C++ Part 3: File I/O, Multiple Files, Compound Data types

FILE I/O

- C++ has 3 basic classes to handle files,
  - ofstream: Stream class to write on files
  - ifstream: Stream class to read from files
  - fstream: Stream class to both read and write from/to files.
- Defined in <fstream> header file.
- Open file for reading.

```
ifstream file ( “filename” );
```

Or

```
ifstream file;
file.open (“filename”, mode);
```

Mode	Meaning
ios::in	Open for input operations.
ios::out	Open for output operations.
ios::binary	Open in binary mode.
ios::ate	Set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file.
ios::app	All output operations are performed at the end of the file, appending the content to the current content of the file.
ios::trunc	If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.

Class	Default Mode Parameter
Ofstream	Ios::out
ifstream	Ios::in
fstream	Ios::in|ios::out

- Open file for writing.

```
ofstream file("new.txt");
```

Or

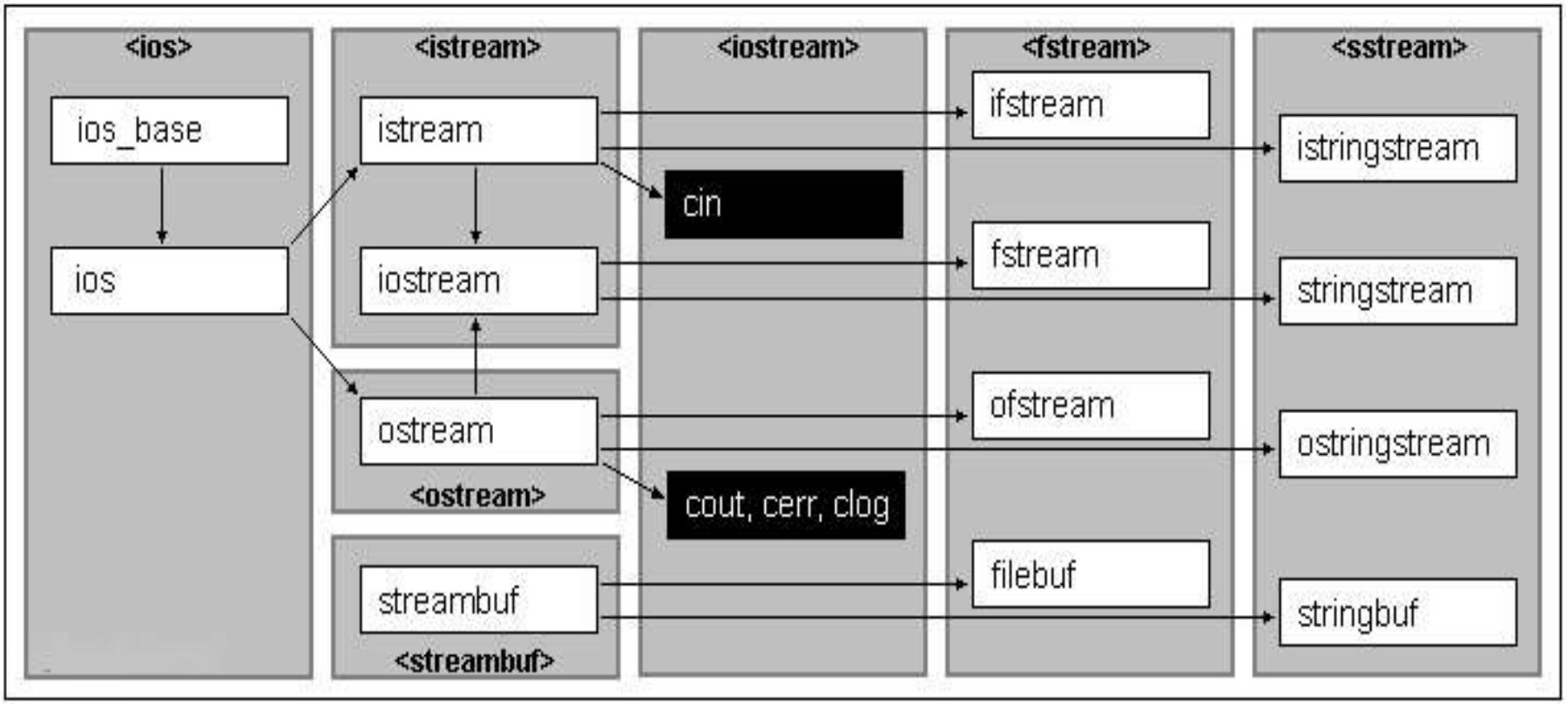
```
ofstream file;
file.open("new.txt");
```

- If file doesn't exist, the file is created. If file exists file is overwritten.
- To close a file after input/output operations

```
file.close();
```

- This method flushes the associated buffers and closes the file.

Input/Output library



- Opening a file for reading/writing using fstream class:

  - Reading:

```
fstream addresses(“Address.fle”, ios::in );
```

  - Writing:

```
fstream addresses(“Address.fle”, ios::out | ios::app);
```

Error Handling

- Check if opening the file was sucessful:

```
if( !myfile)        // or: if( myfile.fail())
```

- Check if end of file has reached:

```
if( myfile.eof())     // At end-of-file?
```

- Check if file is open:

```
if( myfile.is_open() ) { /* . . . */ }          // File is open
```

- To stop a new file from being created when we try to open a file for writing:

```
Ofstream file("file.txt", ios::nocreate);
```

Error Handling

- Other methods used to check specific states of file:
  - Bad(): returns true if reading or writing fails.
  - Fail(): returns true if reading/writing fails or format error occurs.
  - Eof(): returns true if file open for reading reaches the end of the file.
  - Good(): returns false if any of the above are true.
  - Clear(): resets the status flags.

Programming With Multiple Files

```
//Add.cpp
int add(int num1, int num2)
{
        return num1+num2;
}
```

```
//Main.cpp
#include <iostream>
using namespace std;
int add(int, int);
int main()
{
 cout<<" The sum of 3 and 4 is: "<< add(3,4) << endl;
 return 0;
}
```

Header Files

- Header files consist of declarations.
- Consists of 2 parts:
  - Header guard
  - Actual content i.e. declarations.
- Guards
  - #pragmaonce
  - #Ifndef

- #pragmaonce : non-standard, widely supported preprocessor directive

- Standard guard

  #Ifndef <name>: checks if given token has been #define-d earlier, if not statements between *ifndef* and *else* or *endif* is included.

- Syntax:

```
#ifndef _INCL_GUARD
#define _INCL_GUARD
#endif
```

Example

Add.h

```
#ifndef ADD_H
#define ADD_H
int add(int x, int y); //
function prototype, content
of Add.h
#endif
```

Main.cpp

```
#include <iostream>
#include "Add.h"
using namespace std;
int main()
{
 cout<<" The sum of 3 and 4
is: "<< add(3,4) << endl;
 return 0;
}
```

Compound Data types

- Data Structures
- Unions
- Enumerated types (enum)
- Typedef/Using (type aliasing)

Data Structures

- Syntax:

```
Struct typename
{
Type1 member1;
Type2 member2;
.
.
}object_name;
```

- Example:

```
struct product {
int weight;
double price;
} clothes, toys, fruits;
```

- Objects can be declared seperately.

```
Product clothes, toys;
Product fruits;
```

- Syntax to access member:

```
Object.member1;
```

Example Program

```
#include <iostream>
using namespace std;
struct Employee
{
    short id;
    int age;
    double wage;
};
void printInformation(Employee employee)
{
    std::cout << "ID:   " << employee.id << "\n";
    std::cout << "Age:  " << employee.age << "\n";
    std::cout << "Wage: " << employee.wage << "\n";
}
```

Example Program

```
int main()
{
    Employee joe = { 14, 32, 24.15 };
    Employee frank = { 15, 28, 18.27
};

    // Print Joe's information
    printInformation(joe);

    std::cout << "\n";

    // Print Frank's information
    printInformation(frank);

    return 0;
}
```

Expression	Meaning	Equivalent to
a.b	Member b of object a	
A->b	Member b of object pointed to by A	(*a).b
*a.b	Value pointed to by member b of object a	*(a.b)

Example:

```
#include <iostream>
using namespace std;
struct xampl {
  int x;
};
int main()
{
  xampl structure;
  xampl *ptr;
  structure.x = 12;
  ptr = &structure;
  cout<< ptr->x;    // The -> acts somewhat like the * when used with pointers
                    //  It says, get whatever is at that memory address
                    //  Not "get what that memory address is"
}
```

Nested struct

```
struct Employee
{
    short id;
    int age;
    float wage;
};

struct Company
{
    Employee CEO; // Employee is a struct within the Company struct
    int numberOfEmployees;
};

Company myCompany = {{ 1, 42, 60000.0f }, 5 };
```

Struct size

```
struct Employee
{
    short id;
    int age;
    double wage;
};
int main()
{
    std::cout << "The size of Employee is " << sizeof(Employee) << "\n";

    return 0;
}
```

- For performance reasons, the compiler sometimes add gaps into structures called padding.
- To access struct across multiple files add it to a header file.

Unions

- Unions allow one portion of memory to be accessed as different data types
- Syntax:

```
union type_name {
type1 member1;
type2 member2;
type3 member3;
. .
} object_names;
```

Example:

```
union mytypes_t {
char c;
int i;
float f;
} mytypes;

mytypes.c;
mytypes.i;
mytypes.f;
```

Structures with Unions

- structure with regular union

```
struct book1_t {
char title[50];
char author[50];
union {
float dollars;
int yen; } price;
} book1;
```

- Object of the structure with a regular union :

```
book1.price.dollars
book1.price.yen
```

- structure with anonymous union

```
struct book2_t {
char title[50];
char author[50];
union {
float dollars;
int yen; };
} book2;
```

- Object of the Structure with an anonymous union:

```
book2.dollars
book2.yen
```

Typedef/Using

- Type alias: A different name by which a type is identified.
- 2 types:
  - C Style.

  Syntax:

  Typedef existing_type new_name;

  - C++ Style

  Syntax:

  Using new_name= existing_type;

- Typedef example:

```
typedef char C;
typedef unsigned int WORD;
typedef char * pChar;
typedef char field[50];
```

- Using example:

```
using C = char;
using WORD = unsigned int;
using pChar = char *;
using field = char[50];
```

Enumerated types (enum)

- Types defined with a set of custom identifiers, known as enumerators, as possible values.
- Objects of these enumerated types can take any of these enumerators as value.

- There are two kinds of enumerations:
  - unscoped enumeration (declared with the enum-key enum)
  - and scoped enumeration (declared with the enum-key enum class or enum struct).

Syntax

- SYNTAX: Unscoped Enumeration

```
enum type_name {
value1,
value2,
value3,
. .
} object_names;
```

- SYNTAX: Scoped Enumeration

```
Enum class/struct  type_name
{
value1,
value2,
value3,
. .
} object_names;
```

Example

- Example: Unscoped Enumeration

```
enum colors_t { black, blue,
green, cyan, red, purple, yellow,
white };


colors_t mycolor;
mycolor = blue;
if (mycolor == green)
mycolor = red;
```

- Example: Scoped Enumeration

```
enum class Colors { black, blue,
green, cyan, red, purple, yellow,
white };


Colors mycolor;
mycolor = Colors::blue;
if (mycolor == Colors::green)
mycolor = Colors::red;
```

Exercises

- Suppose ted is a double variable. Declare a pointer that points to ted and use the pointer to display ted's value.
- Write a program that reads a file containing a list of numbers and writes two files, one with all the numbers divisible by 3 and another containing all the other numbers.
- Design a file format to store a person's name, address, and other information. Write a program to read this file and produce a set of mailing labels.
- write a Program to Store Information(name, roll and marks) of a Student Using Structure
- write a Program to Store Information of 10 Students Using Structure
- Use enum to define a type called Response with the possible values Yes, No, and Maybe. Yes should be 1, No should be 0, and Maybe should be 2.