایل متن myfile.txt را به منظور خواندن باز می کند.
```
myfile.open("myfile.txt",ios::in|ios::out);
```
مثال. فایل متن myfile.txt را به منظور اضافه کردن به انتهای آن خواندن باز می کند.
```
outfile.open("myfile.txt",ios::app);
```
مثال. فایل متن myfile.txt را پاک می کند و برای نوشتن باز می کند.
```
outfile.open("myfile.txt",ios::trunc);
```

تحت شرایطی ممکن است باز کردن فایل با عدم موفقیت روبرو شود نظیر: استفاده از نام فایل غیر مجاز، موجود نبودن فایل روی دیسک یا مسیر ذکر شده، نداشتن اجازه دسترسی و .... اگر بازکردن فایل موفق نباشد تابع مقدار NULL را برمی گرداند که توسط متدهای is_open یا fail بررسی می شود و برنامه باید پیغام خطای مناسب را با cerr نمایش دهد.

مثال.

```
if (!outfile.is_open()) {
   cerr << "Could not create file." << endl;
   exit(1);
   }
```

خواندن و نوشتن فایل

بعد از بازکردن فایل برنامه می تواند داده را از فایل بخواند یا مقداری را در فایل بنویسد. برای فایل های متن عملگرهای << و >> مشابه cin و cout عمل می کنند و می توانند برای خواندن و نوشتن استفاده شوند.

تابع getline تابع خوبی است که اجازه می دهد یک خط از فایل متن (که به کاراکتر انتهای خط ختم شده است) را بخوانید و در یک متغیر رشته ای ذخیره کنید. getline از فایل متن کاراکترها را تا رسیدن به کاراکتر انتهای خط می خواند. اما خود کاراکتر انتهای خط را در رشته ذخیره نمی کند.

در فایل های باینری متدهای read و write برای خواندن و نوشتن بکار می روند.

مثال. کپی کردن یک فایل در دیگری.

```
#include <fstream.h>

int main() {
   ifstream in("scopy.cpp");     // Open for reading
   ofstream out("scopy2.cpp");   // Open for writing

   char s[255];

   while(in.getline(s, 100)) // Discards newline char
       out << s << "\n"; // ... must add it back

   in.close();          //Close input stream
   out.close();         //Close output stream
}
```

تشخیص انتهای فایل

گاهی دقیقا می دانید طول فایل چند بایت است بنابراین نیازی به تشخیص انتهای فایل نیست. ولی در اکثر مواقع از طول فایل اطلاعاتی ندارید. متد eof() زمانی که به انتهای فایل برسید مقدار true را بر می گرداند.

بستن فایل

با ایجاد پیوند بین یک جریان و فایل دیسک اتوماتیک یک بافر ایجاد و به جریان مرتبط می شود. بافر بلاکی از حافظه است که به عنوان واسطه ای بین جریان و سخت افزار دیسک عمل می کند و برای ذخیره موقت داده هائی که از فایل خوانده یا نوشته می شوند بکار می رود. چون دیسک درایو به صورت بلاکی کار می کند، داده ابتدا در بافر ذخیره می شود تا بافر پر شود سپس کل بافر به صورت یک بلاک روی دیسک ذخیره می شود. همین فرآیند زمان خواندن داده از دیسک هم اتفاق می افتد.

در طی اجرای برنامه داده هائی که برنامه روی فایل می نویسد ممکن است در بافر باقی بماند. اگر برنامه بدون بستن فایل خاتمه پیدا کند داده از بافر به درون فایل منتقل نمی شود و اطلاعات از دست می رود. بنابراین بعد از اینکه کارتان با فایل تمام شد باید آنرا ببندید. درصورت نیاز مجددا فایل را باز کنید.

متد close() فایل را می بندد. این متد کل جریان های بافر شده را به فایل منتقل می کند.

```
myfile.close();
```

اگر برنامه دچار شکست شود داده ممکن است داده موجود در بافر از دست برود. برای جلوگیری از این کار در صورت نیاز با استفاده از متد flush() می توانید محتویات بافر را بدون بستن فایل به فایل منتقل کنید.

مثال. یک فایل ممکن است در برنامه به منظور خواندن و نوشتن چندبار باز و بسته شود.

```
#include <fstream.h>
#include <iostream.h>

int main () {
  char buffer[256];
  fstream myfile;       // open it for output then write to it

  myfile.open("test2.txt",ios::out | ios::trunc);
  if (myfile.is_open()) {
     myfile << "This outputting a line.\n";
     myfile.close();
     }

  myfile.open("test.txt",ios::in);  // open it for input and read in
  myfile.getline(buffer,100);
  cout << "The file contains   " << buffer << "\n";
```

```
  myfile.close();

  myfile.open("test.txt",ios::app);  //open for appending and append
  myfile << " Hey this is another line \n";
  myfile.close();

  myfile.open("test.txt",ios::in);  // open it for input and read in
  myfile.getline(buffer,200);
  cout << "The file contains   " << buffer << "\n";
  myfile.close();

  return 0;
}
```

نکته. مدیریت بافر با سیستم عامل است.
نکته. قبل از خواندن یا نوشتن فایل آن را باز کنید.
نکته. همیشه بعد از بازکردن فایل چک کنید فایل بطور موفق باز شده است یا خیر.
نکته. موقعیت خود را درون فایل چک کنید تا از انتهای فایل عبور نکنید.
نکته. در انتها حتما فایل را ببندید.

فایل های متنی

فایل های متنی مورد استفاده بسیاری دارند. یک فایل متنی (text file) جریانی از کاراکترهاست که دارای کاراکتر(های) خاصی برای نشانه گذاری انتهای هر خط است. فایل های متنی را با هر ادیتورمتنی می توان تولید کرد یا محتویات آن را مشاهده کرد.

مثال. ایجاد یک فایل متنی با نام test.txt.

```
#include <fstream.h>

int main() {
   ofstream myfile ("test.txt");

   if (myfile.is_open()){
      myfile << "This outputting a line.\n";
      myfile << "Guess what, this is another line.\n";
      myfile.close();
      }

   return 0;
}
```

مثال. خواندن فایل متنی test.txt و نمایش آن روی صفحه.

```
#include <fstream.h>
#include <iostream.h>

int main (){
   char buffer[256];

   ifstream myfile ("test.txt");

   while (! myfile.eof() ) {
      myfile.getline (buffer,100);
      cout << buffer << endl;
      }

   return 0;
}
```

در فایل های متنی می توانیم اعداد را هم ذخیره کنیم. اعداد به صورت متن ذخیره می شوند. برای مثال عدد 236 به صورت کاراکتر '2' ، کاراکتر '3' و کاراکتر '6' ذخیره می شود. این تبدیل زمان اضافه می برد اما فایل حاصل قابل خواندن است.

فایل های باینری

فایل های باینری هم نوعی فایل مسطح هستند که در حالت دودوئی ذخیره می شوند. در حالت باینری هر فایل یک فرمت بایت به بایت دارد که باعث می شود فایل در ادیتور اسکی خوانا نباشد و کاراکترهای عجیبی نشان داده شود.

تابع write برای نوشتن داده در فایل باینری استفاده می شود. تابع دارای دو پارامتر است. اولی آدرس جائی که داده باید نوشته شود و دومی تعداد بایت های داده است که نوشته می شود. تابع read برای خواندن از فایل باینری است.

دسترسی تصادفی فایل

هر فایل بازی یک اندیکاتور موقعیت دارد که تعیین می کند عمل خواندن/نوشتن در کدام محل فایل انجام می شود. موقعیت همیشه برمبنای تعداد بایت ها از ابتدای فایل داده می شود. اگر فایل موجودی در مد اضافه کردن باز شود اندیکاتور در انتهای فایل است در بقیه مدها اندیکاتور در ابتدای فایل است و برابر صفر است.

توابع خواندن/نوشتن روی فایل که در محل اندیکاتور انجام می شود باعث تغییر موقعیت آن می شود. مثلا اگر فایلی باز شود و ۱۰ بایت از آن خوانده شود اندیکاتور روی موقعیت ۱۰ فایل قرار می گیرد. عمل بعدی روی موقعیت ۱۰ انجام می گیرد.

C++ توابعی را در اختیار می گذارد (seekp و seekg) که امکان کنترل اندیکاتور و دسترسی تصادفی به فایل را می دهند. یعنی می توانید به هر نقطه ای درون فایل مراجعه کنید بدون اینکه مجبور باشید فایل را از ابتدا بخوانید یا بنویسید.

مثال. نمایش اندازه یک فایل.

```
#include <fstream.h>
#include <iostream.h>

int main () {
   long start,end;
   ifstream myfile ("test.txt", ios::in|ios::binary);
   start = myfile.tellg();
   myfile.seekg (0, ios::end);
   end = myfile.tellg();
   myfile.close();

   cout << "size of " << "test.txt";
   cout << " is " << (end-start) << " bytes.\n";
   return 0;
}
```