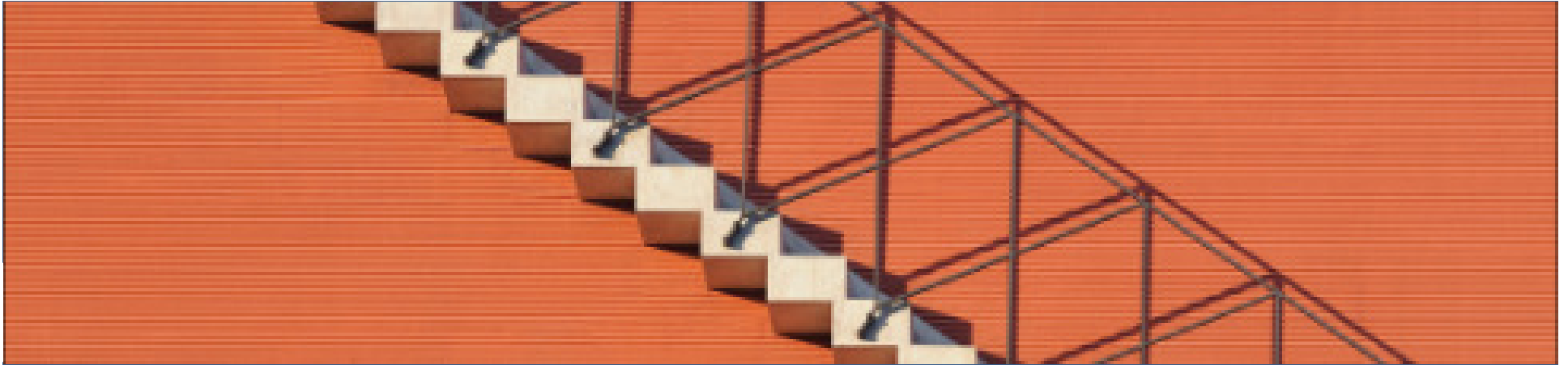




Module 5

File processing

Objectives

- Become familiar with file input and output
- Text file processing
- Binary file processing

Text file processing

- File: area in secondary storage to hold info
- File I/O is a five-step process
 1. Include `fstream` header
 2. Declare file stream variables
 3. Associate the file stream variables with the input/output sources
 4. Use the file stream variables with `>>`, `<<`, or other input/output functions
 5. Close the files

File I/O in C++

- In C++, a *stream* is a special kind of variable known as an *object*.
- We will describe how our program can use *stream objects* to do simple file I/O.
- If you want to use a stream to get the input from a file (or give the output to a file), *you must declare the stream and you must connect the stream to the file.*

File I/O in C++ (cont'd)

- The type for input-stream variables is named `ifstream` (for “input-file stream”)
- The type for output-stream variables is named `ofstream` (for “output-file stream”)
- Need to `#include <fstream>`

File I/O in C++ (int)

/* Read Text file (int) in C++ */

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

```
int main () {
    ifstream readfile ("infile.txt") ;
    // Or alternatively,
    // ifstream readfile;
    // readfile.open ("infile.txt");
```

```
    ofstream writefile ("outfile.txt");
```

```
    int x;
```

```
    while (readfile >> x)
        writefile << x << "***";
```

```
    writefile << endl;
```

```
}
```

infile.txt

1	2	3
4	5	
6	7	8

outfile.txt

1**2**3**4**5**6**7**8**

File I/O in C++ (**char**)

/* Read Text file (**char**) in C++ */

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

```
int main () {
    ifstream readfile ("infile.txt") ;
    // Or alternatively,
    // ifstream readfile;
    // readfile.open ("infile.txt");

    ofstream writefile ("outfile.txt");
```

char x;

```
while (readfile >> x)
    writefile << x;

writefile << endl;
}
```

infile.txt

```
1 2 3
4 5
6 7 8
```

outfile.txt

12345678

**All spaces in
infile are
ignored**



File I/O in C++ (**char** again)

/* Read Text file (**char**) in C++ */

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

```
int main () {
    ifstream readfile ("infile.txt");
    ofstream writefile ("outfile.txt");
```

```
    char x;
```

```
    while (readfile.get (x)) {
        if (isspace (x))
            writefile << '*';
        else
            writefile << x;
    }
```

```
    writefile << endl;
}
```

infile.txt

```
1 2 3
4 5
6 7 8
```

outfile.txt

```
1*2*3*4*5*6*7*8*
```

**eol and eof are
considered as
spaces too**



File I/O in C++ (**string or c-string**)

/* Read Text file (**string**) in C++ */

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

```
int main () {
    ifstream readfile ("infile.txt");

    ofstream writefile ("outfile.txt");
```

```
    string x;
```

```
    while (readfile >> x)
        writefile << x << endl;
```

```
}
```

infile.txt

111 222 3333
444 555
666 7777 888

Outfile.txt

111
222
3333
444
555
666
7777
888

**Again, spaces
are delimiters**



File I/O in C++ (**getline function**)

/* Read Text file (**string**) in C++ */

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

```
int main () {
    ifstream readfile ("infile.txt") ;

    ofstream writefile ("outfile.txt");

    string x;

    while (getline (readfile, x))
        writefile << x << endl;

}
```

infile

111 222 3333
444 555
666 7777 888

outfile

111 222 3333
444 555
666 7777 888

readfile.getline (x, MAX); // if c string

Open a File in C++

- **File Open Options:**
- **ios::app** (append) Seek to the end of the stream before each output operation.
- **ios::ate** (at end) Seek to the end of the stream when opening.
- **ios::binary** Consider stream as binary rather than text.
- **ios::in** Allow input operations on a stream.
- **ios::out** Allow output operations on a stream.
- **ios::trunc** (truncate) Truncate file to zero when opening.

Examples

```
fstream ins("data.dat", ios::in | ios::binary);  
// open an input stream attached to the file data.dat  
// in binary mode.
```

```
fstream outs("data.txt", ios::out | ios::trunc);  
// open an output stream attached to the file data.txt.  
// The file upon open is to be truncated (erased).
```

```
fstream outs("data.txt", ios::out | ios::app);  
// open an output stream attached to a file called data.txt.  
// Upon open place the write marker so the next write occurs  
// at the end of the file.
```

#include <fstream>	Use
ios::out	“write”
ios::out ios::app	“append”
ios::out ios::trunc	“write”
ios::in	“read”
ios::in ios::out	“read/write” (update)
ios::in ios::out ios::trunc	“read/write”
ios::out ios::binary	“write binary”
ios::out ios::app ios::binary	“append binary”
ios::out ios::trunc ios::binary	“write binary”
ios::in ios::binary	“read binary”
ios::in ios::out ios::binary	“read/write binary”
ios::in ios::out ios::trunc ios::binary	“read/write binary”

Reading and Writing Binary Files

- Read and write a block of data from/to a file
- Use member functions

```
istream& read(reinterpret_cast <char*> buf,  
               streamsize size);
```

```
ostream& write(reinterpret_cast <const char*> buf,  
               streamsize size);
```

Binary Files in C++

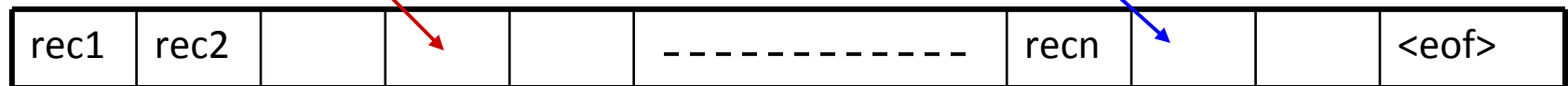
```
fstream fio;  
fio.open("...", ...);
```



one record

```
fio.read(...)
```

```
fio.write(...)
```



records

read position
(get) indicator

write position
(put) indicator

```
Pos = fio.tellg()  
fio.seekg(offset, whence)
```

```
Pos = fio.tellp()  
fio.seekp(offset, whence)
```

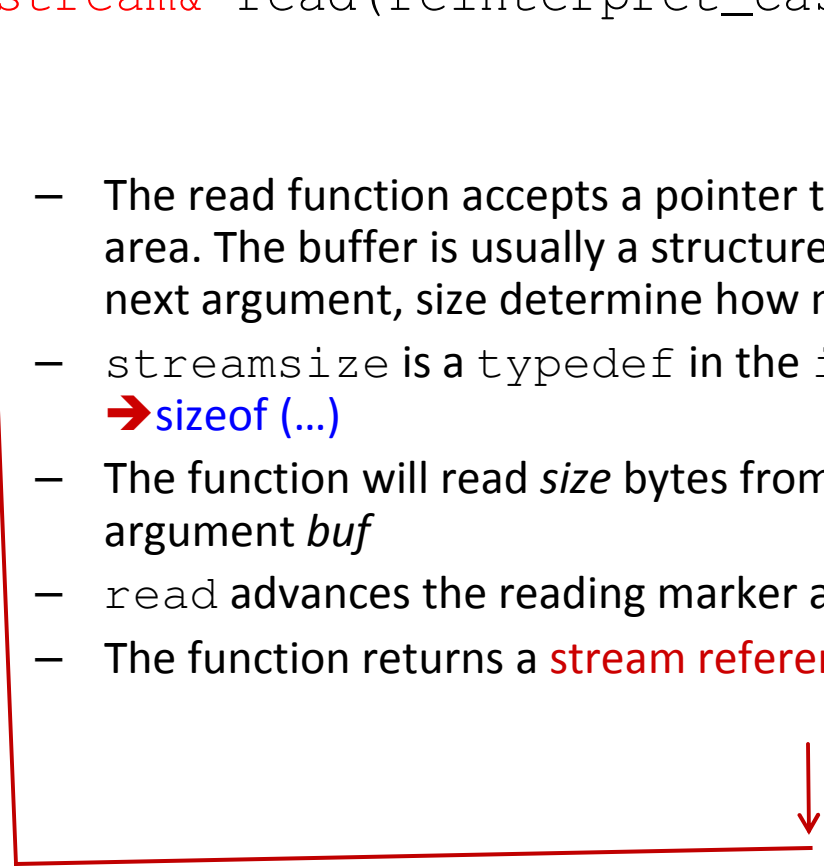
Will discuss later how to use them

Reading a Block

Reference



```
istream& read(reinterpret_cast <char *> (&buf),  
               streamsize size);
```

- The read function accepts a pointer to a buffer, which it will use as its input area. The buffer is usually a structure or class. It could even be a basic type. The next argument, size determine how many bytes are to be read into the stream.
 - `streamsize` is a typedef in the `istream` header. It is effectively an int
→ `sizeof (...)`
 - The function will read *size* bytes from the stream and place them into the argument *buf*
 - `read` advances the reading marker and set the stream status.
 - The function returns a **stream reference!!!**
- 

For example; imagine we have a file of characters and we wish to read them using unformatted I/O calls. Assume the name of the file is "input.dat".

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    char x;
    ifstream fin("input.dat", ios::in | ios::binary);
    if (fin.good())
    {
        while(!fin.eof())
        {
            fin.read(reinterpret_cast <char*>(&x),
                    sizeof(char));
            cout << x;
        }
        fin.close();
    }
    return 0;
}
```

Open the file in input and binary mode and associate it with the stream ins.

check if the stream is opened successfully or not

Check the state of the stream

Using unformatted I/O read sizeof(char) bytes from the stream and place the bytes into the buffer x.

Lets read a structure in now;

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

struct details
{
    char name[120];
    int age;
};

int main()
{
    details x;
    ifstream fin("struct.dat", ios::in | ios::binary);
    if (fin.good())
    {
        while(!fin.eof())
        {
            fin.read(reinterpret_cast <char*>(&x),
                    sizeof(details));
            cout << x.name << endl;
            cout << x.age << endl;
        }
        fin.close();
    }
    return 0;
}
```

Writing a Block

```
ostream& write (reinterpret_cast <const char*> (&buf)  
               , streamsize size);
```

- It takes a `const char*` to the buffer, that is what is to be written to the file and it takes the number of bytes that are to be written (this is normally the size of the `buf` variable).
- The function advances the writer marker and set the stream status
- returns the **reference** to the output stream

Example

- Consider the following problem, a programmer wants to store 5 records. Each record contains information about his/her friends.

```
struct friends
{
    char name[64];
    long phone;
    char email[128];
};
```

We want to store, the name, phone number and email address. This is a fixed size record (it contains no dynamics & no string types)

Note: It can be a class as well

- To write 5 records to the file we would do this;

- ```
int main()
{
 friends x;
 ofstream fout("friends.dat", ios::out | ios::binary);
 if (fout.good())
 {
 for (int i = 0; i < 5; i++)
 {
 // get details
 cout << "Name: ";
 cin >> x.name;
 cout << "Phone: ";
 cin >> x.phone;
 cout << "Email: ";
 cin >> x.email;

 // write the structure to the file
 fout.write(reinterpret_cast <const char*>(&x),
 sizeof(friends));
 }
 fout.close();
 }
 return 0;
}
```

- To read 5 records from the file we would do this:

```
int main()
{
 friends x;
 ifstream fin("friends.dat", ios::out | ios::binary);
 if (fin.good())
 {
 for (int i = 0; i < 5; i++)
 {
 // read the structure from the file
 fin.read(reinterpret_cast<char*>(&x),
 sizeof(friends));

 // print details
 cout << "Name: " << x.name << endl;
 cout << "Phone: " << x.phone << endl;
 cout << "Email: " << x.email << endl << endl;

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

# Comments

- Because I know the file is always going to have five records, I could just read or write an array of five records in one hit;

I would define this array:

```
friends myarray[5];
```

To read all five records into this array I would:

```
fin.read(reinterpret_cast<char*> (myarray),
 sizeof(friends) * 5);
```

To write the array of five records to the stream I would do:

```
fout.write(reinterpret_cast<const char*> (myarray),
 sizeof(friends) * 5);
```

**Let us now look  
at workshop 5  
question**

