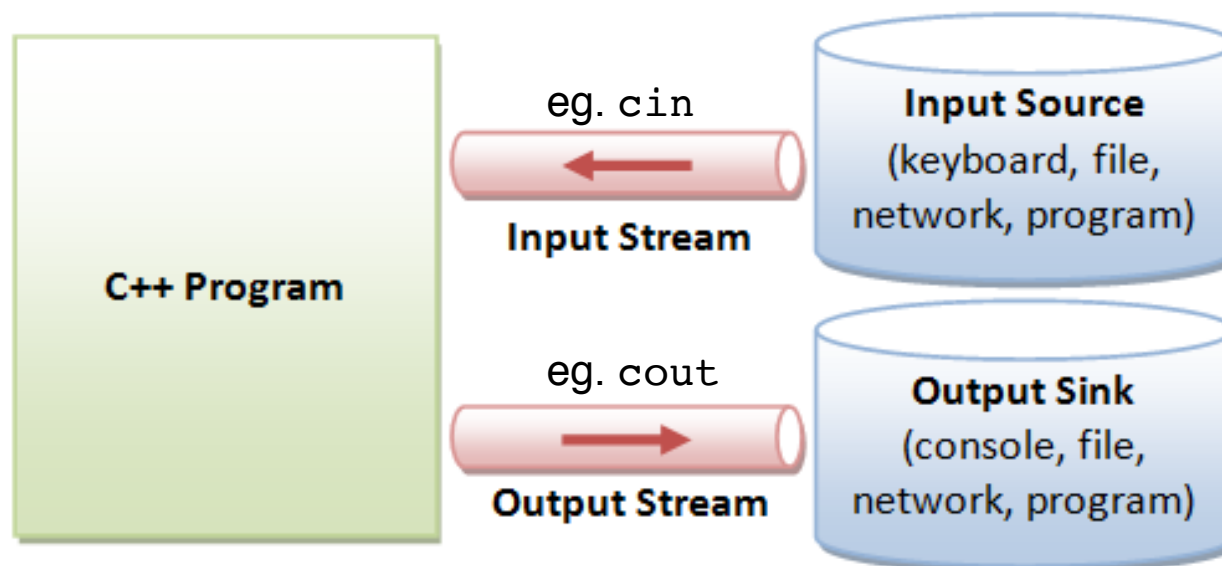
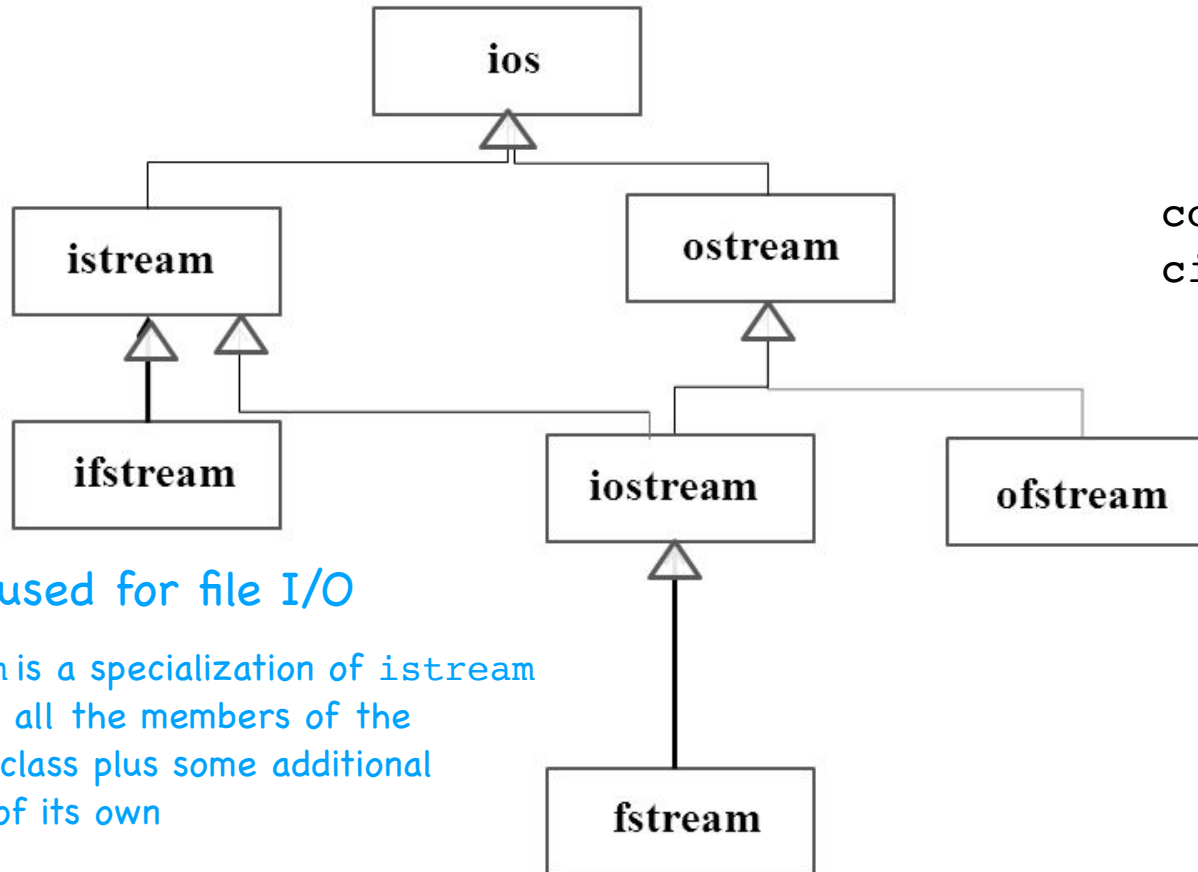


- The C language did not build the input/output facilities into the language.
- There are no keywords like `read` or `write`.
- Instead, it left I/O to C standard library functions such as `printf` and `scanf`, declared in `stdio.h`
- C++ continues this approach and formalizes I/O in header files such as `iostream` and `fstream`, all part of the C++ standard library.
- Stream classes allow us to create stream objects that are abstractions of sources or sinks of data.



C++ Stream Class Hierarchy (simplified)

- part of the C++ standard library

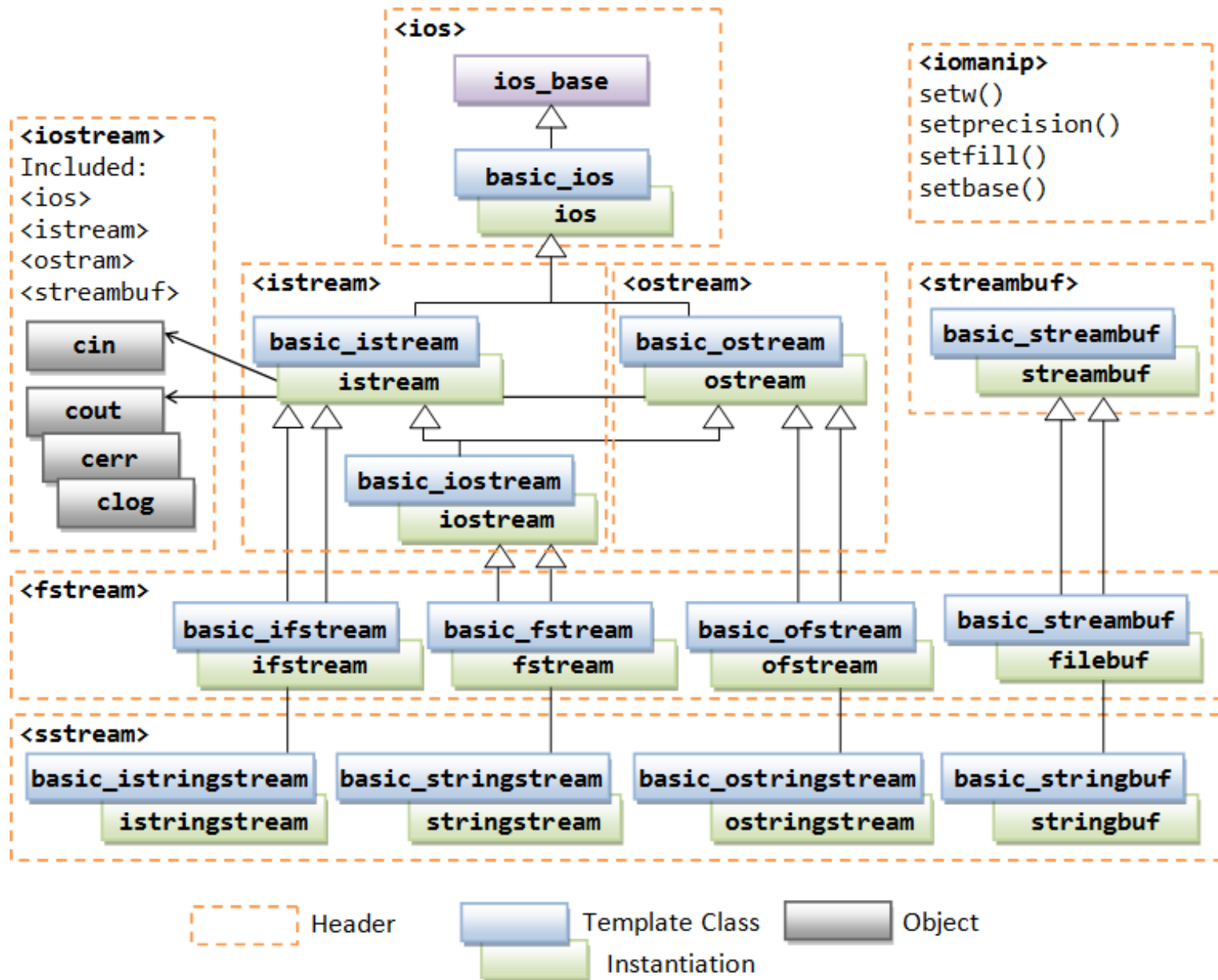


cout is an object of class ostream
cin is an object of class istream

classes used for file I/O

ifstream is a specialization of istream
it includes all the members of the
istream class plus some additional
members of its own

Wow



We can use file streams in the same way as `cin` and `cout` but we need to associate these streams with files in the filesystem.

The C++ standard library provides the following classes to perform file I/O:

`ofstream`: stream class to write to files
`ifstream`: stream class to read from files
`fstream`: stream class to both read and write files.

```
#include <iostream>
#include <fstream>

using namespace std;

int main ()
{
    ofstream myfile;
    myfile.open("/Users/joshelliott/Documents/fluffy.txt");
    if (myfile.is_open())
    {
        cout << "file opened\n";
        myfile << "Hello, line 1.\n";
        myfile << "Hello, line 2.\n";
        myfile.close();
    }
    else
        cout << "Unable to open file";

    return 0;
}
```

```
#include <iostream>
#include <fstream>

using namespace std;

int main ()
{
    string line;
    ifstream myfile("/Users/joshelliott/Documents/fluffy.txt");
    if (myfile.is_open())
    {
        while (getline( myfile, line ) )
            cout << line << '\n';
        myfile.close();
    }
    else
        cout << "Unable to open file";

    return 0;
}
```

Exception handling in C++

provides a way to handle unexpected events like runtime errors

```
try
{
    // the protected code
}
catch( Exception_Name exception1 )
{
    // catch block
}
catch( Exception_Name exception2 )
{
    // catch block
}
```


Exceptions are preferred in modern C++ for the following reasons:

- An exception forces calling code to recognize an error condition and handle it. Unhandled exceptions stop program execution.
- An exception jumps to the point in the call stack that can handle the error. Intermediate functions can let the exception propagate. They don't have to coordinate with other layers.
- The exception **stack-unwinding mechanism** destroys all objects in scope after an exception is thrown, according to well-defined rules.
- An exception enables a clean separation between the code that detects the error and the code that handles the error.

```
#include<iostream>
#include<vector>

using namespace std;

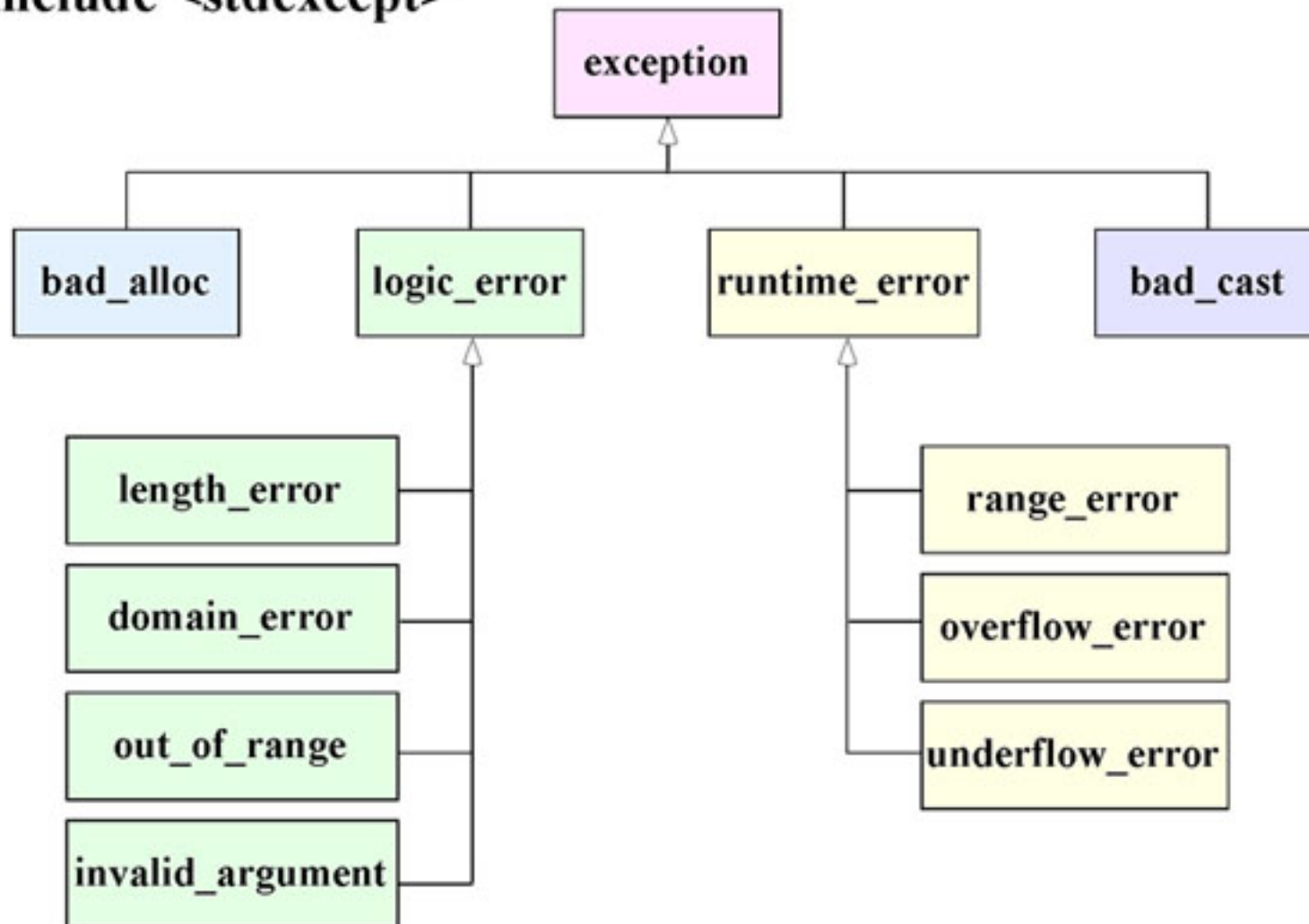
int main()
{
    vector<int> vec;
    vec.push_back(99);
    vec.push_back(56);

    // access the third element, which doesn't exist
    try
    {
        vec.at(2);
    }
    catch (exception& ex) 
    {
        cout << "Exception occurred!" << endl;
    }
    return 0;
}
```

Exceptions

C++ Exception Classes

`#include <stdexcept>`




```

#include <iostream>

using namespace std;

double zeroDivision(int x, int y)
{
    if (y == 0)
        throw string("Division by Zero!");
    return (x / y);
}

int main()
{
    int a = 11;
    int b = 0;
    double c = 0;

    try
    {
        c = zeroDivision(a, b);
        cout << c << endl;
    }
    catch (string message)
    {
        cerr << message << endl;
    }
    return 0;
}

```

You can use an object of any type as the operand of a `throw` expression. Typically, this object is used to communicate information about the error. In most cases, we recommend that you use the `std::exception` class or one of the derived classes that are defined in the standard library. If one of those isn't appropriate, we recommend that you derive your own exception class from `std::exception`.

```
#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;
}
```

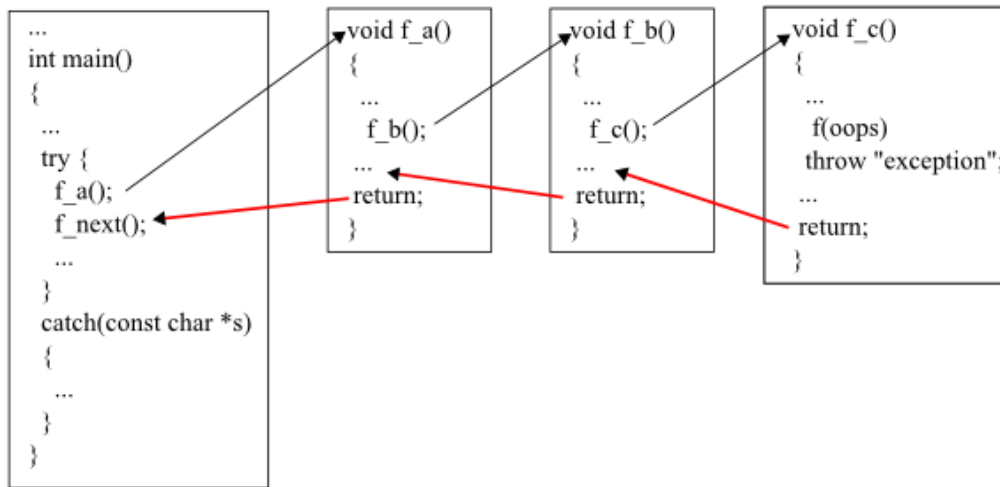
The C++ Standard library provides a base class specifically designed to declare objects to be thrown as exceptions. It is called [`std::exception`](#) and is defined in the [`<exception>`](#) header.

This class has a virtual member function called `what` that returns a null-terminated character sequence (of type `char *`) and that can be overwritten in derived classes to contain some sort of description of the exception.

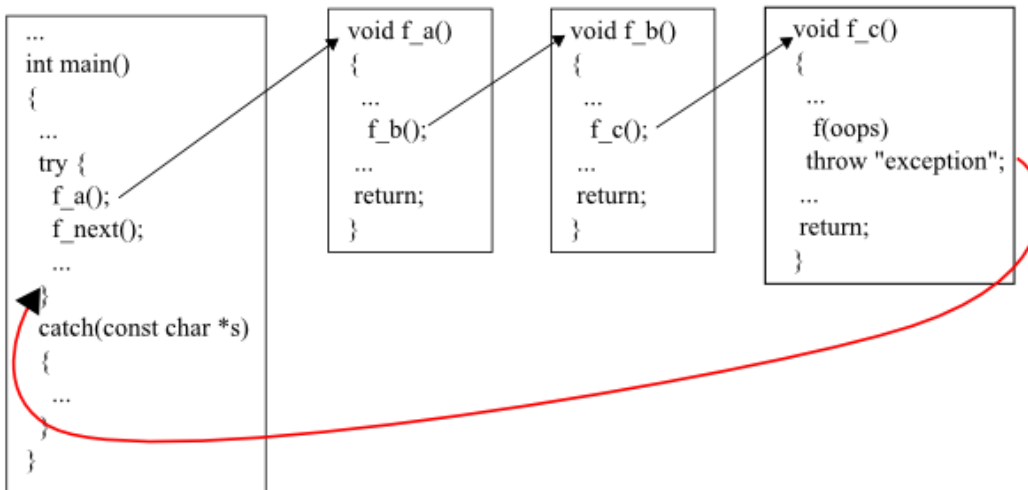
Regarding the `const throw()` part:

- `const` means that this function (which is a member function) will not change the observable state of the object it is called on. The compiler enforces this by not allowing you to call non-`const` methods from this one, and by not allowing you to modify the values of members.
- `throw()` means that you promise to the compiler that this function will never allow an exception to be emitted. This is called an *exception specification*, and (long story short) is useless and possibly misleading.

Stack Unwinding when an Exception Occurs



no exception



exception occurs...

The stack is searched for the 'catch' and other frames are popped.

The compiler inserts calls to the destructors of stack variables so that resources are properly freed.