UNIT V

GENERIC PROGRAMMING WITH TEMPLATES

- Templates supports generic programming.
- Reusable components can be developed.
- Template declared for functions

Function template

- Template declared for class

Class Template

Function Template

- Drawback of a simple function is that they can be used with only a particular data type.
- This can be overcome by function template or generic functions.

Syntax of function template

Keyword

Template datatype

At least one argument must be template type

template <class T,……….>

Returntype Functionname(arguments)

{

……………………………………

……………………………………

}

Overriding of function templates

- Function templates can be overridden by normal functions.
- If the program has both the function and function template with the same name, the compiler first selects the normal function.

Errors while using function template

- No-argument template function.
- Template-type argument unused.
- Usage of partial number of template arguments.

Overloaded Function Template

- The function templates can also be overloaded with multiple declarations.
- Must differ either in number or type of arguments.

Multiple arguments function template

- Multiple generic arguments can also be taken.

User defined template arguments

Class Template

- Class can also be declared to work on different data types.
- This generic class will support similar operations for different data types.

Syntax of class template

Keyword

Template datatypes
T1,T2,....

```
template <class T1,class T2,......>
class  ClassName
{
 T1  data;
  ..........
void func1(T1 a, T2 b);
  ................
};
```

Syntax for class template instantiation

Datatype to be substituted for template datatype

- ClassName <char> object1;
- ClassName <int> object2;

Template arguments

- A template can have character strings, function names etc as template type arguments.
- Example:

```
Template <class T1, int size>
Class myClass
{
 T arr[size];
};
```

- The object of the class will be created as:
- myClass <float,10> new1;

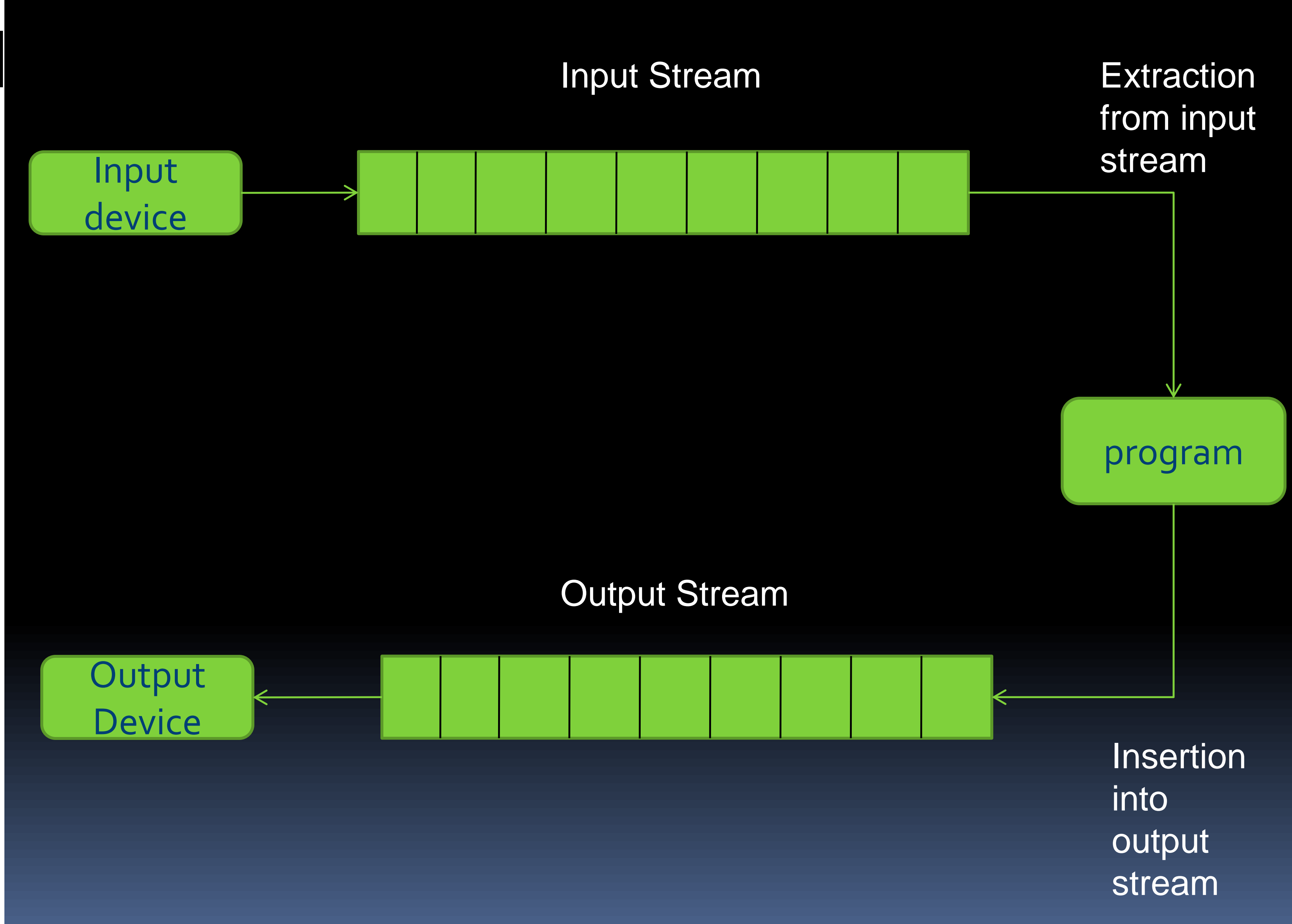
Member function definition outside the class

CONSOLE I/O OPERATION

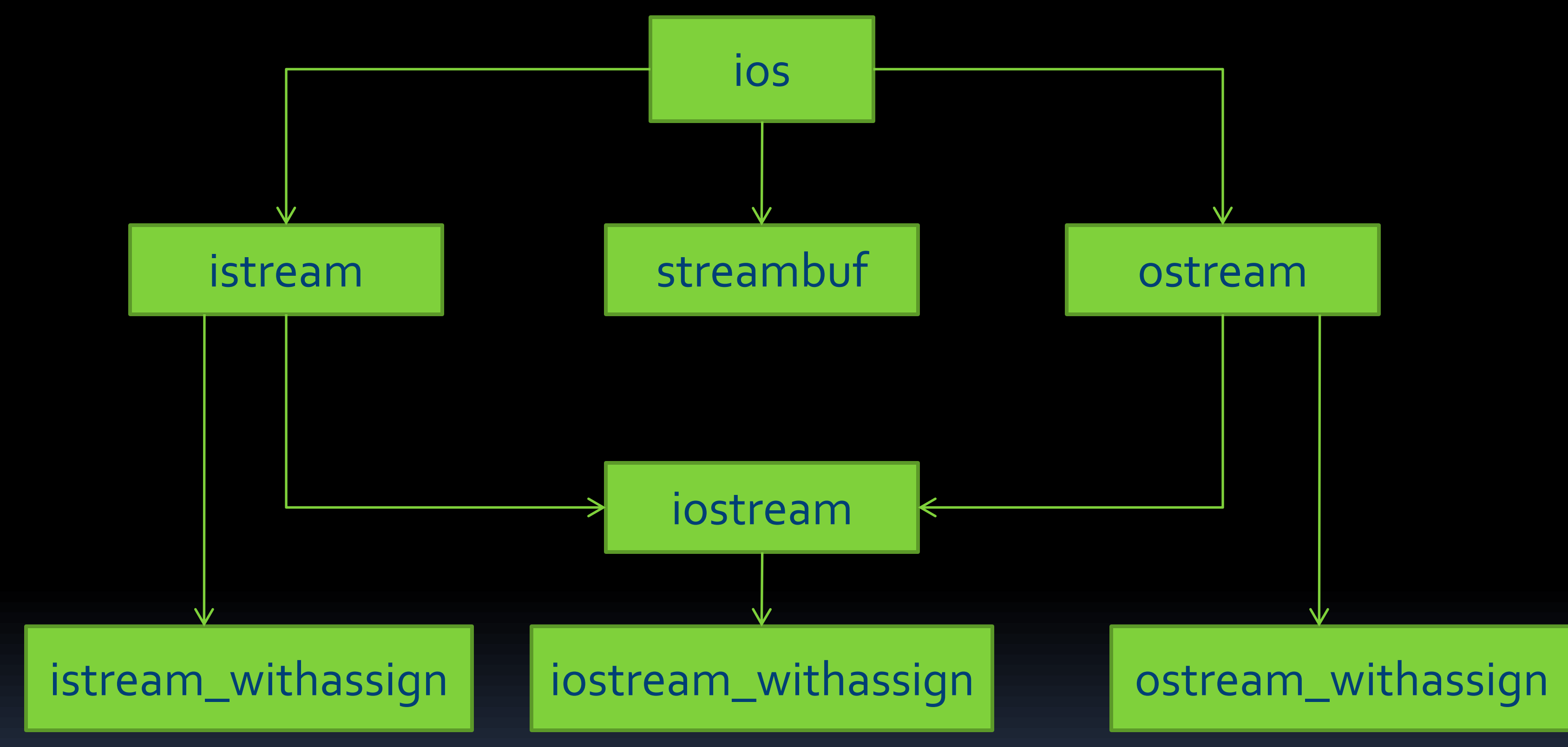
- C++ uses concept of streams and stream classes to implement I/O operation with console and disk files.

C++ Streams

- I/O system in C++ supplies an interface to the user that is independent of device being used.
- This interface is called streams.
- Stream is a sequence of bytes.


Input Stream
Extraction from input stream
Input device
program
Output Stream
Output Device
Insertion into output stream

C++ Stream Classes



ios

- Base class for istream and ostream
- Declares constants and functions that are necessary for handling formatted input and output operations.

istream

- Inherits ios
- Declares input functions such as get(), getline() and read()
- Conatins overloaded insertion operator

ostream

- Inherits ios
- Declares output functions put() and write()

iostream

- Inherits ios, istream, ostream
- Contains all the input and output functions

Unformatted I/O operations

- put() & get() functions
  - Handle the single character input//output operations.
- getline() & write() functions
  - Line oriented input/output functions.

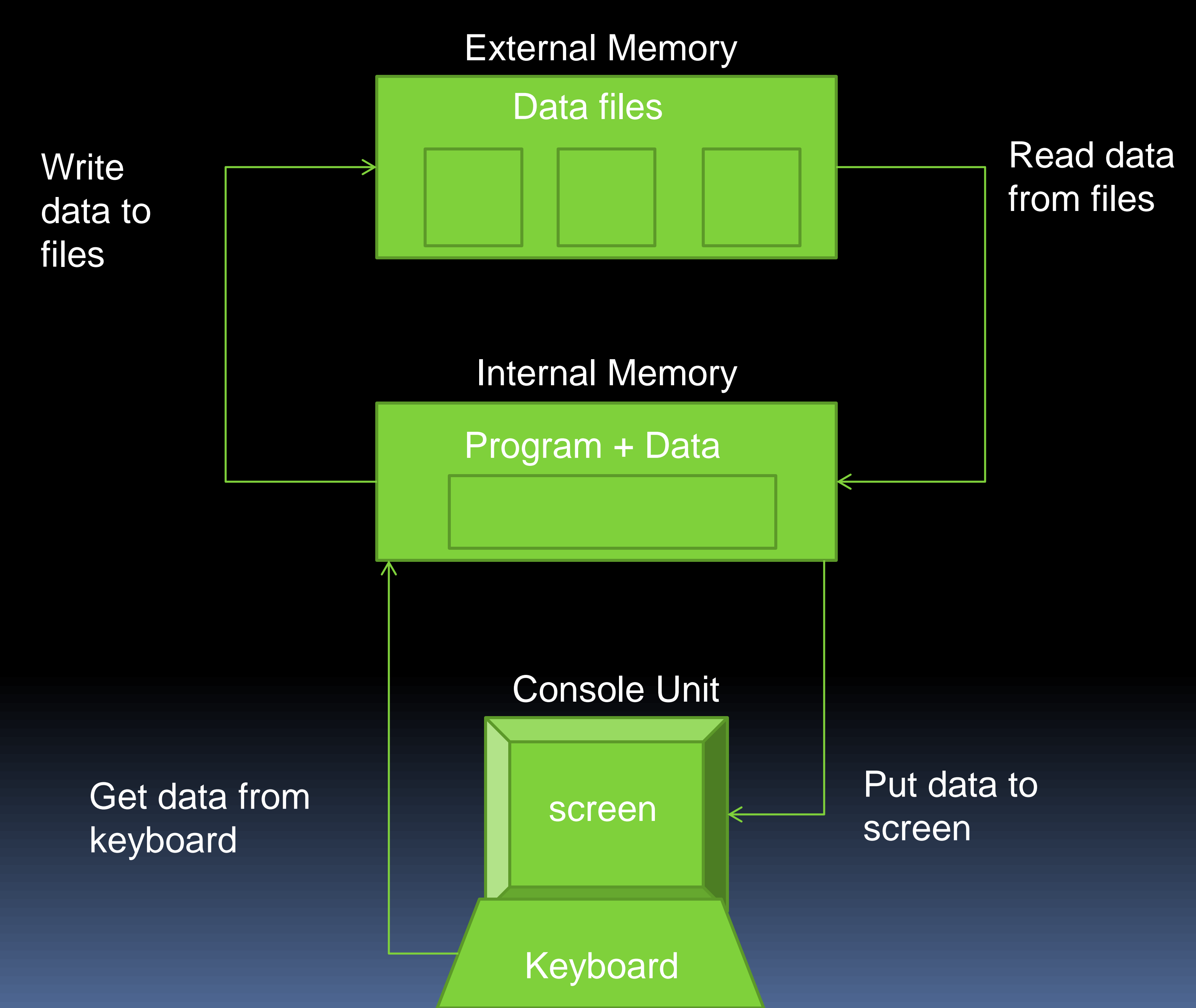
Formatted Console I/O Operations

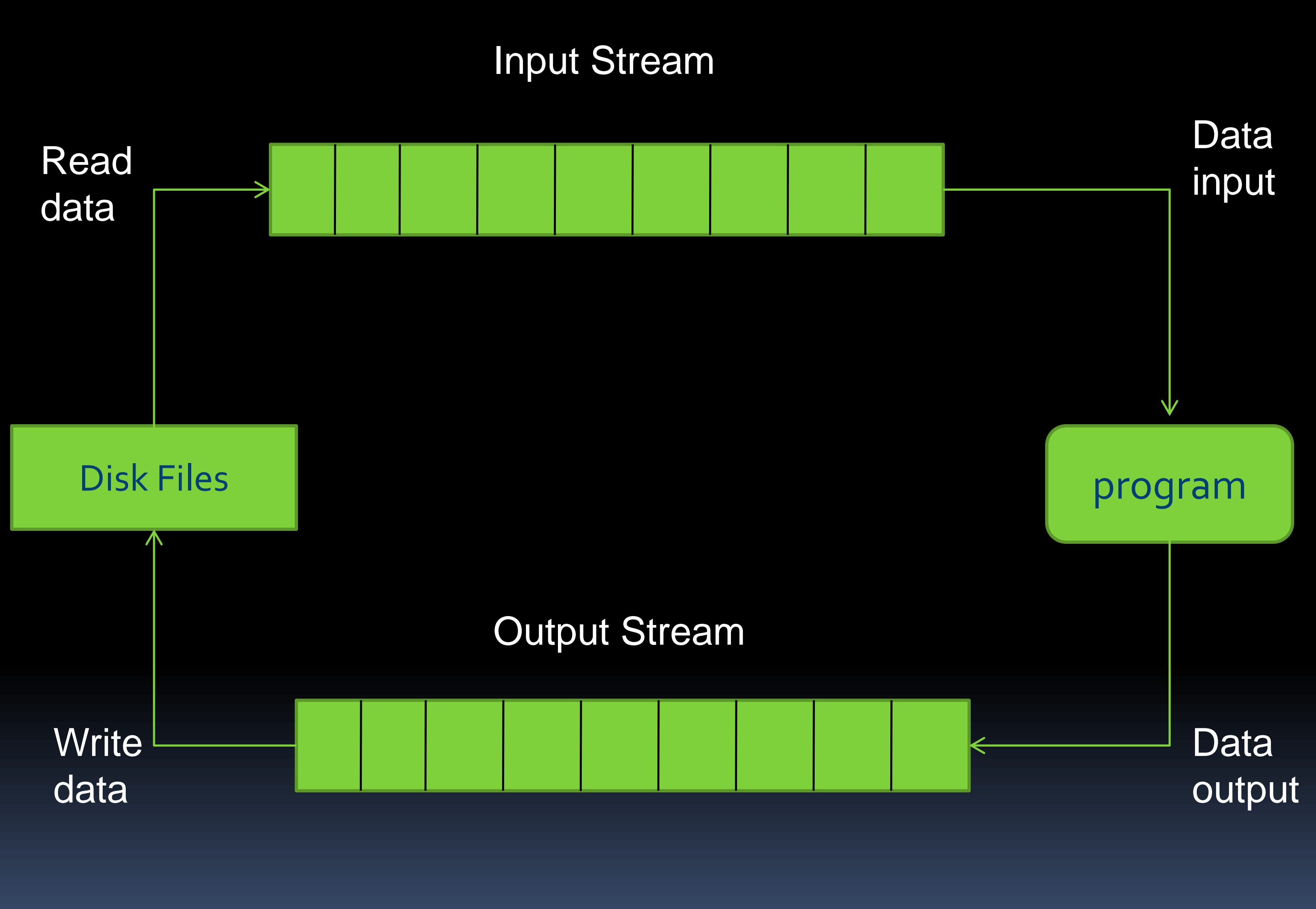
- Functions
  - width()
  - precision()
  - fill()
  - setf()
  - unsetf()
- Manipulators
  - setw()
  - setprecision()
  - setfill()

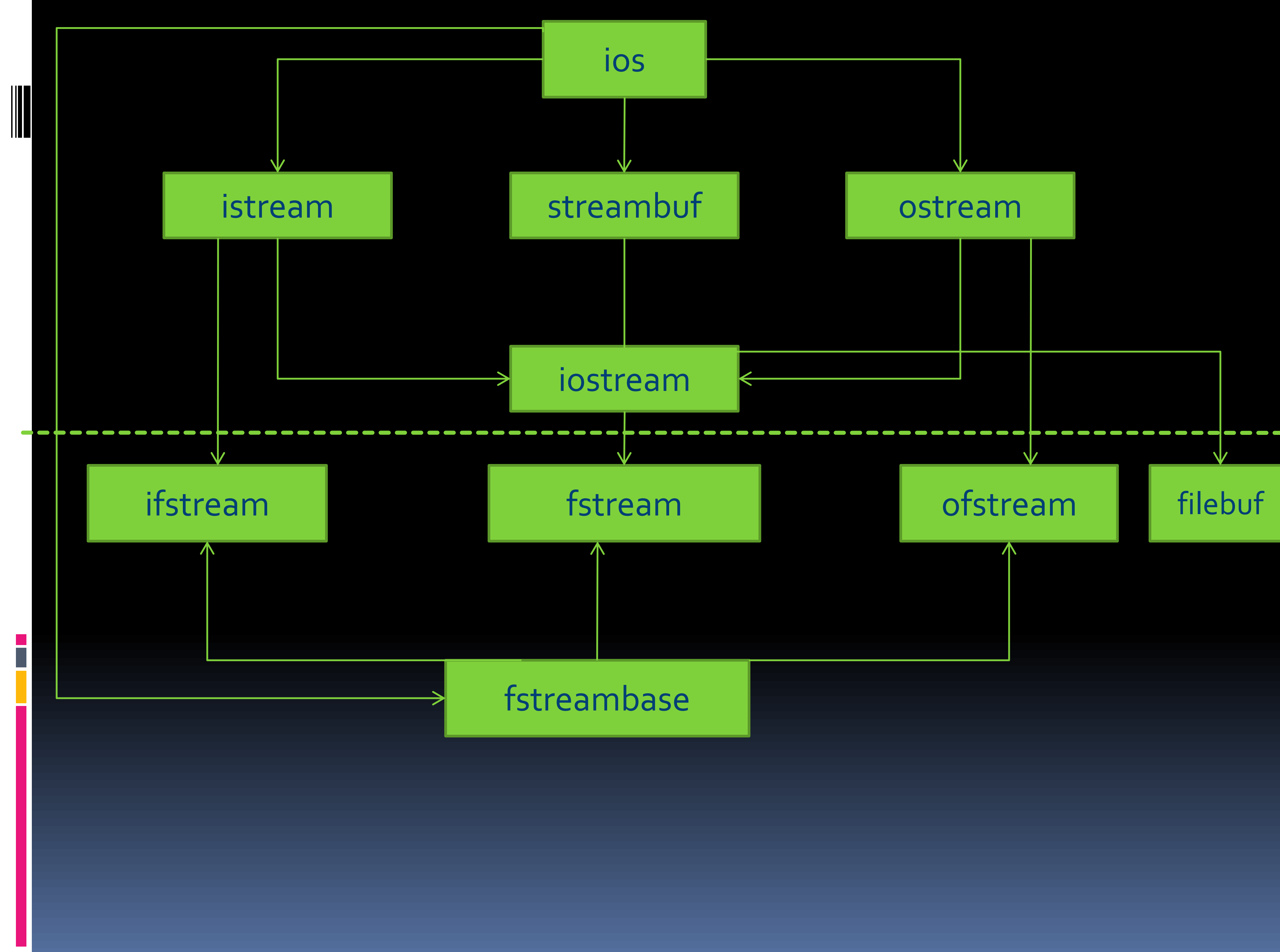
iomanip.h

FILE I/O OPERATION

- Data is stored using the concept of files.
- A file is a collection of related data stored in a particular area on the disk.
- Program contains two type of operations for these files:
  - Data transfer b/w the console unit and the program.
  - Data transfer b/w the programs and a disk file.


External Memory
Data files
Write data to files
Read data from files
Internal Memory
Program + Data
Console Unit
screen
Keyboard
Get data from keyboard
Put data to screen


Input Stream
Read data
Data input
Disk Files
program
Output Stream
Write data
Data output


ios
istream
streambuf
ostream
iostream
ifstream
fstream
ofstream
filebuf
fstreambase

fstreambase

- Base for fstream, ifstream, ofstream.
- Contains open() and close() functions.

ifstream

- Provides input operations.
- Contains open() with default input mode.
- Inherits get(), getline(), read(), seekg() and tellg().

ofstream

- Provide output operations
- Contains open() with default output mode.
- Inherits put(), write(), seekp(), tellp().

fstream

- Provide support for simultaneous input and output operations.
- Inherits all the functions from istream and ostream through iostream.

Opening & Closing a File

- Opening a file using constructor of the class.
  - ofstream is used to create output stream.
  - ifstream is used to create input stream.
  - Initialize the file object with file name.
- Opening a file using open ()
  - It can be used to open multiple files using the same stream object .

Detecting eof

- eof() function can be used. It’s the member function of ios.
- Return non-zero value when end-of-file encountered.

File Modes

- ios::app
  - Append to end-of-file
- ios::ate
  - Go to end-of-file on opening
- ios::binary
  - Binary file
- ios::in
  - Open file for reading only
- ios::out
  - Open file for writing only
- ios::trunc
  - Delete the contents of file if it exists
- ios::nocreate
  - Open fails if the file does not exist
- ios::noreplace
  - Open fails if the file already exist

File pointers and their manipulations

- Each file has two pointers:
  - Input pointer (get pointer)
  - Output pointer (put pointer)

Functions for manipulating file pointers

- seekg()
  - seekg(offset, refposition)
- seekp()
  - seekp(offset,refposition)
- tellg()
- tellp()

ios::beg	ios::cur	ios::end

Sequential I/O Operation

- put() and get() functions
- write() and read() functions:
  - infile.read((char*) & V, sizeof(v));
  - outfile.write((char*) & V, sizeof(v));
- Reading and writing of class object.

Error handling during file operations

- A file which we are attempting to open for reading does not exist.
- The fine name used for a new file may already exist.
- We may attempt an invalid operation such as reading past the end-of-file.
- There may not be any space in the disk for storing more data.
- We may use an invalid file name.
- We may attempt to perform an operation when the file is not opened for that purpose.

Function	Description
eof()	Non-zero if end-of-file encountered.
fail()	True when I/O operation failed.
bad()	True when unrecoverable error occurred.
good()	True if no error has occurred. Means all other functions are false.

EXCEPTION HANDLING

Two common bugs

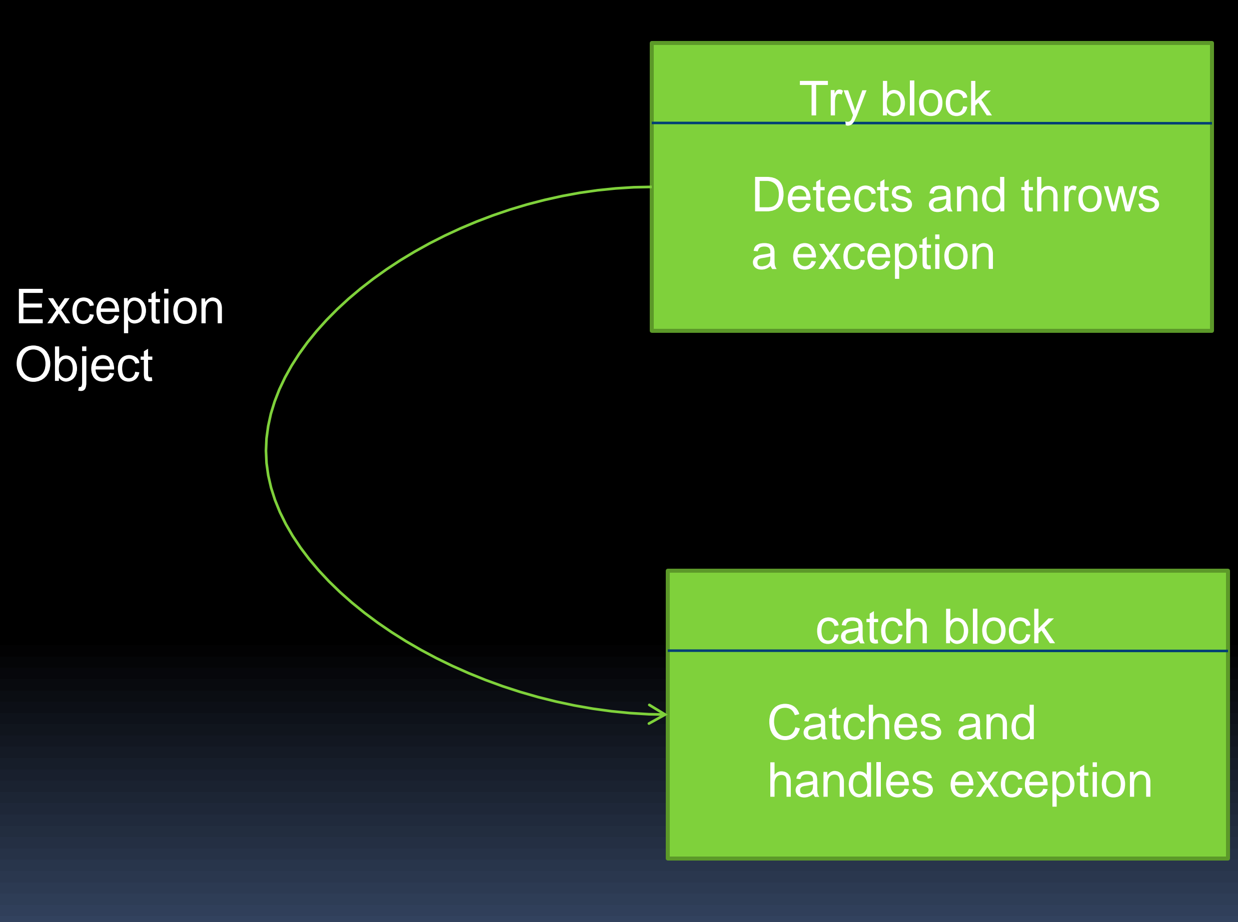
- Logical error – due to poor understanding of solution & problem procedure.
- Syntactical Error – due to poor understanding of language.

Two kinds of exceptions

- Synchronous – Errors under control, like out of range index, overflow.
- Asynchronous – Errors occurred beyond the control of program.

Steps for error handling

- Find the problem(Hit the exception)
- Inform that error has occurred(Throw the exception)
- Receive the error information (Catch the exception)
- Take corrective actions(Handle the exception)


Try block
Detects and throws
a exception
Exception
Object
catch block
Catches and
handles exception