13.2 Text I/O

Monday, April 3, 2023 9:52 AM

- Data in a txt file can be read by a text editor
- Section for I/o
- Every file is placed in a director in the file sys, absolute file name has file named w/ complete path and drive letter
  - Like c:\example\scores.txt
  - This is the absolute file name for the file scores.txt on windows
- Absolute file names are machine dependent
  - UNIX- absolute file name can be
  - Like /home/liang/example/scores.txt
  - Where /home/liang/example is the directory path
- Relative file name is relative to its current working directory, complete directory path for a relative file name is omitted
  - Like scores.txt is a relative file name
  - Current working directory is c:\example
-
-

13.2.1 Writing Data to a File

Saturday, April 08, 2023 10:05 PM

- The ofstream class can be used to write primitive data type vals, arrays, strings, and objects to a text file
-
```
#include <iostream>
#include <fstream>
using namespace std;

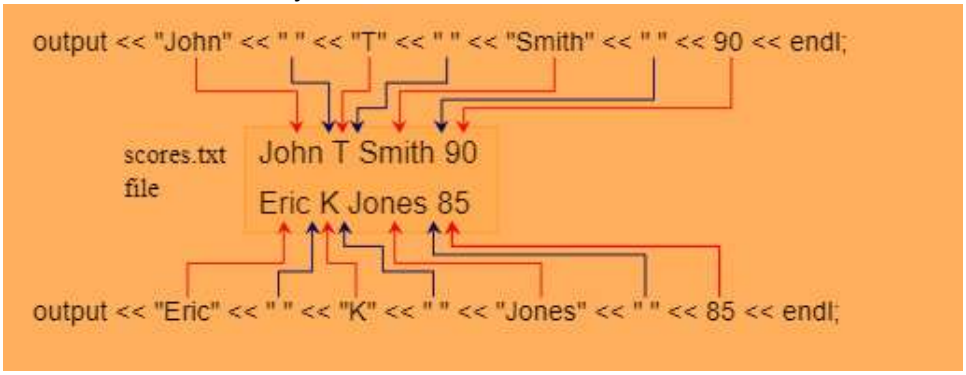
int main()
{
  ofstream output;

  // Create a file
  output.open("scores.txt");

  // Write two lines
  output << "John" << " " << "T" << " " << "Smith"
    << " " << 90 << endl;
  output << "Eric" << " " << "K" << " " << "Jones"
    << " " << 85 << endl;

  output.close();

  cout << "Done" << endl;

  return 0;
}
```
- This pgrm makes an instance of ofstream and writes 2 lines to the file scores.txt
  - Each line has first name (string), middle name initial (char), last name (string), and score (int)
- Since ofstream class defined in the fstream header file, ln 2 has this header file (the header file <fstream>)
- Can write data to the output object using the stream insertion operator (<<) in same way that send data to the cout object
- 

- The close() fn must be used to close the stream for the object, if not invoked, data may not be saved properly in the file
- Can open an output stream using using this constructor:
-
```
ofstream output("scores.txt");
```
- Which is same as
-
```
ofstream output;
output.open("scores.txt");
```
- But if file already exists, the contents would be destroyed w/out warning
- When pgrm writes data to a file, it first stores the data temp to a buffer in the mem, when buffer

full, data auto saved to file on the disk, once close the file, all data left in the buffer are saved to the file on the disk, so must close the file to ensure all data saved to the file

- The directory separator for windows is \, its also a special escape char and should be written as \\ in string literal, like
- ```
  output.open("c:\\example\\scores.txt");
  ```
- An absolute file name is platform dependent, better to use relative file name w/out drive letters, if use IDE, directory of the relative file name can be specified in the IDE
-

To write data to a file, you create an instance of _____.

- iostream
- ifstream
- ofstream ✓
- stream

-

What does the following statement do?

```
ofstream stream("scores.txt");
```

- Open a file for input.
- Open a file for output, the statement fails if the file already exists.
- Open a file for output, the contents of the file is destroyed if the file already exists. ✓
- Open a file for input, the statement fails if the file does not exist.

-

What is the purpose of invoking the close function?

- If this function is not invoked, the data may not be saved properly in the file. ✓
- If this function is not invoked, the file may be deleted.

-
```

13.2.2 Reading Data from a File

Sunday, April 09, 2023 5:02 PM

- The ifstream class can be used to read data from a txt file
-
```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

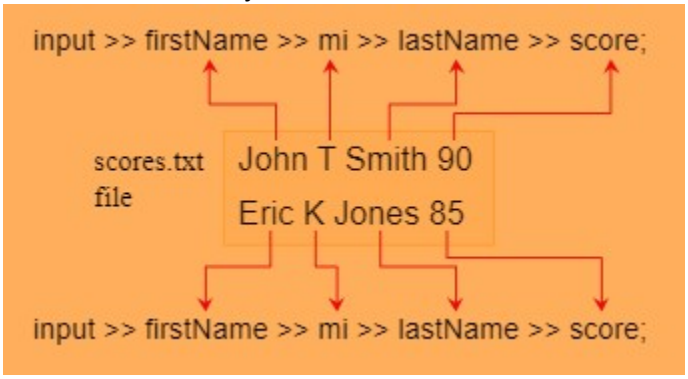
int main()
{
  ifstream input("scores.txt");

  // Read data
  string firstName;
  char mi;
  string lastName;
  int score;
  input >> firstName >> mi >> lastName >> score;
  cout << firstName << " " << mi << " " << lastName << " "
    << score << endl;

  input >> firstName >> mi >> lastName >> score;
  cout << firstName << " " << mi << " " << lastName << " "
    << score << endl;

  input.close();

  cout << "Done" << endl;

  return 0;
}
```
- Make sure use <fstream> header file
- Can read data from the input object using the stream extractor operator (>>) in same way u read data from the cin object
-


- Use close() fn to close stream for the object, not necessary but good practice
- Can open input stream using this constructor:
-
```
ifstream input("scores.txt");
```
- Which is same as
-
```
ifstream input;
input.open("scores.txt");
```
- To read data correctly, need to know exactly how data stored
  - Ex, if thi pic above (john t …) had the score as a double val (using a decimal), it wouldn't

have worked

- To read data from a file, you create an instance of _____.

  - iostream
  - ifstream
  - ofstream
  - stream

- What does the following statement do?

```
ifstream stream("scores.txt");
```

  - Open a file for input.
  - Open a file for output, the statement fails if the file already exists.
  - Open a file for output, the contents of the file is destroyed if the file already exists.
  - Open a file for input, the statement fails if the file does not exist.

What is the output of the following code?

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
  // Create a file
  ofstream output("scores.txt");

  // Write two lines
  output << "John" << " " << "T" << " " << "Smith"
    << " " << 90 << endl;
  output << "Eric" << " " << "K" << " " << "Jones"
    << " " << 85;

  output.close();

  ifstream input;

  // Open a file
  input.open("scores.txt");

  // Read data
  char firstName[80];
  char mi;
  char lastName[80];
  int score;
  input >> firstName >> mi >> lastName >> score;
  double sum = score;

  input >> firstName >> mi >> lastName >> score;
  sum += score;
```

```
  input >> firstName >> mi >> lastName >> score;
  sum += score;

  cout << "Total score is " << sum << endl;

  input.close();

  return 0;
}
```

- Total score is 90
- Total score is 85
- ✓ Total score is 175
- Total score is 0

13.2.3 Testing File Existence

Sunday, April 09, 2023 5:12 PM

- If file don't exist when reading a file, pgrm will run and make wrong results
- Check if file exists first using fail() fn immediately after open fn, if fail() fn returns true, then file don't exist
-

```
// Open a file
input.open("scores.txt");

if (input.fail())
{
  cout << "File does not exist" << endl;
  cout << "Exit program" << endl;

  return 0;
}
```

-

Suppose the file scores.txt does not exist. What will be displayed by the following code?

```
// Open a file
input.open("scores.txt");

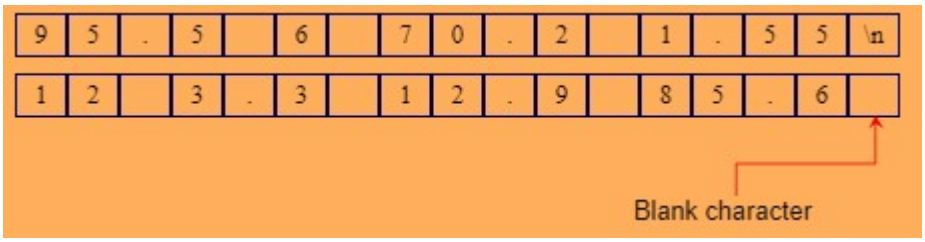
if (input.fail())
{
  cout << "File does not exist" << endl;
  return 0;
}
```

- [x] File does not exist
- [ ] Nothing printed

-

13.2.4 Testing End of File

Sunday, April 09, 2023 5:16 PM

- If don't know how many lines are in file and want to read all, can recognize end of file
- Invoke eof() fn on input object to detect it, wont work if extra blank chars after the last actual chars
- 

- A line ends w/ an end-of-line char
  - Windows -> \r\n
  - Mac -> \r
  - UNIX -> \n
- But there no end-of-file char to designate the end of a file
- If you use the following code to read all data and add the total, the last number will be added twice.

```
ifstream input("score.txt");
double sum = 0;
double number;
while (!input.eof()) // Continue if not end of file
{
  input >> number; // Read data
  cout << number << " "; // Display data
  sum += number;
}
```

- This bc last numb 85.6 is read, file sys doesn't know its last char bc of the blank char after it, so eof() fn returns false
- When pgrm reads numb again, the eof() fn returns true, but var numb is not changed bc nothing read from the file, the var number still has value 85.6 which is added again to sum
- 2 ways to fix prob
  - 1 to check eof() fn right after reading a numb, if eof() fn returns true, exit the loop
  -

```
ifstream input("score.txt");
double sum = 0;
double number;
while (!input.eof()) // Continue if not end of file
{
  input >> number; // Read data
  if (input.eof()) break;
  cout << number << " "; // Display data
  sum += number;
```

```
ifstream input("score.txt");
double sum = 0;
double number;
while (!input.eof()) // Continue if not end of file
{
  input >> number; // Read data
  if (input.eof()) break;
  cout << number << " "; // Display data
  sum += number;
}
```

- 2 is write it like this

```
while (input >> number) // Continue to read data until it fails
{
  cout << number << " "; // Display data
  sum += number;
}
```

- The statement input>>number is actually to invoke an operator fn (later)
- This fn returns an object if a numb is read, otherwise NULL (const val of 0, C++ auto casts it to bool val false when used as condition in loop/selection statement)
- If no numb read from the input stream, input>>number returns NULL, loop terminates

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
  // Open a file
  ifstream input("score.txt");

  if (input.fail())
  {
    cout << "File does not exist" << endl;
    cout << "Exit program" << endl;
    return 0;
  }

  double sum = 0;
  double number;
  while (input >> number) // Read data to the end of file
  {
    cout << number << " "; // Display data
    sum += number;
  }

  input.close();

  cout << "\nTotal is " << sum << endl;

  return 0;
}
```

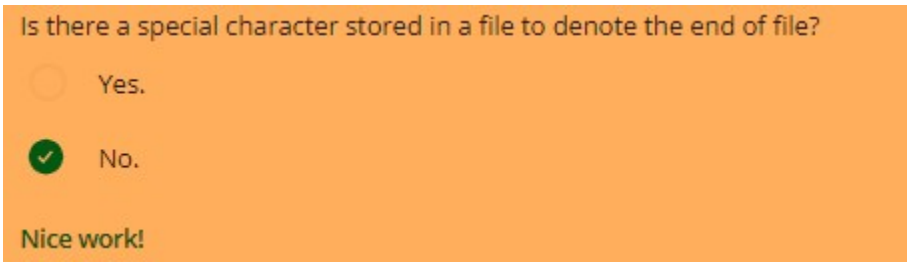
Compile/Run Reset Answer Cho

Execution Result:

```
command>cl TestEndOfFile.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>TestEndOfFile
1 2 3
Total is 6
```

- Pgrm reads data in a loop, each iteration of loop reads 1 numb and adds to sum
- Loop terminates when input reaches end of file

- If replace input>>number w/ cin>>number, the input will be entered from the console, input ends when CTRL + D is pressed
-


-

13.2.5 Letting User Enter a Filename

Sunday, April 09, 2023 5:30 PM

- In these cases, file names are string literals hard coded in the pgrm, many times its desirable to let user enter name of file at runtime
-

```cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
  string filename;
  cout << "Enter a file name: ";
  cin >> filename;

  ifstream input(filename.c_str());

  if (input.fail())
    cout << filename << " does not exist" << endl;
  else
    cout << filename << " exists" << endl;

  return 0;
}
```

Enter input data for the program (Sample data provided below. You may modify it.)

```
c:\example\Welcome.cpp
```

Compile/Run Reset Answer

Execution Result:

```
command>cl CheckFile.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>CheckFile
Enter a filename: c:\example\Welcome.cpp
c:\example\Welcome.cpp exists

command>
```

- This pgrm prompts user to enter a file name as a string, but the file name passed to the input and output stream constructors/to the open fn must be a C-string prior to C++11
- So the c_str() fn in the string class invoked to return a C-string from a string object
- In C++11, u can also pass a string as the file name in the open fn
-

Given a `bool` variable `isReadable`, write some statements that assign `true` to `isReadable` if the file "`topsecret`" exists and can be read by the program and assigns `false` to `isReadable` otherwise.

-

```cpp
ifstream open ("topsecret");
if(open.fail()){
    isReadable = false;
}else{
    isReadable = true;
}
```

-

- Given an `int` variable `x`, write some statements that attempt to open a file named "`table20`" and read a value into `x`; if that turns out not to be possible, your code should then read a value from console input into `x`.

```
ifstream open ("table20");
if(open.fail()){
    cin>>x;
}else{
    open>>x;
```

- Given the availability of an `ofstream` object named `output` and a `string` variable name `tweet`, write the statements to `open` a file named "`mytweet`", display the prompt `tweet:` and then read an entire line into `tweet` and then write it out to the file `mytweet`.

```
output.open("mytweet");
cout<<"tweet:";
getline(cin, tweet);
output<<tweet<<endl;
output.close();
```

- Given the availability of a file named `numbers`, write the statements to read an `int` from standard input and then read in that many values from `numbers` and display their total.

```
ifstream input ("numbers");
int sum = 0;
int numOfNums;
cin>>numOfNums;
int x;
for(int i=0; i<numOfNums; i++){
    input>>x;
    sum+=x;
}
cout<<sum;
```

13.3 Formatting Output

Sunday, April 09, 2023 5:54 PM

- The stream manipulators can be used to format console output as well as file output
- Have already used stream manipulators to format output to console
- Can use same stream manipulators to format output to a file
-

```cpp
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;

int main()
{
  ofstream output;

  // Create a file
  output.open("formattedscores.txt");

  // Write two lines
  output << setw(6) << "John" << setw(2) << "T"
    << setw(6) << "Smith" << " " << setw(4) << 90 << endl;
  output << setw(6) << "Eric" << setw(2) << "K"
    << setw(6) << "Jones" << " " << setw(4) << 85;

  output.close();

  cout << "Done" << endl;

  return 0;
}
```

Compile/Run Reset Answer Cho

Execution Result:

```
command>cl WriteFormattedData.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>WriteFormattedData
Done

command>
```

-

The contents of the file are shown below:

-

		J	o	h	n		T		S	m	i	t	h			9	0	\n
		E	r	i	c		K		J	o	n	e	s			8	5	

-
-

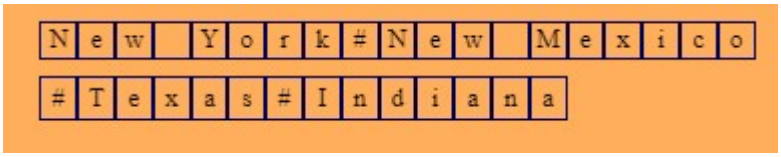
- The following code write text _____ to the file Text.txt.

```
ofstream output("Test.txt");
output << "ba" << setw(6) << left << "bc";
```

  - ba␣␣␣␣bc
  - babc␣␣␣␣
-

13.4 Fn: getline, get, and put

Sunday, April 09, 2023 6:25 PM

- The getline fn can be used to read a string that includes whitespace chars, and the get/put fn can be used to read and write a single char
- Problem in reading data using stream extraction operator, data are delimited by whitespace
- What if whitespace chars are part of a string, can use same fn to read strings from a file
- Syntax for getline fn is
- ```
  getline(ifstream& input, int string& s, char delimitChar)
  ```
- Fn stops reading chars when delimiter char or end-of-file mark encountered
- If delimiter is encountered, its read but not stored in array
- 3rd arg delimitChar has default val ('\n'), the getline fn is defined in the iostream header file
- If file name state.txt made that makes the state names delimited by the pound (#) symbol, file stuff look like
- 

- ```
  #include <iostream>
  #include <fstream>
  #include <string>
  using namespace std;

  int main()
  {
    // Open a file
    ifstream input("state.txt");

    if (input.fail())
    {
      cout << "File does not exist" << endl;
      cout << "Exit program" << endl;
      return 0;
    }

    // Read data
    string city;

    while (!input.eof()) // Continue if not end of file
    {
      getline(input, city, '#'); // Read a city with delimiter #
      cout << city << endl;
    }

    input.close();

    cout << "Done" << endl;

    return 0;
  }
  ```
-

```
command>cl ReadCity.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>ReadCity
New York
New Mexico
Texas
Indiana
Done

command>
```

- Invoking getline(input. State. '#') reads chars to the array *state* until encounters the # char/ end of file
- 2 other useful fn: get and put
- Invoke get fn on input object to read a char and invoke the put fn on an output object to write a char
- Get fn has 2 versions
  -
  -
```
char get() // Return a char
ifstream* get(char& ch) // Read a character to ch
```
  - First Returns a char from the input
  - 2nd passes a char reference arg, reads a char from the input, stores it in ch
  - This fn also returns the reference to the input object being used
- Header for the put fn is
-
```
void put(char ch)
```
- Writes the specified char to the output object
-
```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
  // Enter a source file
  cout << "Enter a source file name: ";
  string inputFilename;
  cin >> inputFilename;

  // Enter a target file
  cout << "Enter a target file name: ";
  string outputFilename;
  cin >> outputFilename;

  // Create input and output streams
  ifstream input(inputFilename.c_str());
  ofstream output(outputFilename.c_str());

  if (input.fail())
  {
    cout << inputFilename << " does not exist" << endl;
    cout << "Exit program" << endl;
    return 0;
  }

  char ch = input.get(); // Read a character
  while (!input.eof()) // Continue if not end of file
  {
    output.put(ch); // Write a character
    ch = input.get(); // Read next character
  }

  input.close();
  output.close();

  cout << "\nCopy Done" << endl;

  return 0;
}
```

-
```
c:\example\CopyFile.cpp
c:\example\temp.txt
```
Compile/Run | Reset | Answer

Execution Result:

```
command>cl CopyFile.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>CopyFile
Enter a source file name: c:\example\CopyFile.cpp
Enter a target file name: c:\example\temp.txt
Copy Done

command>
```

- The pgrm prompts user to enter a src file name and a target file name
- An input object for inputFilename is made, and an output object for outputFilename is made
- What if replace ln 29-34 by

```
while (!input.eof()) // Continue if not end of file
{
  output.put(input.get());
}
```

- If rn the pgrm w/ this new thing, see that new file is 1 byte larger than og
- New file has an extra garbage char at the end bc when last char is read from the input file (using input.get()), input.eof() is still false
- After pgrm attempts to read another char, input.eof(0 now becomes true, but garbage char already sent to output file
- W/out this chunk (so the og), it reads a char and checks eof(), if eof() is true, the char is not put to output, else its copied
- Process continues till eof() == false
-

A variable `c` of type `char` has been declared. Write the code to read in the next character from standard input and store it in `c`, regardless of whether it is a whitespace character.

```
1
2 c = cin.get();
```

-

Declare a `string` named `line` and write a statement that reads in the next line of console input into this variable.

```
1 string line;
2 getline(cin, line);
```

-

Which of the following statements is false?

- The stream.getline(char array[], int size, char delimitChar) function reads data to array. It stops reading characters when the delimiter character or end-of-file mark is encountered, or when the size - 1 number of characters are read.
- The stream.get() function reads one character.
- The stream.put(ch) function writes one character.
- ✓ The stream.getline() function returns a string.

13.5 fstream and File Open Modes

Sunday, April 09, 2023 6:50 PM

- Can use fstream to make a file object for both input and output
- Past, used the ofstream to write data and the ifstream to read data
- Can also use fstream if ur pgrm needs to use the same stream object for both input and output
- To open an fstream file, have to specify a file open mode to tell C++ how the file will be used
-

Mode	Description
ios::in	Opens a file for input.
ios::out	Opens a file for output.
ios::app	Appends all output to the end of the file.
ios::ate	Opens a file for output. If the file already exists, move to the end of the file. Data can be written anywhere in the file.
ios::trunc	Discards the file's contents if the file already exists. (This is the default action for ios:out.)
ios::binary	Opens a file for binary input and output.

- Some of the file modes also can be used w/ ifstream and ofstream objects to open a file
  - Like can use ios::app mode to open a file w/ an ofstream object so u can append data to the file, but for consistency and simplicity, better to use the file modes w/ the fstream objects
- Several modes can be comboed using | operator, its bitwise inclusive OR operator
  - Like to open an output file named city.txt for appending data, can use
  - ```
    stream.open("city.txt", ios::out | ios::app);
    ```
-
    ```

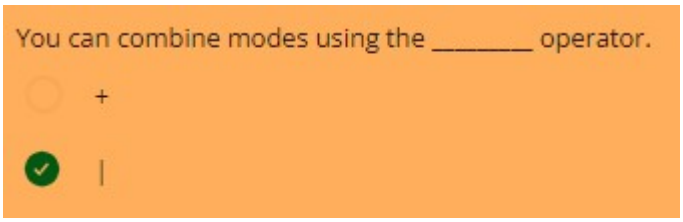
```cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
  fstream inout;

  // Create a file
  inout.open("city.txt", ios::out);

  // Write cities
  inout << "Dallas" << " " << "Houston" << " " << "Atlanta" << " ";

  inout.close();

  // Open a file named city.txt for appending
  inout.open("city.txt", ios::out | ios::app);

  // Write cities
  inout << "Savannah" << " " << "Austin" << " " << "Chicago";

  inout.close();

  string city;

  // Open the file
  inout.open("city.txt", ios::in);
  while (!inout.eof()) // Continue if not end of file
  {
    inout >> city;
    cout << city << " ";
  }

  inout.close();

  return 0;
}
```

```
command>cl AppendFile.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>AppendFile
Dallas Houston Atlanta Savannah Austin Chicago

command>
```

- Pgrm makes an fstream object and opens the file city.txt for output using the file modes ios::out, after writing data the pgrm closes the stream
- Pgrm uses the same stream object to reopen the txt file w/ combo modes ios::out|ios::app , the pgrm then appends new data at the end of the file and closes the stream
- Finally the pgrm uses the same stream object to reopen the txt file w/ the input mode ios::in , pgrm reads all data from the file

Which of the following modes is for input?

- [x] ios::in
- [ ] ios::out
- [ ] ios::app
- [ ] ios::ate
- [ ] ios::binary

- 
You can combine modes using the _____ operator.
+
|

13.6 Testing Stream States

Sunday, April 09, 2023 7:22 PM

- The fns eof(), fail(), good(), and bad() can be used to test the states of stream operations
- C++ has many fn in stream for testing stream states

Stream State Functions

Function	Description
`eof()`	Returns `true` if the end of input stream is reached.
`fail()`	Returns `true` if an operation has failed.
`bad()`	Returns `true` if an unrecoverable error has occurred.
`good()`	Returns `true` if an operation is successful.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

void showState(const fstream&);

int main()
{
  fstream inout;

  // Create an output file
  inout.open("temp.txt", ios::out);
  inout << "Dallas";
  cout << "Normal operation (no errors)" << endl;
  showState(inout);
  inout.close();

  // Create an input file
  inout.open("temp.txt", ios::in);

  // Read a string
  string city;
  inout >> city;
  cout << "End of file (no errors)" << endl;
  showState(inout);

  inout.close();

  // Attempt to read after file closed
  inout >> city;
  cout << "Bad operation (errors)" << endl;
  showState(inout);

  return 0;
}
void showState(const fstream& stream)
{
  cout << "Stream status: " << endl;
  cout << "  eof(): " << stream.eof() << endl;
  cout << "  fail(): " << stream.fail() << endl;
  cout << "  bad(): " << stream.bad() << endl;
  cout << "  good(): " << stream.good() << endl;
}
```

```
command>cl ShowStreamState.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>ShowStreamState
Normal operation (no errors)
Stream status:
  eof(): 0
  fail(): 0
  bad(): 0
  good(): 1
End of file (no errors)

Stream status:
  eof(): 1
  fail(): 0
  bad(): 0
  good(): 0
Bad operation (errors)

Stream status:
  eof(): 1
  fail(): 1
  bad(): 0
  good(): 0

command>
```

- The pgrm makes a fstream object using its no-arg constructor, opens temp.txt for output, and writes a string Dallas, the state of the stream is displayed, no errors so far
- Then pgrm closes the stream, reopens temp.txt for input, reads a string Dallas, state of stream displayed, no errors so far, end of file reached
- Finally, pgrm closes the stream, attempts to read data after file is closed, causes an error, state of stream displayed
- When invoking the showState fn, the stream object is passed to the fn by reference
-

To know whether it is the end of a file, you use the function _____.

- [x] stream.eof()
- [ ] stream.fail()
- [ ] stream.bad()
- [ ] stream.good()
- [ ] stream.clear()

-

To know whether the I/O operation succeeded, you use the function _____.

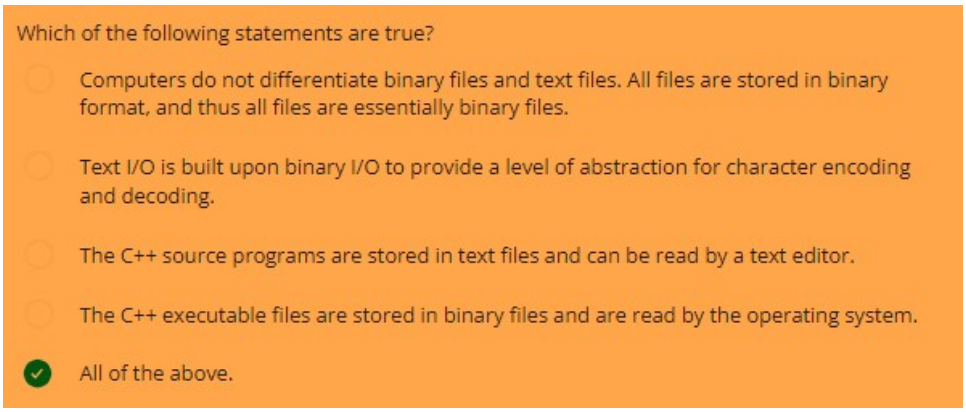
- [ ] stream.eof()
- [x] stream.fail()
- [ ] stream.bad()
- [ ] stream.good()
- [ ] stream.clear()

•

13.7 Binary I/O

Sunday, April 09, 2023 7:29 PM

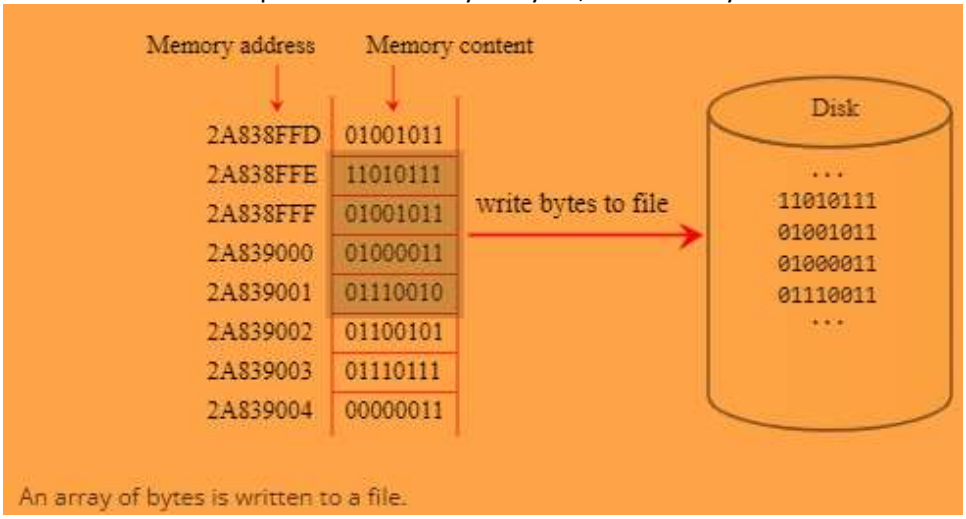
- The ios::binary mode can be used to open a file for binary input & output
- Files can be classified as text or binary
  - Files that can be processed (read, made, moded) using text editor are called text files
  - All others are binary files, cannot read binary files using a txt editor, they designed to be read by pgrms
    - C++ src pgrms are stored in txt files and can be read by a text editor, but the C++ executable files are stored in binary files and are read by the os
- Think of txt files as having a sequence of chars and a binary file as having a sequence of bits (not correct or precise, but good to think abt)
  - Deciaml int 199 stored as sequence of 3 chars "199" in txt file, but "C7" in binary file bc decimal 199 = C7 in hex
- Advantage of binary files is they more efficient to process than txt files
- Computers do not differentiate btwn binary and txt files, all files stored in binary format, so they all kinda binary files
- Text I/O built on binary I/O to give a lvl of abstraction for char encoding and decoding
- Binary I/O don't req conversion, if write a numeric val to a file using binary I/O, the exact val in the mem is copied into the file
- To perform binary I/O in C++, have to open a file using the binary mode ios::binary, by default, a file is opened in text mode
- Cant use <<, >>, get, and getline fns to read data from a txt file
- To read/write data to /from binary file, must use read and write fns on a stream
-



-

13.7.1 The write fn

Sunday, April 09, 2023 7:40 PM

- Syntax for the write fn is
- ```
  streamObject.write(const char* bytes, int size)
  ```
- This writes an array of bytes in the type char* , each char is a byte
- Char* should be interpreted as an array of bytes, not an array of chars
- 

- In C++, no data type byte (but if it did, it would make sense to use byte* instead of char*)
- ```
  #include <iostream>
  #include <fstream>
  #include <string>
  using namespace std;

  int main()
  {
    fstream binaryio("city.dat", ios::out | ios::binary);
    string s = "Atlanta";
    binaryio.write(s.c_str(), s.size()); // Write s to file
    binaryio.close();

    cout << "Done" << endl;

    return 0;
  }
  ```

  Compile/Run Reset Answer Cho

  Execution Result:

  ```
  command>cl BinaryCharOutput.cpp
  Microsoft C++ Compiler 2019
  Compiled successful (cl is the VC++ compile/link command)

  command>BinaryCharOutput
  Done

  command>
  ```
- Often need to write data other than chars, can use the reinterpret_cast operator , which can cast a pointer type to any other pointer type, simply does a reinterpretation of the value from one type to other w/out altering the data
- Syntax for reinterpret_cast is

- `reinterpret_cast<dataType*>(address)`
- Where address is starting address of the data (primitive, array, or object) and dataType is the data type u are casting to, in this case it is char*
-
```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
  fstream binaryio("temp.dat", ios::out | ios::binary);
  int value = 199;
  binaryio.write(reinterpret_cast<char*>(&value), sizeof(value));
  binaryio.close();

  cout << "Done" << endl;

  return 0;
}
```

Compile/Run Reset Answer Choose a Com

Execution Result:

```
command>cl BinaryIntOutput.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>BinaryIntOutput
Done

command>
```

- For consistency, the website uses .txt for txt files, and .dat for binary files
-
```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
  float floatValue = 19.5;
  int* p = reinterpret_cast<int*>(&floatValue);
  cout << "int value " << *p << " and float value "
    << floatValue << "\nhave the same binary representation "
    << hex << *p << endl;

  return 0;
}
```

Automatic Check Compile/Run Reset Answer Choose

Execution Result:

```
command>cl ReinterpretCastingDemo.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>ReinterpretCastingDemo
int value 1100742656 and float value 19.5
have the same binary representation 419c0000

command>
```

- Pgrm casts a pointer to a float number 19.5 to a pointer of the int type, same raw data in the mem

is reinterpreted as an int, raw data is 419c0000 in hex which is 1100742656 when interpreted as int and 19.5 when interpreted as a float

- Note- hex is a manuplator defined in <iomanip> header file to display a number in hex
-

To write a character string to a binary file, use the function _____.

- ✓ biStream.write(char* address, int size)
- biStream.write(char* address)
- biStream.write(string address, int size)
- biStream.write(string address)

-

Suppose you declare int value = 99, to write it to a binary file, use _____.

- `binaryio.write(value);`
- ✓ `binaryio.write(reinterpret_cast<char*>(&value), sizeof(value));`
- `binaryio.write(reinterpret_cast<char*>(&value));`
- `binaryio.write(reinterpret_cast<char*>(value));`
- `binaryio.write(reinterpret_cast<char*>(value), sizeof(value));`

-

This book uses .dat to denote _____ file.

- a text
- ✓ a binary
- a source
- an input
- an output

-

13.7.2 The read fn

Sunday, April 09, 2023 8:39 PM

- Syntax for read fn is
- ```
  streamObject.read(char* address, int size)
  ```
- The size param indicates the max number of bytes read, actual numb of bytes read can be gotten from a member fn gcount
- ```
  #include <iostream>
  #include <fstream>
  using namespace std;

  int main()
  {
    fstream binaryio("city.dat", ios::in | ios::binary);
    char s[10];
    binaryio.read(s, 10); // Read into s
    cout << "Number of chars read: " << binaryio.gcount() << endl;
    s[binaryio.gcount()] = '\0'; // Append a C-string terminator
    cout << s << endl;
    binaryio.close();

    return 0;
  }
  ```

  Compile/Run Reset Answer Choose a Compiler:

  Execution Result:

  ```
  command>cl BinaryCharInput.cpp
  Microsoft C++ Compiler 2019
  Compiled successful (cl is the VC++ compile/link command)

  command>BinaryCharInput
  Number of chars read: 7
  Atlanta

  command>
  ```
- Pgrm opens the binary file city.dat for input, invoking binaryio.read(s, 10) reads up to 10 bytes from the file to the array, the actual number of bytes read can be found by invoking binaryio.gcount()

-
```cpp
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
  fstream binaryio("temp.dat", ios::in | ios::binary);
  int value;
  binaryio.read(reinterpret_cast<char*>(&value), sizeof(value));
  cout << value << endl;
  binaryio.close(); // Close binaryio

  return 0;
}
```

Compile/Run Reset Answer Choose a Comp

Execution Result:

```
command>cl BinaryIntInput.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>BinaryIntInput
199

command>
```

- The data in the file temp.dat were made b4, the data had an integer and were cast to chars b4 stored, this pgrm first read the data as bytes and then used the reinterpret_cast operator to cast bytes into an int val
-
- To read a character string from a binary file, use the function _____.
- ✓ biStream.read(char* address, int size)
- biStream.read(char* address)
- biStream.read(string address, int size)
- biStream.read(string address)
-

- Suppose you want to read an int to the variable value from a binary file, use _____.

  ```
  value = biStream.read();
  ```

  ```
  biStream.read(value);
  ```

  ✓
  ```
  binaryio.read(reinterpret_cast<char*>(&value), sizeof(value));
  ```

  ```
  biStream.read(&value);
  ```

- To open a file for binary input, use the mode _____.

  ✓ ios::in | ios::binary

  ios::out | ios::binary

  ios::app | ios::binary

  ios::ate | ios::binary

13.7.3 Ex: Binary Array I/O

Sunday, April 09, 2023 8:47 PM

- U can reinterpret_cast operator to cast data of any type to bytes and vice versa
-

```cpp
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
  const int SIZE = 5;  // Array size

  fstream binaryio; // Create stream object

  // Write array to the file
  binaryio.open("array.dat", ios::out | ios::binary);
  double array[SIZE] = {3.4, 1.3, 2.5, 5.66, 6.9};
  binaryio.write(reinterpret_cast<char*>(&array), sizeof(array));
  binaryio.close();

  // Read array from the file
  binaryio.open("array.dat", ios::in | ios::binary);
  double result[SIZE];
  binaryio.read(reinterpret_cast<char*>(&result), sizeof(result));
  binaryio.close();

  // Display array
  for (int i = 0; i < SIZE; i++)
    cout << result[i] << " ";

  return 0;
}
```

Compile/Run Reset Answer Choose a Comp

Execution Result:

```
command>cl BinaryArrayIO.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>BinaryArrayIO
3.4 1.3 2.5 5.66 6.9

command>
```

- Pgrm writes an array of double vals to a binary file and reads it back from the file
- Pgrm makes a stream object, opens the file array .dat for binary output, writes an array of double vals to the file, and closes the file
- Pgrm then opens the file array .dat for binary input, reads an array of double vals from the file, and closes the file
- Finally, pgrm displays the contents in the array result
-

Suppose you declare double array[SIZE] and array is written to the file using binary I/O. To read the array to result (double result[SIZE]), use ____.

```
binaryio.read(&result, sizeof(Student));
```

```
binaryio.read(result);
```

```
binaryio.read(&student1);
```


binaryio.read(&student1);
binaryio.read(reinterpret_cast<char*>(&result), sizeof(result));

13.7.4 Ex: Binary Object I/O

Sunday, April 09, 2023 8:52 PM

- Section gives ex of writing objects to a binary file and reading the objects back from the file
- B4, had pgrm that wrote student records into a txt file, a student record had a first name, middle initial, last name, and a score
- These fields were written to the file separately, better version would be to define a class to model records, each record is an object of the Student class
- Let class be named Student w/ data fields firstName, mi, lastName, and score, their supporting accessors and mutators, and 2 constructors
- **Figure 13.5**

  The getter and setter functions for property values are provided in the class, but omitted in the UML diagram for brevity.

Student
-firstName: char[25] -mi: char -lastName: char[25] -score: int
+Student() +Student(firstName: string&, mi: char, lastName: string&, score: int)

  The Student class describes student information.
- So in this, Student class in the header file, later implemented
- First names and last names are stored in 2 arrays of chars w/ a fixed-length 25 internally, so every student record will have same size, this needed so students can be read from the file correctly
- Easier to use the string type than C-string, the string type is used in the get and set fn for firstName and lastName
- ```
  1  #ifndef STUDENT_H
  2  #define STUDENT_H
  3  #include <string>
  4  using namespace std;
  5
  6  class Student
  7  {
  8  public:
  9    Student();
 10   Student(const string& firstName, char mi,
 11   const string& lastName, int score);
 12   void setFirstName(const string& s);
 13   void setMi(char mi);
 14   void setLastName(const string& s);
 15   void setScore(int score);
 16   string getFirstName() const;
 17   char getMi() const;
 18   string getLastName() const;
 19   int getScore() const;
 20
 21 private:
 22   char firstName[25];
 23   char mi;
 24   char lastName[25];
 25   int score;
 26 };
 27
 28 #endif
  ```
  ```

```
  int score;
};

#endif
```

-

```
#include "Student.h"
#include <cstring>
// Construct a default student
Student::Student()
{
}

// Construct a Student object with specified data
Student::Student(const string& firstName, char mi,
  const string& lastName, int score){
  setFirstName(firstName);
  setMi(mi);
  setLastName(lastName);
  setScore(score);
}
void Student::setFirstName(const string& s){
  strcpy(firstName, s.c_str());
}
void Student::setMi(char mi){
  this->mi = mi;
}
void Student::setLastName(const string& s){
   strcpy(lastName, s.c_str());
}
void Student::setScore(int score){
  this->score = score;
}

string Student::getFirstName() const{
  return string(firstName);
}

char Student::getMi() const{
  return mi;
}

string Student::getLastName() const{
  return string(lastName);
}

int Student::getScore() const{
  return score;
}
```

-

```
#include <iostream>
#include <fstream>
#include "Student.h"
using namespace std;

void displayStudent(const Student& student){
  cout << student.getFirstName() << " ";
  cout << student.getMi() << " ";
  cout << student.getLastName() << " ";
  cout << student.getScore() << endl;
}
int main(){
  fstream binaryio; // Create stream object
  binaryio.open("student.dat", ios::out | ios::binary);

  Student student1("John", 'T', "Smith", 90);
  Student student2("Eric", 'K', "Jones", 85);
  Student student3("Susan", 'T', "King", 67);
  Student student4("Kim", 'K', "Peterson", 95);

  binaryio.write(reinterpret_cast<char*>
    (&student1), sizeof(Student));
  binaryio.write(reinterpret_cast<char*>
    (&student2), sizeof(Student));
  binaryio.write(reinterpret_cast<char*>
    (&student3), sizeof(Student));
  binaryio.write(reinterpret_cast<char*>
    (&student4), sizeof(Student));
  binaryio.close();
   // Read student back from the file
  binaryio.open("student.dat", ios::in | ios::binary);

  Student studentNew;
  binaryio.read(reinterpret_cast<char*>
    (&studentNew), sizeof(Student));

  displayStudent(studentNew);

  binaryio.read(reinterpret_cast<char*>
    (&studentNew), sizeof(Student));

  displayStudent(studentNew);

  binaryio.close();
  return 0;
}
```

- This ^ pgrm makes 4 Student objects, writes them to a file called student.dat, and reads them back from the file

```
command>cl BinaryObjectIO.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>BinaryObjectIO
John T Smith 90
Eric K Jones 85

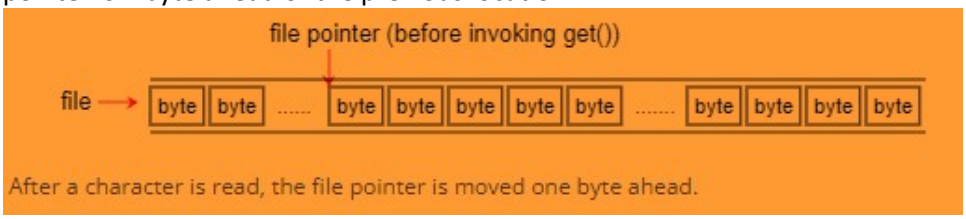
command>
```

- The pgrm makes a stream object, opens the file student.dat for binary output, makes 4 Student objects, writes them to the file, and closes the file
- Statement to write an object to the file is

```
binaryio.write(reinterpret_cast<char*>
  (&student1), sizeof(Student));
```

- Address of object student1 is cast into the type char* , the size of an object is determined by the data fields in the object, every student has the same size, which is sizeof(Student)
- The pgrm opens the file student.dat for binary input, makes a Student object using its no-arg constructor, reads a Student object from the file, and displays the object's data
- The pgrm cont read another object and displays its data, finally pgrm closes the file

- Suppose you declare `Student student1`. To write `student1` to a binary file, use ____.

  ```
  binaryio.write(&student1, sizeof(Student));
  ```

  ```
  binaryio.write(student1);
  ```

  ```
  binaryio.write(&student1);
  ```

  ✓ `binaryio.write(reinterpret_cast<char*>(&student1), sizeof(Student));`

- Suppose you declare `Student student1`. To read a `Student` object from a binary file, use ____.

  ```
  binaryio.read(&studentNew, sizeof(Student));
  ```

  ```
  binaryio.read(&studentNew);
  ```

  ✓ `binaryio.read(reinterpret_cast<char*>(&studentNew), sizeof(Student));`

  ```
  binaryio.read(reinterpret_cast<char*>(studentNew), sizeof(Student));
  ```

13.8 Random Access File

Sunday, April 09, 2023 9:10 PM

- Fn seekg() and seekp() can be used to move file pointer to any position in a random-access file for input and output
- A file has a sequence of bytes, special marker called file pointer positioned at one of these bytes
- A read/write op takes place @ the location of the file pointer, when a file is opened, the file pointer is set at the beginning of the file, when u read/write data to the file, the file pointer moves forward to the next data item
- Ex: if u read a char using the get() fn, C++ reads 1 byte from the file pointer, and now the file pointer is 1 byte ahead of the previous location
- 

After a character is read, the file pointer is moved one byte ahead.
- All the pgrms so far read/write sequentially, this called sequential access file
- So file pointer always moves forward, if a file is open for input, it starts to read data from the beginning to the end, if a file is open for output, it writes data one item after the other from the beginning or from the end (w/ the append mode ios::app)
- Prob w/ sequential access is that in order to read a byte in a specific location, all the bytes that precede it must be read, not efficient, C++ lets the file pointer to jump back/forward freely using the seekp and seekg member fns on a stream object, this called random access file
- The seekp ("seek put") fn is for the output stream, and the seekg ("seek get") fn is for the input stream, each fn has 2 versions w/ 1 arg or 2 args
- W/ 1 arg, arg is the absolute location
- ```
  input.seekg(0);
  output.seekp(0);
  ```
- Which moves the file pointer to the beginning of the file
- W/ 2 args, the first arg is an int that indicates an offset, the 2nd arg (aka seek base) specifies where to calculate the offset from
- **Table 13.3**
  Seek Base

Seek Base	Description
ios::beg	Calculates the offset from the beginning of the file.
ios::end	Calculates the offset from the end of the file.
ios::cur	Calculates the offset from the current file pointer.

  Table 13.4 gives some examples of using the seekp and seekg functions.
  ```

Table 13.4

seekp and seekg Examples

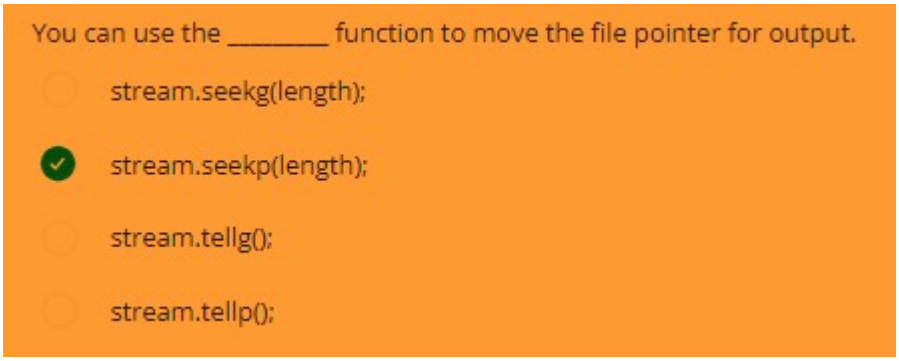
Statement	Description
`seekg(100, ios::beg);`	Moves the file pointer to the 100th byte from the beginning of the file.
`seekg(-100, ios::end);`	Moves the file pointer to the 100th byte backward from the end of the file.
`seekp(42, ios::cur);`	Moves the file pointer to the 42nd byte forward from the current file pointer.
`seekp(-42, ios::cur);`	Moves the file pointer to the 42nd byte backward from the current file pointer.
`seekp(100);`	Moves the file pointer to the 100th byte in the file.

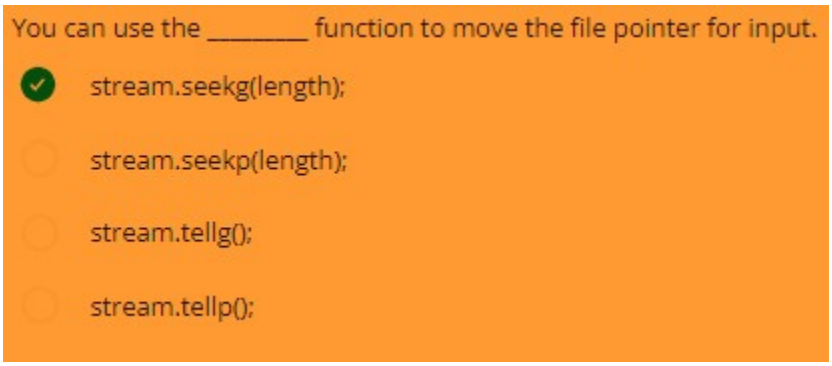
- Can also use the tellp and tellg fns to return the position of the file pointer in the file

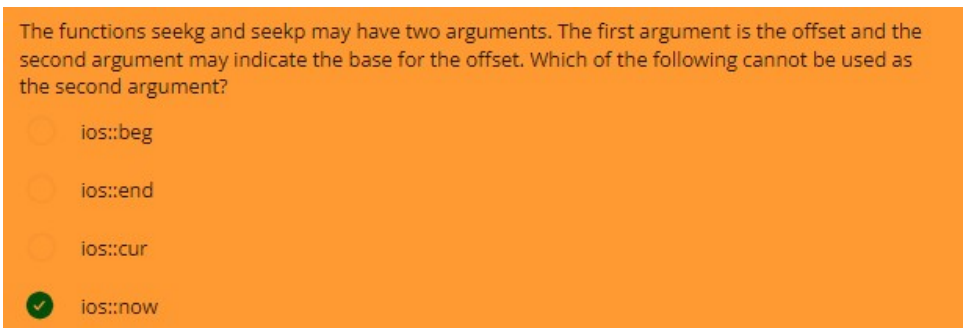
```
#include <iostream>
#include <fstream>
#include "Student.h"
using namespace std;
void displayStudent(const Student& student){
  cout << student.getFirstName() << " ";
  cout << student.getMi() << " ";
  cout << student.getLastName() << " ";
  cout << student.getScore() << endl;
}
int main(){
  fstream binaryio; // Create stream object
  binaryio.open("student.dat", ios::out | ios::binary);
  Student student1("FirstName1", 'A', "LastName1", 10);
  Student student2("FirstName2", 'B', "LastName2", 20);
  Student student3("FirstName3", 'C', "LastName3", 30);
  Student student4("FirstName4", 'D', "LastName4", 40);
  Student student5("FirstName5", 'E', "LastName5", 50);
  Student student6("FirstName6", 'F', "LastName6", 60);
  Student student7("FirstName7", 'G', "LastName7", 70);
  Student student8("FirstName8", 'H', "LastName8", 80);
  Student student9("FirstName9", 'I', "LastName9", 90);
  Student student10("FirstName10", 'J', "LastName10", 100);
  binaryio.write(reinterpret_cast<char*> (&student1), sizeof(Student));
  binaryio.write(reinterpret_cast<char*> (&student2), sizeof(Student));
  binaryio.write(reinterpret_cast<char*> (&student3), sizeof(Student));
  binaryio.write(reinterpret_cast<char*> (&student4), sizeof(Student));
  binaryio.write(reinterpret_cast<char*> (&student5), sizeof(Student));
  binaryio.write(reinterpret_cast<char*> (&student6), sizeof(Student));
  binaryio.write(reinterpret_cast<char*> (&student7), sizeof(Student));
  binaryio.write(reinterpret_cast<char*> (&student8), sizeof(Student));
  binaryio.write(reinterpret_cast<char*> (&student9), sizeof(Student));
  binaryio.write(reinterpret_cast<char*> (&student10), sizeof(Student));
  binaryio.close();
  // Read student back from the file
  binaryio.open("student.dat", ios::in | ios::binary);
  Student studentNew;
  // Move to the 3rd student
  binaryio.seekg(2 * sizeof(Student));
  cout << "Current position is " << binaryio.tellg()
    << endl;
  binaryio.read(reinterpret_cast<char*> (&studentNew), sizeof(Student));
  displayStudent(studentNew);
  cout << "Current position is " << binaryio.tellg() << endl;
  binaryio.close();
  return 0;}
```

```
command>cl RandomAccessFile.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>RandomAccessFile
Current position is 112
FirstName3 C LastName3 30
Current position is 168

command>
```

- The pgrm shows how to access a file rando, the pgrm first stores 10 student objects into the file and then retrieves the 3rd student from the file
- The pgrm makes a stream object, opens the file student.dat for binary output, makes 10 Student objects, writes them to the file, and closes the file
- The pgrm opens the file student.dat for binary input, makes a Student object using its no-arg construction, moves the file pointer to the address of the 3rd student in the file, the current pos is now 112
  - The sizeof(Student) is 56
- After the 3rd object is read, the file pointer is moved to the 4th object, so the current pos becomes 168







13.9 Updating Files

Sunday, April 09, 2023 9:37 PM

- U can update a binary file by opening a file using the mode ios::in | ios::out | ios::binary
- Often u need contents of the file, can open a file for both input and output, like
- ```
  binaryio.open("student.dat", ios::in | ios::out | ios::binary);
  ```
- This statement opens the binary file student.dat for both input and output
- Using code from b4, where student.dat was made by the code
- ```
  #include <iostream>
  #include <fstream>
  #include "Student.h"
  using namespace std;

  void displayStudent(const Student& student)
  {
    cout << student.getFirstName() << " ";
    cout << student.getMi() << " ";
    cout << student.getLastName() << " ";
    cout << student.getScore() << endl;
  }

  int main()
  {
    fstream binaryio; // Create stream object

    // Open file for input and output
    binaryio.open("student.dat", ios::in | ios::out | ios::binary);

    Student student1;
    binaryio.seekg(sizeof(Student)); // Move to the 2nd student
    binaryio.read(reinterpret_cast<char*>
      (&student1), sizeof(Student)); // Read the 2nd student
    displayStudent(student1);

    student1.setLastName("Yao"); // Modify 2nd student
    binaryio.seekp(sizeof(Student)); // Move to the 2nd student
    binaryio.write(reinterpret_cast<char*>
      (&student1), sizeof(Student)); // Update 2nd student in the file

    Student student2;
    binaryio.seekg(sizeof(Student)); // Move to the 2nd student
    binaryio.read(reinterpret_cast<char*>
      (&student2), sizeof(Student)); // Read the 2nd student
    displayStudent(student2);

    binaryio.close();

    return 0;
  }
  ```
- ```
  command>cl UpdateFile.cpp
  Microsoft C++ Compiler 2019
  Compiled successful (cl is the VC++ compile/link command)
  ```
- ```
  command>UpdateFile
  FirstName2 B LastName2 20
  FirstName2 B Yao 20

  command>
  ```
- The pgrm demonstrates how to update a file
- The pgrm makes a stream object, and opens the file student.dat for binary input and output
- The pgrm moves to the 2nd student in the file and reads the student, displays it, changes its last name, and writes the revised object back to the file
- The pgrm then moves to the second student in the file again and reads the student and displays it, u will see that the last name of this object has been changed in the sample output
-

- Can you open a file for both input and output?
  - Yes
  - No
-

Ch 13 Summary

Sunday, April 09, 2023 9:44 PM

1. C++ provides the classes ofstream, ifstream, and fstream for facilitating file input and output.
2. You can use the ofstream class to write data to a file, use ifstream to read data from a file, and use the fstream class to read and write data.
3. You can use the open function to open a file, the close function to close a file, the fail function to test whether a file exists, the eof function to test whether the end of the file is reached.
4. The stream manipulators (e.g., setw, setprecision, fixed, showpoint, left, and right) can be used to format output.
5. You can use the getline function to read a line from a file, the get function to read a character from a file, and the put function to write a character to a file.
6. The file open modes (ios::in, ios::out, ios::app, ios::trunc, and ios::binary) can be used to specify how a file is opened.
7. File I/O can be classified into text I/O and binary I/O.
8. Text I/O interprets data in sequences of characters. How text is stored in a file is dependent on the encoding scheme for the file. C++ automatically performs encoding and decoding for text I/O.
9. Binary I/O interprets data as raw binary values. To perform binary I/O, open the file using the iso::binary mode.
10. For binary output, use the write function. For binary input, use the read function.
11. You can use the reinterpret_cast operator to cast any type of data into an array of bytes for binary input and output.
12. You can process a file sequentially or in a random manner.
13. The seekp and seekg functions can be used to move the file-access pointer anywhere in the file before invoking the put/write and get/read functions.