یادآوری: تمامی تمرینات و اطلاعات مربوط به تحویل آنها در سایت درس قرار داده میشوند:

http://ele.aut.ac.ir/~btaheri/cpp/

فایل ها:

ارسال خروجی به فایل

کلاس `ofstream` که از آن برای انجام عملیات روی فایل خروجی استفاده میشود، از کلاس `ostream` مشتق شدهاست. برای نوشتن در یک فایل باید یک شئ از نوع `ofstream` ایجاد شود و با استفاده از آن عملیات خروجی اجرا شود. در ذیل مراحل ایجاد یک فایل و نوشتن در آن بصورت مقدماتی، مرحلهبهمرحله گفته میشود:

۱ – `#include<fstream>` را به برنامهی خود ضمیمه کنید.

```
#include<fstream>
```

۲ – ایجاد شیئی که باید مشخصات فایلی که قصد نوشتن در آن را داریم از کلاس `ofstream` به صورت زیر:

```
ofstream myOutputFile;
```

۳ – قبل از شروع به نوشتن به فایل، باید فایل مربوطه توسط تابع عضو `open()` از کلاس `ofstream` باز شود:

```
myOutputFile.open("someData.dat");
```

با این تعریف، فایل `someData.dat` به عنوان فایلی که قرار است، اطلاعاتی در آن ذخیره شود، در شی `myOutputFile` تعیین میشود.

۴ – همانند تابع `cout` میتوانید به فایل خود اطلاعاتی را ذخیره کنید. برای مثال:

```
int i = 1, j = 2;
myOutputFile << i << ", " << j << endl;
```

۵ – زمانیکه کار نوشتن به فایل به اتمام رسید، با استفاده از متد `close()` فایل را ببندید:

```
myOutputFile.close();
```

هدف از بستن فایل این است که اولا منابعی از سیستم عامل که توسط برنامه برای نگارش گرفته شدهاند، آزاد شوند و اینکه در صورت توقف ناگهانی سیستمعامل، نوشتن فایلها ناقص نماند. باید گفت دلیل دوم، مسئلهای است که به ساختار سیستمعامل نیز بستگی دارد. بستگی دارد.

مثال۱۴– ۱: میخواهیم برنامهای بنویسیم که یک فایل را ایجاد کرده و اطلاعاتی را در آن ثبت کند.

کد۱۴– ۱

```
1.  //File: streamOutputFile.cpp
2.  #include <fstream>
3.  using namespace std;
4.
5.  int main ()
6.  {
7.     ofstream cppBook("cppBook.txt");
8.     cppBook << "This a C++ programming book!\n"
9.             << "And here we are in steams chapter.\n"
10.            << "And this is an example on file output! ";
11.    return 0;
12. }
```

با توجه به اینکه آدرس فایل مورد نظر نسبی است، بعد از اجرای برنامهی فوق، با رفتن به پوشهای که برنامهتان در آن ذخیره شده است، فایل `cppBook.txt` را خواهیدیافت. اگر آن را باز کنید، خواهید دید که خروجی زیر در آن ذخیره شدهاست.

خروجی

```
This a C++ programming book!
And here we are in steams chapter.
```

```
And this is an example on file output!
```

میبینید توانستیم با استفادهکردن از سازنده[1] مربوط به کلاس ofstream یک فایل در رایانه ایجادکرده و در آن اطلاعاتی را ذخیره کنیم. توجه کنید که در سطر ۷ این مثال از سازندهی ofstream::ofstream(char*) استفاده شدهاست که تنها اسم فایل را به عنوان آرگومان ورودی دریافت میکند و سایر آرگومانها را مقدار ازپیشتعیین شدهی آنها قرار میدهد. در واقع در صورتی که فایلی از قبل با این نام موجود باشد، حذف شده و فایل جدیدی جایگزین آن میشود و در صورتی که وجود نداشته باشد، ایجاد میشود.

در ادامه بحث دقیقتری را روی جزئیات ایجاد فایل و تنظیمات آن انجام میدهیم. همانطور که گفتهشد برای نوشتن در یک فایل باید یک شئ از نوع ofstream ایجاد شود؛ توضیح آرگومان ها به این شرح اند:

- اولین آرگومان ، مربوط به نام فایلی است که قرار است در آن عملیات خروجی مانند نوشتن در فایل انجام گیرد. در واقع با مشخصکردن نام و آدرس فایل، شئ مورد نظر را به فایل متصل میکنیم.
- دومین آرگومان، چگونگی برقراری ارتباط با فایل مورد نظر را مشخص میکند. مقادیر مجاز برای mode در جدول۱۴- ۱ آمدهاست. باید گفت کلاس ios کلاس مادر کلاس ofstream است. (کلاس ها را نخوانده ایم!!)

جدول۱۴- ۱- ثابتهای تعریف شده در کلاس ios برای کنترل شیوهی خواندن و نوشتن فایلها

نشانه(Flag)	مفهوم
ios::app	اضافهکردن خروجی به ادامهی محتوای موجود در فایل- تولید خطا در صورتی که چنین فایلی با چنین نامی وجود نداشته باشد. (مخفف append)
ios::ate	قرارگرفتن در انتهای فایل؛ میتوان اطلاعات را در هرجای فایل نوشت. اگر این تنظیم انجام نگیرد، شروع نوشتن به فایل از ابتدای فایل و با حذفکردن مطالب قبلی انجام می گیرد.(مخفف at the end)
ios::in	بازکردن فایل برای گرفتن محتوای آن به عنوان ورودی (عملیات ورودی)
ios::out	بازکردن فایل برای ذخیرهی خروجی در آن(عملیات خروجی)
ios::trunk	سبب میشود که اطلاعات موجود(در صورت وجود) حذف شوند. (مخفف truncate)
ios::noreplace	اگر قبلا فایلی با چنین نامی وجود داشتهباشد، عملیات با خطا مواجه میشود.
ios::nocreate	اگر فایل با این نام وجود نداشتهباشد، بازکردن فایل با خطا مواجه میشود.
ios::binary	بازکردن فایل در حالت دودویی[2]

نکته!

پارامتر دوم پارامتری اختیاری برای سازندهی ofstream است. لذا معمولا در برنامهها میتوان از مدل سادهشدهی تعریف استفادهکرد و پارامتر دوم را نیاورد. برای مثال فایل myfile.cpp با یک شئ ofstream برای بازکردن و نوشتن در فایل به اینصورت اعلان میشود:

```
ofstream("myfile.cpp");
```

توجه کنید که در این شرایط مقادیر پیشفرض برای این پارامترها در نظر گرفته خواهدشد. یعنی در صورتیکه فایل از قبل وجود داشتهباشد، محتویات قبلی آن از بین میرود.

توجه!

همواره باید توجه کنید که آدرسی که به عنوان آدرس فایل تعریف میکنید، درست باشد. یکی از مهمترین نکاتی که افراد به آن توجه ندارند و منجرو به اشتباه آنها میشود، عدم توجه به تفاوت آدرس کلی(سراسری) ۳ و آدرس نسبی(محلی)۴ است. آدرس نسبی، آدرسی است که نسبت به آدرس فایل اجرایی برنامه

[1] Constructor
[2] Binary

تعیین میشود. لذا با جابجا کردن فایل اجرایی، فایل ایجاد شده نیز در مکانهای مختلفی ایجاد خوهدشد. برای مثال آدرس زیر در پوشهای به نام folderName در آدرس جاری فایل اجرایی برنامه است:

```
myOutputFile.open("foldeName\\someData.dat");  //Win
```

بطور معادل در محیط سیستم عامل لینوکس داریم:

```
myOutputFile.open("/foldeName/someData.dat "); //Linux
```

آدرس سراسری آدرسی است مستقل از آدرس فایل فایل اجرایی برنامهی شماست. نمونهای از آن شیوهی آدرسدهی در ذیل آمدهاست. یا در ویندوز این چنین فایل را باز کنید :

```
myOutputFile.open("C:\\eng101\\hw\\hw6\\hw6.dat"); //Win
```

برای مثال در لینوکس/یونیکس می توانید این چنین وارد کنید :

```
myOutputFile.open("~/Private/someData.dat"); //Linux
```

آیا می توانید دلیل استفاده از دو \ را در ویندوز بگویید؟ شما قبلا دلیل آن را در فصل ۳ خواندهاید!

درخواست انجام عملی برای نوشتن در فایل خروجی همیشه موفقیت آمیز نیست. در واقع گاهی اتفاقاتی غیرقابل پیشبینی ممکن است باعث اخلال در عملیات خواندن و نوشتن شود. در اینصورت اگر برنامهی شما دارای ساختاری برای کنترل موفقیتآمیزبودن بازکردن فایل نباشد، بیتوجه به این مساله به کار خود ادامه میدهد و کارهایی غیرقابل پیشبینی انجام میدهد. لذا لازم است همواره بعد از اجرای تابع open()، بررسی کنید که آیا عملیات بازکردن فایل موفقیتآمیز بودهاست یا خیر. برای مثال تابع fail() نشان میدهد که عملیات بازکردن فایل یا عملیات ورودی/خروجی موفقیتآمیز بودهاست یا خیر. در صورتی که این تابع مقدار true برگرداند، عمل ناموفق بودهاست.

```
if (myOutputFile.fail())
{
   //File open fail
   return 1; //Return to calling function with abnormal termination.
}
// implement file actions here...
```

هرکدام از اشیا از نوع ofstream و ifstream دربردارندهی اطلاعاتی دربارهی نحوهی اجرای عملیات ورودی/خروجی هستند. برای دسترسی به این اطلاعات مربوط به کیفیت اجرای عملیات ورودی/خروجی میتوان از توابع عضو [5] این اشیا استفاده کرد. مهمترین این توابع عضو eof()، bad()، fail()، good() میباشند که خروجی آنها از نوع bool (true یا false) میباشد .

جدول۱۴- ۲-توابع عضو برای اشیای کلاس iostream

تابع	عملکرد
bool bad()	رخداد اتفاقی ناگوار در جریان اجرای عملیات ورودی/خروجی است. این اتفاق میتواند شامل عدم بازشدنفایل، اشیائی در جریان اجرای برنامه نابود(گم)شدهاند و یا هر اتفاق دیگر که عملیات ورودی خروجی را با مشکل مواجه کردهاست.
bool fail()	نشاندهندهی این است که عملیات ورودی/خروجی bad() بوده است یا اینکه عملیات خواندن/نوشتن اخیر با مشکل همراه بودهاست.
bool eof()	در صورتیکه که شئ iostream با EOF(End of File) مواجهشود، مقدار true برمی-گرداند.
bool good()	نشاندهندهی موفقیتآمیزبودن عملیات ورودی/خروجی است. لذا درصورتی این تابع عضو مقدار true بر می گرداند که توابع عضو bad() و fail() و eof() مقدار false را برگردانند.
void clear()	علامت خطا را صفر کرده و این فرصت را میدهد که عملیات را دوباره امتحان کرد.
void close()	فایل بازشده را میبندد. با به موقع بستن فایل بازشده در برنامه، از آسیب دیدن دادههای

[3] Global Address
[4] Local Address
[5] Member Function

	موجود در فایل جلوگیری میشود.

نمونهای از کاربرد این توابع عضو را در مثال بعد مشاهده میکنید.

مثال۱۴– ۲: برنامهای مینویسیم که با استفاده از توابع عضو کلاس `ofstream` تشخیص دهیم خروجی چگونه انجام شدهاست. در این مثال از تابع عضو `bad()` استفاده میکنیم.

کد۱۴– ۲

```
1.   //File: streamOutputErrorChecking.cpp
2.   #include <fstream>
3.   #include <iostream>
4.   using namespace std;
5.
6.   int main ()
7.   {
8.      const static char fileName[] = "myFile.txt”;
9.      ofstream outFile(filename);
10.     if(outFile.bad())
11.     {
12.        cerr << "Error opening file! "
13.             << fileName
14.             << endl;
15.        return 0;  // finish the program
16.     }
17.     outFile << "This a C++ programming book!\n”
18.             << "And here we are in steams chapter.\n”
19.             << "And this is an example on file output!”;
20.     if(outFile.bad())
21.        cerr << "Error writing file!”
22.             << fileName
23.             << endl;
24.     return 0;
25.  }
```

در صورتی که (به احتمال زیاد) هیچ مشکلی در مراحل نوشتن در فایل خروجی بوجود نیاید، در فایل `myFile.txt` خروجی زیر را خواهید یافت:

خروجی

```
This a C++ programming book!
And here we are in steams chapter.
And this is an example on file output!
```

در کد بالا، تمامی تلاشها برای ایجاد خروجی در صورت `true` بودن `outFile.bad()` بیفایدهاست. لذا در صورت بروز هرگونه خطا در عملکرد داخلی برنامه برای نوشتن خروجی، مقدار `outFile.bad()` درست بوده و لذا هیچچیزی در خروجی نوشته نخواهدشد تا زمانی که تابع عضو `clear()` فراخوانی شود تا تمامی علامت(flag)های خطا را از حافظه پاک کند.

مثال۱۴– ۳: برای مراقبتکردن از فایلهایی که از قبل وجود داشتهاند، زمانی که میخواهیم فایلی را برای نوشتن باز کنیم، باید ابتدا این فایل را ابتدا برای خواندن باز کنید و امتحان کنید که آیا بازکردن فایل با موفقیت انجام میشود یا نه؟ در صورتی که بازکردن این فایل موفقیتآمیز نباشد، نشاندهنده ی آن است که این چنین فایلی وجود ندارد. در صورتیکه فایلی با این اسم وجود داشتهباشد، باید به کاربر اعلام کنید تا اسم جدید برای فایل وارد کند.
مثال: برنامهی زیر چنین الگوریتمی را نشان میدهد:

کد۱۴– ۳

```
1.   //File: fileSafeOutput.cpp
2.   #include <iostream>
3.   #include <fstream>
4.   using namespace std;
5.
6.   int main(void)
```

```
7.   {
8.      ifstream testFile; // Create an output file object
9.      ofstream myOutputFile; // Create an output file object
10.     cout << "What filename do you want to write to: ";
11.     char fileName[20]; // 1D array of chars
12.     cin >> fileName;
13.     testFile.open(fileName);
14.     if (!testFile.fail())
15.     {
16.        cout << "File called " << fileName << " already exists.\n";
17.        cout << "Overwrite ";
18.        cout << "(Input 0 for no, any other int to overwrite):";
19.        int clobberIt;
20.        cin >> clobberIt;
21.        testFile.close();
22.        if (!clobberIt)
23.           return 0;
24.     }
25.     myOutputFile.open(fileName); //Connect the stream to the file
26.     if (myOutputFile.fail())
27.     {
28.        // File could not be opened
29.        cerr <<"File called "<<fileName<< " could not be opened.\n";
30.        return 1; // Return to O/S with abnormal return code
31.     }
32.     else
33.        cout << "File " << filename << " was successfully opened.\n";
34.     bool keepReading = true; // Keep reading if true
35.     do
36.     {
37.        cout << "Two floats? ";
38.        float x, y; // Two input floats
39.        cin >> x >> y;
40.        if (cin.eof())
41.           keepReading = false; // End of file <CNTL>-D detected
42.        else
43.           myOutputFile << x << " " << y <<endl;
44.     }while(keepReading);
45.     cout<< "End of input detected\n";
46.     myOutputFile.close(); //Disconnect the stream from the file
47.     return 0;
48.  }
```

این کد از متود `eof()`. بررسی میکند آیا علامتی مبنی بر EOF دریافت کرده است یا نه؟

`cin.eof()` زمانی که `true` برمیگرداند که EOF دریافت شدهباشد، در غیراینصورت مقدار `false` را به عنوان خروجی بر میگرداند. در واقع در این برنامه از `eof()` به عنوان نگهبانی برای بررسی اینکه آیا کنترل برنامه به پایان برنامه رسیدهاست یا نه استفاده میشود. برای ایجاد سیگنال EOF، در سیستمعامل لینوکس/یونیکس، از ctrl+d و در سیستم عامل ویندوز از ctrl+z استفاده میشود.

۸ ۱ ۱ خواندن اطلاعات از فایل

نکته!

مشابه آنچه که در بخش قبل گفته شد، پارامترهای دوم و سوم پارامترهای اختیاری برای سازندهی `ifstream` هستند. لذا معمولا در برنامهها میتوان از مدل سادهشدهی تعریف استفادهکرده و پارامتر سوم و گاهی پارامتر دوم را نیاورد. برای مثال فایل `myfile.cpp` با یک شئ `ifstream` برای خواندن فایل به اینصورت اعلان میشود:

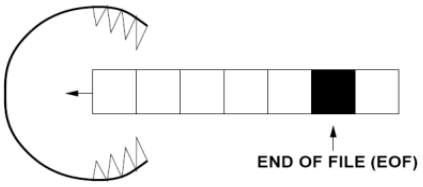
```
ifstream("myfile.cpp");
```

توجه کنید که در این شرایط مقادیر پیشفرض برای این پارامترها در نظر گرفته خواهدشد.

توجه!

لذا برای استفاده از کتابخانهی ifstream تعریف #include<ifstream> را در ابتدای برنامهتان را فراموش نکنید!

هر شئ از جنس ifstream دارای یک تابع عضو eof() است. با استفاده از این فایل میتوان تشخیص داد چه زمانی خواندن فایل به اتمام رسیده است و دیگر چیزی برای خواندن وجود ندارد. معمولا از این تابع برای زمانی استفاده میشود که قصد خواندن تعداد کاراکتر را از فایل ورودی داریم.



شکل۱۴- ۱- نمایش خواندن(خوردن!) محتویات فایل توسط شئ ifstream تا EOF یا eof()

میتوان در یک ساختار حلقهای از تابع eof() استفاده کرد و کاراکترهای داخل یک فایل را خواند. به عنوان مثال:

```
char next;
ifstream in_stream("infile.dat");
in_stream.open();

do
{
   in_stream.get(next);
   cout << next;
} while(! in_strean.eof() )
```

مثال۱۴- ۴: برنامهای مینویسیم که با استفاده از توابع عضو کلاس ofstream بعد از اتمام عملیات روی فایل مورد نظر، از آن خارج شود. شیوهی کار با گرفتن با استفاده از ifstream که مشابه کار با ofstream است نیز آورده شدهاست.

کد۱۴- ۴

```
1.  //File: streamInput.cpp
2.  #include <fstream>
3.  #include <iostream>
4.  using namespace std;
5.
6.  ifstream* openFile()
7.  {
8.     ifstream* pFileStream = 0;
9.     while(true)
10.    {
11.       //Open file:
12.       char fileName[80];
13.       cout << "Enter the name of an existing file: " << endl;
14.       cin >> filename;
15.       pFileStream = new ifstream(filename);
16.       if(pFileStream->good())
17.          break;
18.       cerr << "Couldn't open " << fileName << endl;
19.       delete pFileStream;
20.    }
21. }
22. int main ()
23. {
24.    //Get file stream;
```

```
25.     ifstream* pFileStream = openFile();
26.     //Stop when no more data in file:
27.     while(!pFileStream->eof())
28.     {
29.        // read a value
30.        int nValue = 0;
31.        (*pFileStream) >> nValue;
32.
33.        // Stop if the file read failed.
34.        // Probably because we ran upon something
35.        // that is not an int of because we found a new line
36.        // nothing after it
37.        if(pFileStream->fail())
38.           break;
39.        // Output the value just read:
40.        cout << nValue << endl;
41.     }
42.     return 0;
43.  }
```

تابع `openFile()` در نمایشگر خروجی از کاربرد میخواهد آدرس فایل موجودی را دهد. این روند تا جایی ادامه پیدا میکند که کاربرد آدرس یک فایل معتبر را وارد کند. تابع یک شئ از نوع `istream` را با مشخصات آدرسی که کاربرد وارد کرده است، میسازد. در صورتی که بتواند فایل با این مشخصات را باز کند، آدرس آن را به عنوان خروجی برمیگرداند. در صورتی که بتواند این آدرس را باز کند، آدرس شئ را نابود کرده و با گرفتن آدرسی جدید از کاربر یک شئ جدید میسازد. عملیات `delete` در صورت عدم موفقبودن در بازکردن فایل مسالهی مهمی است. چرا که در صورت فراموششدن، بدون اینکه هرگونه اخطاری از کامپایلر دریافت کنید، نشتی حافظه[6] خواهید داشت.

برنامه مقادیر را از فایل میخواند، مگر اینکه عملیات خواندن با `fail()` مواجه شود یا اینکه روند خواندن به انتهای برنامه برسد که با تابع عضو `eof()` مشخص میشود.

خروجی

```
Enter the name of an existing file:
sarekari
Couldn't open sarekari
Enter the name of an existing file:
integers.txt
1
2
3
4
5
6
```

توجه!

اگر در برنامهای هم نیاز به چاپ در فایل خروجی دارید و هم باید از فایل خروجی اطلاعاتی را بخوانید بجای تعریف دو کتابخانهی `ifstream` و `ofstream` در ابتدای برنامه، کافیست تنها کتابخانهی `fstream` را به برنامهی خود ضمیمه کنید:

```
#include <fstream>
```

[6] Memory leakage

8 2 خواندن و نوشتن فایل باینری

فایل متنی هرچیزی را به عنوان متن ذخیره میکند. در واقع در آنها اعداد همچون 53.456 بصورت رشتهای از اعداد و بصورت ('5','3','.','4','5','6') ذخیره میشوند. اما فایل باینری نه تنها قادر به ذخیرهکردن رشتهها و اعداد صحیح است بلکه قادر به ذخیرهی کل ساختارهای دادهها میباشد. میتوان کل دادهها را به یکباره و با استفاده از متد write() از کلاس fstream در فایل نوشت. اگر از متد write() برای نوشتن استفاده شود، باید از متد read() برای خواندن و بازیابی اطلاعات از یک فایل باینری استفادهکرد.

همانطور که در جدول۱۴– ۱ ذکر شد، برای تعیین اینکه عملیات خواندن و نوشتن روی یک فایل باینری انجام میشود، از علامت(flagِ) ios::binary در سازندهی مربوط به کلاس ifstream یا ofstreamاستفاده میشود.

آرگومان دوم این متدها تعداد کاراکترهایی است که انتظار خواندن و یا نوشتن آنها را داریم که میتوان توسط sizeof() تعیین کرد.

مثال۱۴– ۵: در این مثال فایل باینری...

کد۱۴– ۵

```
1.  //File: binaryFileIO.cpp
2.  #include <iostream>
3.  #include <fstream>
4.  using namespace std;
5.
6.  class Animal
7.  {
8.  public:
9.     Animal(int weight,long days):itsWeight(weight),daysAlive(days){}
10.    ~Animal(){}
11.    int getWight() const {  return itsWeight;  }
12.    void setWeight(int weight) {  itsWeight = weight; }
13.
14.    long getDaysAlive() const { return daysAlive; }
15.    void setDaysAlive(long days){ daysAlive = days; }
16. private:
17.    int itsWeight;
18.    long daysAlive;
19. };
20.
21. int main() //returns 1 on errors
22. {
23.    char fileName[80];
24.
25.    cout << "Please enter file name: ";
26.    cin >> fileName;
27.    ofstream fileOut(fileName, ios::binary);
28.
29.    if(!fileOut)
30.    {
31.       cout << "Unable to open " << fileName << "for writing. \n";
32.       return 1;
33.    }
34.
35.    Animal bear(50,100);
36.    fileOut.write((char*) &bear, sizeof bear);
37.    fileOut.close();
38.
39.    ifstream fileIn(fileName, ios::binary);
40.    if(!fileIn)
41.    {
42.       cout << " " << fileName << "for reading.\n ";
43.       return 1;
44.    }
45.    Animal bearTwo(1,1);
46.
47.    cout << "bearTwo: weight: " << bearTwo.getWight() << endl;
48.    cout << "bearTwo: days: " << bearTwo.getDaysAlive() << endl;
```

```
49.
50.    fileIn.read((char*) &bearTwo, sizeof bearTwo);
51.    cout << "bearTwo weight: " << bearTwo.getWight() << endl;
52.    cout << "bearTwo days: " << bearTwo.getDaysAlive() << endl;
53.    fileIn.close();
54.
55.    return 0;
56. }
```

در خطوط ۵–۲۰ یک کلاس با نام `Animal` تعریف شدهاست. در خطوط ۲۳–۲۴ یک فایل برای خروجی در مد باینری ایجاد شدهاست. در خط ۳۶ یک شئ از نوع `Animal` و با نام `bear` با مشخصات وزن ۵۰ و عمر ۱۰۰ روز ایجاد میشود. اطلاعات مربوط به این شئ(حیوان!) در خط ۳۷ به فایل نوشته میشود.

خروجی

```
Please enter the file name: Animals
bearTwo weight: 1
bearTwo days: 1
bearTwo weight: 50
bearTwo days: 100
```

یک برنامه برای مطالعه برای امتحان پایان ترم(تنها برای مباحث بعد از میانترم):

به ترتیب اولویت:

۱. مطالعه ی فصل های ۶ –۷–۸–۹ کتاب BLIF (به همراه یک قاشق حس کنجکاوی در بررسی برنامه ها!)
۲. مطالعه ی فصل های ۵–۶–۷–۸ از کتاب فارسی (به همراه یک قاشق حس کنجکاوی در بررسی برنامه ها!)
۳. حل دوباره ی تمرینات داده شده (به خصوص ۳ سری آخر)
۴. مرور Handout های تدریسیاران و نوشتن همه ی برنامه ها (به همراه یک قاشق حس کنجکاوی در بررسی برنامه ها!)

در مطالعه ی فصل های فوق حتما سعی کنید زیاد برنامه نویسید و به ایده های اونا فکرکنید. یکی از مزایای یادگیری برنامه نویسی با تمرین اینه که با اشتباهات متداولتون آشنا می شید!

با آرزوی موفقیت در تمام طول زندگی!!