Lecture 11.02

Template and Potpourri

SE271 Object-Oriented Programming (2020)

Yeseong Kim

Original slides from Prof. Shin at DGIST

Short Notice

- Will upload HW2 & midterm score with the solution on Wednesday
- The feedback for project proposals is released.

Today's Topic

- After service – Overloading with overriding
- Template – You will love it
- Something useful for your projects
  - Command line argument
  - I/O
  - String stream
  - Smart Pointer

Function Overloading with Overriding

```
class Base {
public:
    Base() {}
    virtual ~Base() {}
public:
    void hello() {}
    void hello(int) {}
 };
```

```
class Child : public Base {
public:
    using Base::hello;
    void hello(int) {}
};

int main() {
    Child c;
    c.hello(); //Error, use using!
}
```

Function Overloading : Swap

```
void Swap(int& a, int& b) {
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
}

void Swap(double& a, double& b) {
    double tmp;
    tmp = a;
    a = b;
    b = tmp;
}
```

```
int main() {
    int a = 10;
    int b = 5;
    Swap(a, b);
    std::cout << “a=“ << a << “ b=“ <<b;
    double c = 1.0;
    double d = 2.0;
    Swap(c, d);
    std::cout << “\nc=“ << c << “ d=“ <<d;
}
```

Templates

- A templates a class or a function that we parameterize with a set of types or values
- We represent general ideas from which we can generate specific classes or functions by providing types (e.g., int, double, or user-defined class) as parameters
- **Syntax**

```
template<typename Type> function_declaration;  // recommended
template<class Type> function_declaration;


template<typename Type> class_declaration;
template<class Type> class_declaration;
```

Example: function templates

```
void Swap(int& a, int& b) {
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
}

void Swap(double& a, double& b) {
    double tmp;
    tmp = a;
    a = b;
    b = tmp;
}
```

```
template<typename T>
void Swap(T & a, T & b) {
    T tmp;
    tmp = a;
    a = b;
    b = tmp;
}
int main() {
    int a = 10, b = 5;
    Swap<int>(a, b);
    std::cout << "a=" << a << " b=" <<b;
    double c = 1.0, d = 2.0;
    Swap<double>(c, d);
    std::cout << "\nc=" << c << " d=" <<d;
}
```

<type> can be omitted

Example: class templates

```
#include <iostream>
template<typename T>
class Point {
    T x;
    T y;
public:
    Point(T xx = 0, T yy = 0) : x(xx), y(yy) { }
    T getX() { return x; }
    T getY() { return y; }
};
int main() {
    Point<int> pt_i{ 1, 2 };
    std::cout << pt_i.getX() << std::endl;
    Point<double> pt_d{ 1.2, 3.4 };
    std::cout << pt_d.getX() << std::endl;
}
```

Example: multiple types

```
template<typename T1, typename T2>
class Student {
    T1 id;
    T2 name;
public:
    Student(T1 id, T2 name) : id(id), name(name) {}
    void Print() {   std::cout << "id: " << id << " name: " << name;    }
};
int main() {
    Student<int, const char*> st1{ 201911000, "Alice" };
    Student<const char*, const char*> st2{"A001", "Carol"};
    st1.Print();
    st2.Print();
}
```

Example: multiple types with a default type

```
template<typename T1, typename T2 = const char*>
class Student {
    T1 id;
    T2 name;
public:
    Student(T1 id, T2 name) : id(id), name(name) {}
    void Print() {   std::cout << "id: " << id << " name: " << name;    }
};
int main() {
    Student<int> st1{ 201911000, "Alice" };
    Student<const char*> st2{"A001", "Carol"};
    st1.Print();
    st2.Print();
}
```

Example: class template with values

```
#include <iostream>
template<typename T, int dim>
class PointND {
    T* coordinates;
public:
    PointND() {
        coordinates = new T[dim];
    }
    ~PointND() {
        delete[] coordinates;
    }
};
int main() {
    PointND<double, 2> pt_2d;
    PointND<double, 3> pt_3d;
}
```

Template Specialization

```cpp
#include <iostream>
#include <cstring>
#include <string>
template<typename T>
    T Add(T n1, T n2) {
    return n1 + n2;
}
template<>
const char* Add<const char*>(const char* s1, const char* s2) {
    std::string str= s1;
    str += " ";
    str += s2;
    char* cstr = new char[str.length() + 1];
    std::memcpy(cstr, str.c_str(),str.length()+1);
    return cstr;
}
```

```cpp
int main() {
    std::cout << Add<int>(10, 20)
<< std::endl;
    std::cout << Add<double>(1.5, 2.5)
<< std::endl;
    std::cout << Add<const char
*>("Hello", "World") << std::endl;
}
```

References

- Learn C++ (https://www.learncpp.com/)
  - Templates: Ch. 13
  - Smart pointers: Ch. 15
  - STL: Ch. 16

- STL
  - http://en.cppreference.com/w/cpp/container

For your project!

Command Line Arguments

```
C:₩text.exe alice 20 bob 21
```

```
#include <iostream>


int main(int argc, char** argv) {
    std::cout << *argv <<"\n";
    std::cout << argv[0] <<"\n";
    std::cout << argc <<"\n";
    for (int i = 1; i < argc; i++)
        std::cout << argv[i] << "\n";
}
```

```
test.exe
test.exe
5
alice
20
bob
21
C:₩text.exe alice 20 bob 21
```

프로젝트 속성 > 구성 속성 > 디버깅 > 명령 인수

[Recap] Example: operator overloading <<

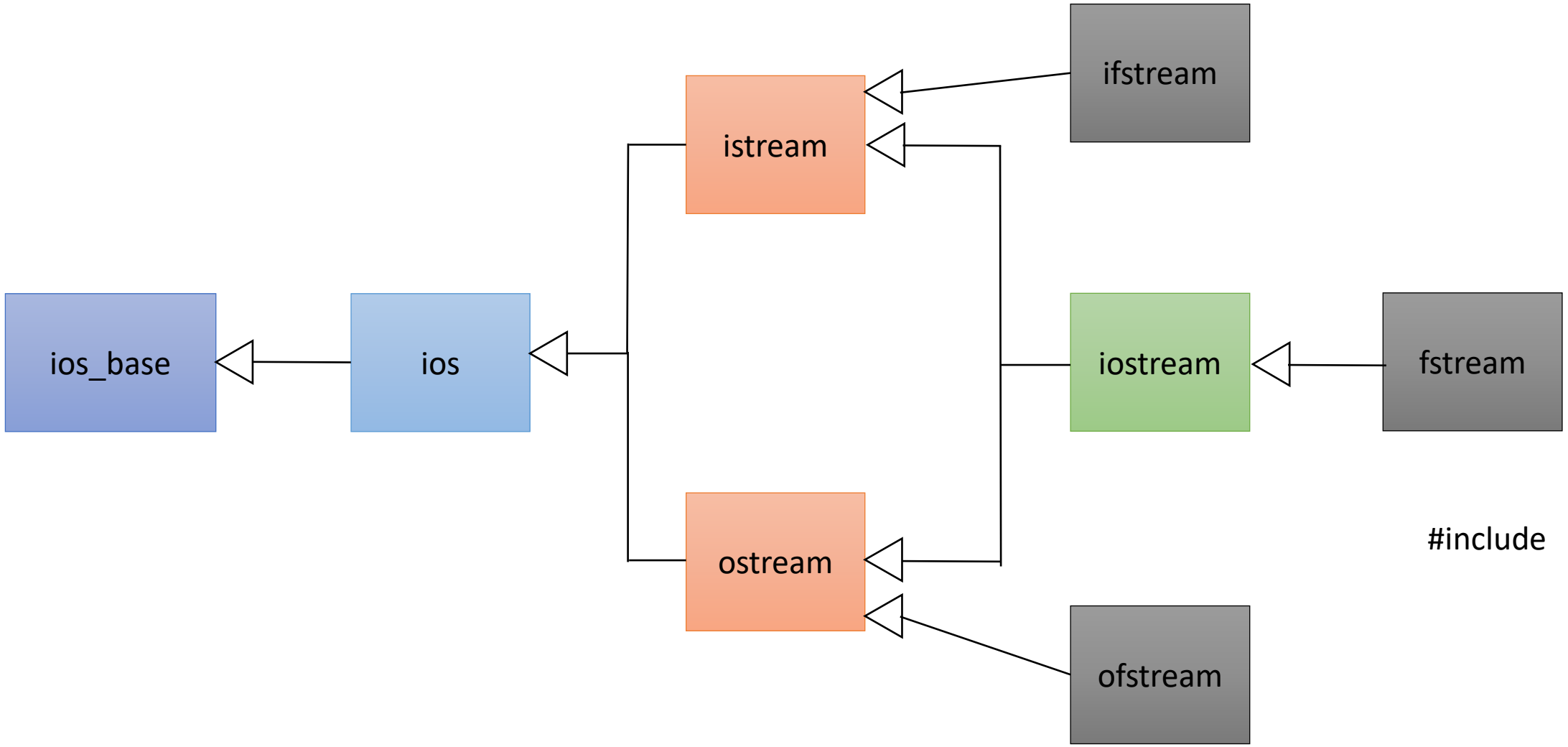
- Complex type (23 + 7j)
  - **cout << Complex(23, 7);**
  - **ostream @ Complex**

```
class Complex {
    int* m_r;  // real part
    int* m_i;  // imaginary part
public:
    ……
    friend ostream& operator<< (ostream& o,
Complex c)
};
```

```
ostream& operator<< (ostream& o, Complex c)
{
    o << *c.m_r << (*c.m_i < 0 ? "" : "+") << *c.m_i
<< "j" ;
    return o;
}

int main(){
    cout << Complex (23, 7);
    return 0;
}
```

Class <fstream>



File Open

- ofstream fo;
- fo.open(“readme.txt”, iso::openmode mode);

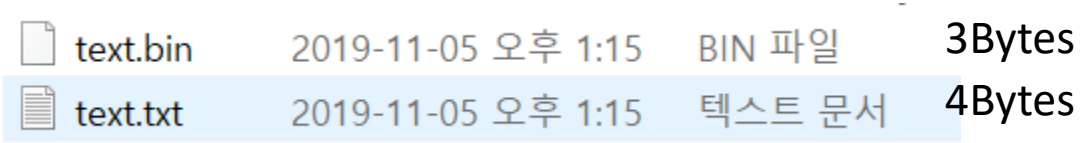
Ios file mode	Meaning
app	Opens the file in append mode
ate	Seeks to the end of the file before reading/writing
binary	Opens the file in binary mode (**instead of text mode**)
in	Opens the file in read mode (default for ifstream)
out	Opens the file in write mode (default for ofstream)
trunc	Erases the file if it already exists

  - Operator |

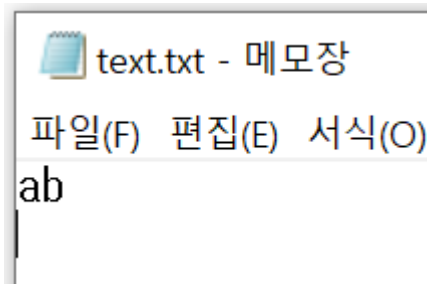
- fo.close();

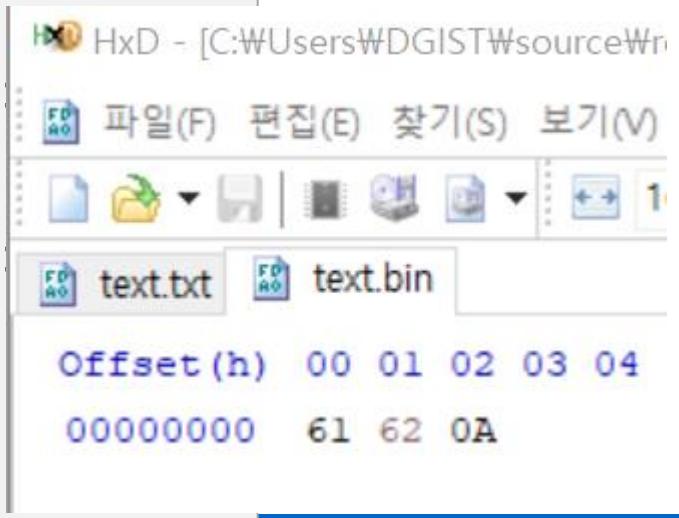
File Type

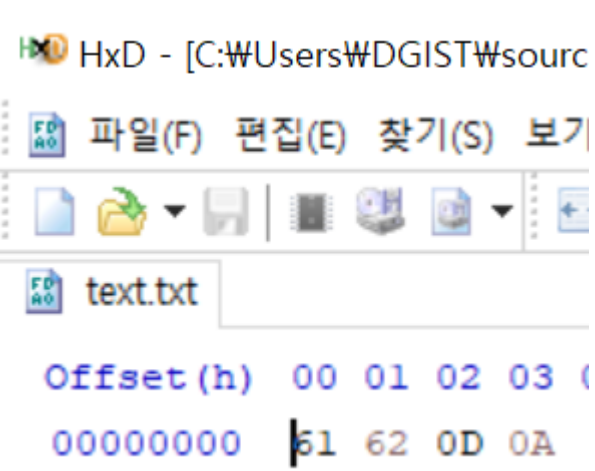
- Two types of files
  - Text file
  - Binary file




```
#include <iostream>
#include <fstream>
int main() {
    char text[] = "ab₩n";
    std::ofstream oft("text.txt", std::ios
    oft.write(text, 3);
    std::ofstream ofb("text.bin", std::ios
std::ios::out);
    ofb.write(text, 3);
}
```







Example: Standard Input Stream

- EOF (end of file)

```
#include <iostream>
#include <fstream>
int main() {
    std::ifstream ift("text.txt", std::ios::in);
    if (!ift) {
        return 0;
    }
    int c;
    while ((c = ift.get()) != EOF) {
        std::cout.put(c);
    }
}
```

```
ab
cd
ef
```

Example: Standard Input Stream

```
#include <iostream>
#include <fstream>
int main() {
    std::ifstream ift("text.txt", std::ios::in);

    while (ift) {
        std::string text;
        ift >> text;
        std::cout << text << std::endl;
    }
}
```

```
ab
cd
ef
```

getline(ifstream& ift, string & line)
getline(char* line, int n)

std::stringstream

```cpp
#include <iostream>
#include <sstream>
#include <string>
#include <iomanip>

int main() {
    std::stringstream os;
    os << std::setw(10) << std::left << "Alice" << '|';
    os << std::setw(10) << std::right << 27 << '|';
    os << std::setw(12) << "053-785-6684" << '|';

    std::string data = os.str();
    std::cout << data;
}
```

```
Alice     |        27|053-785-6684|
```

Smart Pointer

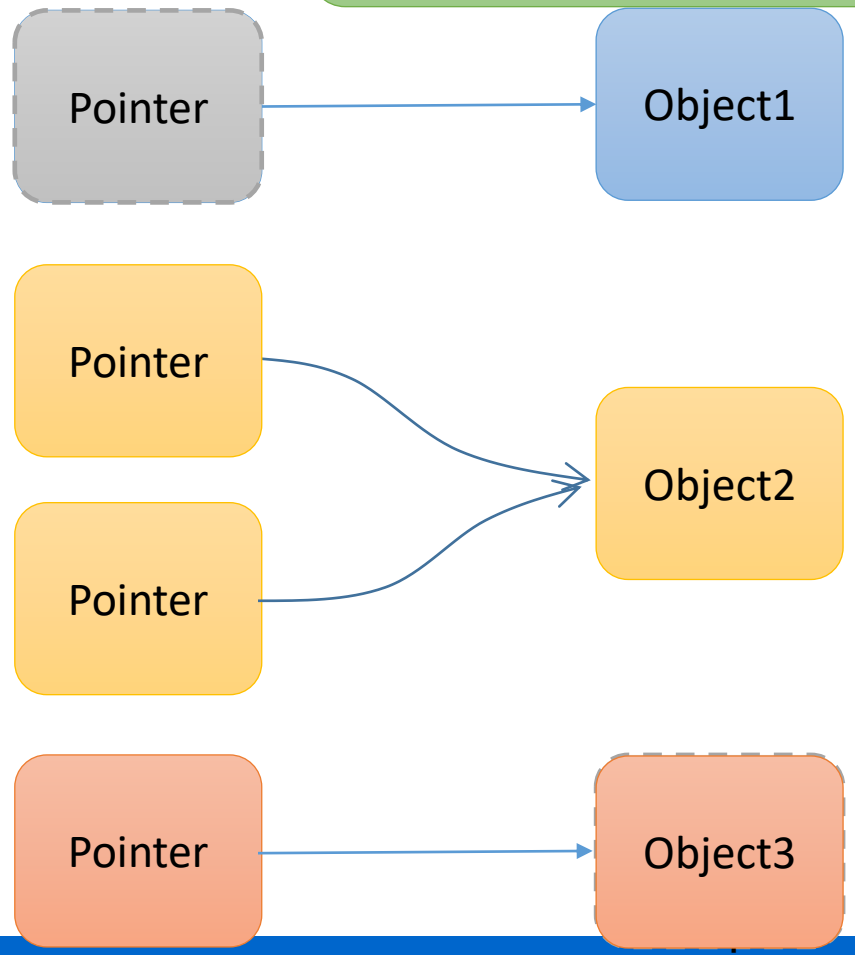
- Problems on using pointers
  - Memory leak
  - Dangling pointer

- Smart pointer
  - auto_ptr (deleted since c++11)
  - std::unique_ptr (c++11)
  - std::shared_ptr (c++11)
  - std::weak_ptr (c++11)

Please Google It!

int * ptr

```
template<typename T>
class Ptr{
int *ptr;
public:
   Ptr():ptr(new int(0)) {}
   ~Ptr() {delete ptr;}
};
```



Example: Smart Pointer

```cpp
#include <iostream>
#include <memory>

int main() {
    std::unique_ptr<int> p1(new int(10));
    std::unique_ptr<int> p2 = p1; // error
    std::unique_ptr<int> p3 = std::move(p1);
    auto& a = *p3;
    std::cout << a << std::endl;

    p3.reset();
    p1.reset();
}
```

```cpp
#include <memory>
int main() {
    std::shared_ptr<int> p1(new int(10));
    std::shared_ptr<int> p2 = p1;

    auto& a = *p2;
    cout <<“value: "<<a<<" owner: "<<p1.use_count();
    p2.reset();
    cout <<"\nvalue: "<<a<<" owner: "<<p1.use_count();
    p1.reset();
    cout <<"\nvalue: "<<a<<" owner: "<<p1.use_count();
}
```

ANY QUESTIONS?

We have backup slides today.
Please take a look at them!

Standard Input Stream

- std::cin
  - Input stream : typically keyboard input
  - Functions
    - get()
      - Read 1 Byte from istream
    - Use binary operator>>
      - istream >> variable
      - variable: int, float, double, char*, char, …
    - getline(char buf[], int size, char delimitChar)
      - Read size-1 characters from input stream until delimiter
      - and write size-1 characters plus ‘\0’ to buf

Example: Standard Input Stream

```
#include <iostream>
int main() {
    char name[20];
    int age;

    std::cout << “Your name?”;
    std::cin >> name;

    std::cout << “Your age?”;
    std::cin >> age;

    std::cout << “Name: “ << name << “\”;
    std::cout << “Age: “ << age;
}
```

```
Your name?  Gildong
Your age?  20
Name: Gildong
Age: 20

Your name?  Gildong
Your age?  20.5
Name: Gildong
Age: 20

Your name?  Gildong Hong
Your age? Name: Gildong
Age: 0
```

Example: file write

```
#include <iostream>
#include <fstream>
int main() {
    char text[] = "ab₩n";
    std::ofstream oft("text.txt", std::ios::out);
    oft.write(text, 3);
    oft.put('c');
    oft.put(100);
    oft << "₩nef";
}
```

```
ab
cd
ef
```

File Output Stream

- ofstream class
  - File Output Stream
    - **Need to specify file where we write**
  - Functions
    - open(filename, mode)
    - put(char)
      - Write 1byte char to output ofstream
    - write(char * buf, int n)
      - Write n-byte from buf to write ofstream
    - Use binary operator<<
      - ofstream << data
      - data: int, float, double, char*, char, …

Example: file write – append mode

```
#include <iostream>
#include <fstream>
int main() {
    std::ofstream oft("text.txt", std::ios::app);
    oft << "end of file";
}
```

<Content of the previous file>

end of file

File Input Stream

- std::ifstream
  - File Input Stream
    - Need to specify file where we read
  - Functions
    - int get()
      - Read 1Byte as integer from input stream
    - Use binary operator>>
      - ifstream >> variable
      - variable: int, float, double, char*, char, …
    - read(char buf[], int size)
      - Read size characters from input stream until eof
    - Int gcout()
      - Number of Bytes recently read

Example: Standard Input Stream

```
#include <iostream>
#include <fstream>
int main() {
    std::ifstream ift("text.bin", std::ios::in |
std::ios::binary);
    while (ift) {
        char data[80];
        ift.read(data, 80);
        for (int i=0; i< ift.gcount(); i++)
            std::cout << data[i];
    }
}
```

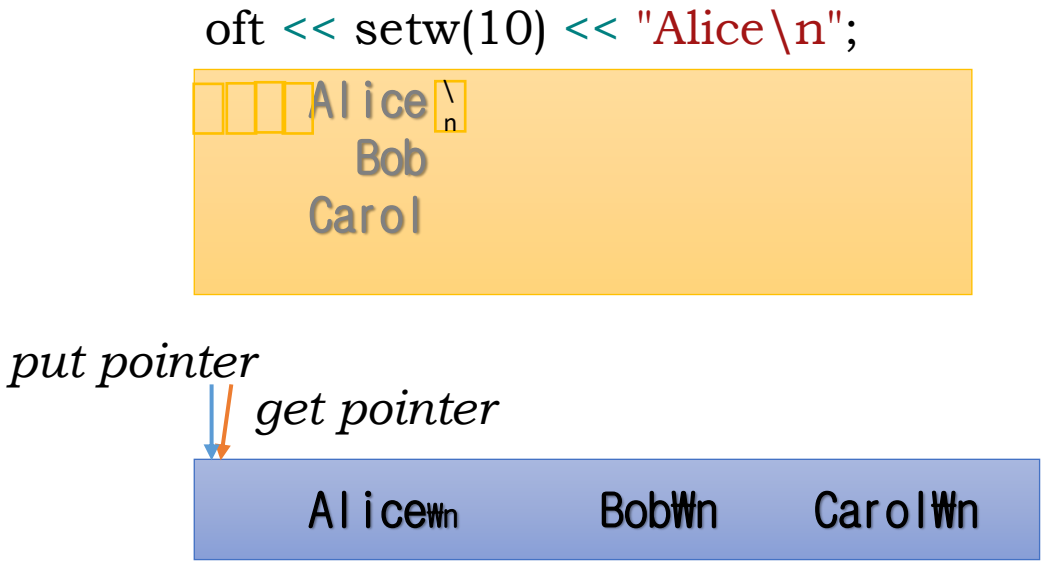
```
3ab
儆儆儆儆儆儆儆儆儆
儆儆儆儆儆儆儆儆儆
儆儆儆儆儆儆儆儆儆
儆儆儆儆儆儆儆儆儆
儆儆儆儆儆儆?r
```

File Pointer Control

- ifstream
  - seekg(streampos pos)
    - Move *get pointer* at pos
  -

Ios seek flag	Meaning
beg	The offset is relative to the beginning of the file (default)
cur	The offset is relative to the current location of the file pointer
end	The offset is relative to the end of the file

- ofstream
  - seekp(streampos pos)
    - Move *put pointer* at pos
  - seekp(streamoff offset, ios:seekdir seekbase)
    - Move *put pointer* at seekbase+offset
  - tellp()
    - Return position of *put pointer*



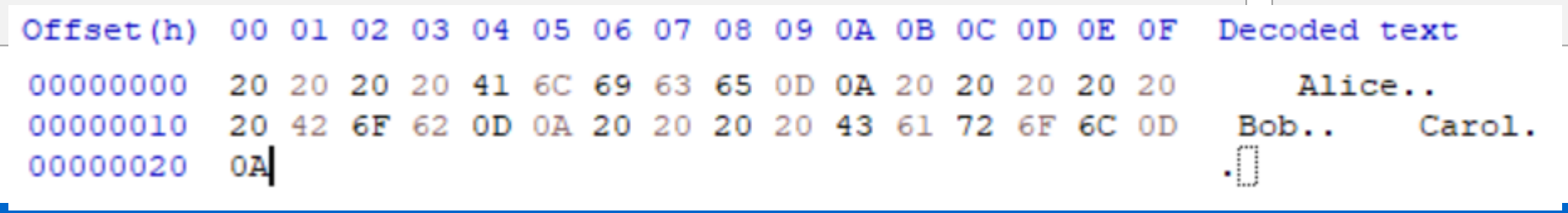
Example: File Pointer Control

```
#include <iostream>
#include <fstream>
int main() {
    std::ifstream ift("text.txt", std::ios::in);
    for (int i = 0; i < 3; i++) {
        std::string text;
        ift.seekg(-11 * (i + 1), std::ios::end);
        getline(ift, text);
        std::cout << text << std::endl;
    }
}
```

```
    Alice
      Bob
    Carol

--------------------

    Carol
      Bob
    Alice
```



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	20	20	20	20	41	6C	69	63	65	0D	0A	20	20	20	20	20	Alice..
00000010	20	42	6F	62	0D	0A	20	20	20	20	43	61	72	6F	6C	0D	Bob.. Carol.
00000020	0A																.

Stream State Flag

Flag	Meaning
goodbit	Everything is okay
badbit	Some kind of fatal error occurred (e.g. the program tried to read past end of a file)
eofbit	The stream has reached the end of a file
failbit	A non-fatal error occurred (eg. the user entered letters when the program was expecting an integer)

```
Bad | fail | eof | good
```

```
cout << ios::goodbit
```

Member function	Meaning
good()	Returns true if the goodbit is set (the stream is ok)
bad()	Returns true if the badbit is set (a fatal error occurred)
eof()	Returns true if the eofbit is set (the stream is at the end of a file)
fail()	Returns true if the failbit is set (a non-fatal error occurred)
clear()	Clears all flags and restores the stream to the goodbit state

Example: Stream State Flag

```cpp
#include <iostream>

int main() {
    std::cout << "Enter your age: \n";
    int nAge;
    std::cin >> nAge;
    std::cout << std::cin.good();
    std::cout << std::cin.bad();
    std::cout << std::cin.fail();
}
```

```
Enter your age:
Alice
001
--------------------
Enter your age:
20
100
```