First Year First Semester Course

M.Tech. (CS) [Batch 2021-23]

Lecture #19

Introduction to Programming

C++: Basics of File Handling, Exception Handling

Course Instructor:

Dr. Monidipa Das

DST-INSPIRE Faculty

Machine Intelligence Unit (MIU), Centre for Artificial Intelligence and Machine Learning (CAIML)

Indian Statistical Institute (ISI) Kolkata, India

Basics of File Handling in C++

Introduction

- A file is a collection of related data stored in a particular area on the disk.
- Files in C++ are interpreted as a ***sequence of bytes*** stored on some storage media.
- The flow of data from any source to a sink is called as a ***stream***
- In C++ this is achieved through a component header file called ***fstream.h***

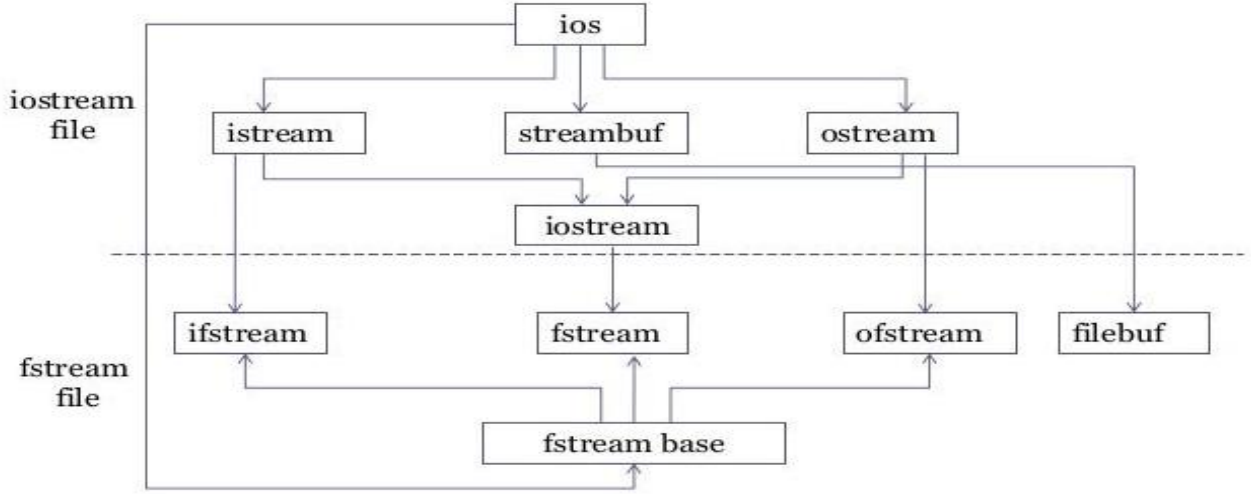
Streams

- These represent as a sequence of bytes and deals with the flow of data.
- In C++ a stream is used to refer to the flow of data from a particular device to the program's variables
  - File -> Program (Input stream) - reads
  - Program -> File (Output stream) – write

- interactive (iostream)
  - **cin** - input stream associated with keyboard.
  - **cout** - output stream associated with display

- file (fstream)
  - **ifstream** - defines new input stream (normally associated with a file).
  - **ofstream** - defines new output stream (normally associated with a file).

Classes for Stream I/O in C++



Class	Description
`ofstream`	Creates and writes to files
`ifstream`	Reads from files
`fstream`	A combination of ofstream and ifstream: creates, reads, and writes to files

Opening a File

- We can open a file using any one of the following methods:

1. using the open() function.
2. by passing the file name in constructor at the time of object creation.

Modes	*Description*
in	Opens the file to read(default for ifstream)
out	Opens the file to write(default for ofstream)
binary	Opens the file in binary mode
app	Opens the file and appends all the outputs at the end
ate	Opens the file and moves the control to the end of the file
trunc	Removes the data in the existing file
nocreate	Opens the file only if it already exists
noreplace	Opens the file only if it does not already exist

void open(const char* file_name,ios::openmode mode);

```
fstream new_file;
new_file.open("newfile.txt", ios::out);
```

Opening /Closing a File

- We can combine the different modes using or symbol | .
- **Example**

  ofstream new_file;

  new_file.open(“new_file.txt”,
  ios::out | ios::app);

- Close the file using close() function

```
#include<iostream>
#include <fstream>
using namespace std;
int main()
{
 fstream new_file;
 new_file.open("new_file.txt",ios::out);
 if(!new_file) {
  cout<<"File creation failed";
 }
 else{
  cout<<"New file created";
  new_file.close();
 }
 return 0;
}
```

Writing to a File


```
#include <iostream>
#include <fstream>
using namespace std;
int main(){
  fstream new_file;
  new_file.open("new_file_write.txt",ios::out);
  if(!new_file) {
    cout<<"File creation failed";
  }
  else{
    cout<<"New file created";
    new_file<<"This is my first code on writing in a File...";
    new_file.close();}
  return 0;
}
```

Reading from a File


```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  fstream new_file;
  new_file.open("new_file_write.txt",ios::in);
  if(!new_file)
     cout<<"No such file";
  else {
    char ch;
    while (new_file.get(ch)) { cout << ch; }
  new_file.close();}
  return 0;
}
```

read(): for block reading
write(): for block writing
put(): byte writing

File Opening using Constructor

- Examples

```
#include <fstream>
using namespace std;
int main(void)
{
    ofstream outFile("fout.txt");
    outFile << "Hello World!";
    outFile.close();
    return 0;
}
```

```
#include <iostream>
#include <fstream>
using namespace std;
int main(void)
{
    ifstream openFile("fout.txt");
    char ch;
    while(!openFile.eof())
    {
        openFile.get(ch);
        cout << ch;
    }
    openFile.close();
    return 0;
}
```

Some useful functions

- **seekg():** Go to a specific position when reading
- **seekp():** Go to a specific position when writing
- **tellg():** Gives the current position of the get pointer
- **tellp():** Gives the current position of the put pointer

Basics of Handling Exception in C++

Exception

- Exception refers to unexpected condition in a program

- Exception handling
  - Can resolve exceptions
    - Allow a program to continue executing or
    - Notify the user of the problem and
    - Terminate the program in a controlled manner
  - Makes programs robust and fault-tolerant

Exception Handling in C++

- Use three constructs:
  throw, try, and ***catch***

- ***throw construct:*** used to raise an exception when an error is generated in the computation.

- ***try construct:*** defines a boundary within which an exception can occur. A block of code in which an exception can occur must be prefixed by the keyword try.

- ***catch construct:*** exception handler. Used immediately after the statements marked by the try keyword.

Example


```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    int a,b;
    cout<<"Enter two numbers: ";
    cin>>a>>b;
    try{
          if (b==0)
                throw b;
          else
               cout<<float(a)/b;
    }
    catch(int x){
          cout<<"2nd operand can't be 0";
    }
    return 0;
}
```

```
#include <iostream>
#include <stdexcept>
using namespace std;
int AddPositiveIntegers(int a, int b){
    if (a < 0 || b < 0)
        throw
    invalid_argument("AddPositiveIntegers:
    arguments must be positive");
    return (a + b);
}
int main(){
    try{
        cout << AddPositiveIntegers(-4, 2);
    }
    catch (invalid_argument& e){
        cout<<e.what();
        return -1;
    }
    return 0;
}
```

Standard Exceptions

Logic errors

logic_error
Logic error exception

domain_error
Domain error exception

invalid_argument
Invalid argument exception

length_error
Length error exception

out_of_range
Out-of-range exception

Runtime errors

runtime_error
Runtime error exception

range_error
Range error exception

overflow_error
Overflow error exception

underflow_error
Underflow error exception

Exception Handling in C++ [contd.]

- Multiple handlers (i.e., catch expressions) can be chained; each one with a different parameter type.
- If an ellipsis (...) is used as the parameter of catch, that handler will catch any exception no matter what the type of the exception thrown.

```
try {
  // code here
}
catch (int param) { cout << "int exception"; }
catch (char param) { cout << "char exception"; }
catch (...) { cout << "default exception"; }
```

Exception Handling in C++ [contd.]

- After an exception has been handled the program, execution resumes after the *try-catch* block, not after the throw statement!.
- It is also possible to nest try-catch blocks within more external try blocks

```
try {
            try {
              // code here
            }
            catch (int n)
            {
                  throw;
            }
}
catch (...)
 {
          cout << "Exception occurred";
 }
```

Exception Handling in C++ [contd.]


```
#include <iostream>
#include <exception>
using namespace std;
class myexception: public exception{
  virtual const char* what() const throw(){
    return "My exception happened";
  }
} myex;
int main () {
  try
  {
    throw myex;
  }
  catch (exception& e)
  {
    cout << e.what() << '\n';
  }
  return 0;
}
```

```
// bad_alloc standard exception
#include <iostream>
#include <exception>
using namespace std;

int main () {
  try
  {
    int* myarray= new int[10000000000];
  }
  catch (exception& e)
  {
    cout << "Standard exception: " << e.what()
    <<"\n";
  }
  cout<<"This statement would still execute.";
  return 0;
}
```

Questions?