Computer Programming

Stream Processing

Hung-Yun Hsieh

December 13, 2022

Computer Programming

Stream I/O

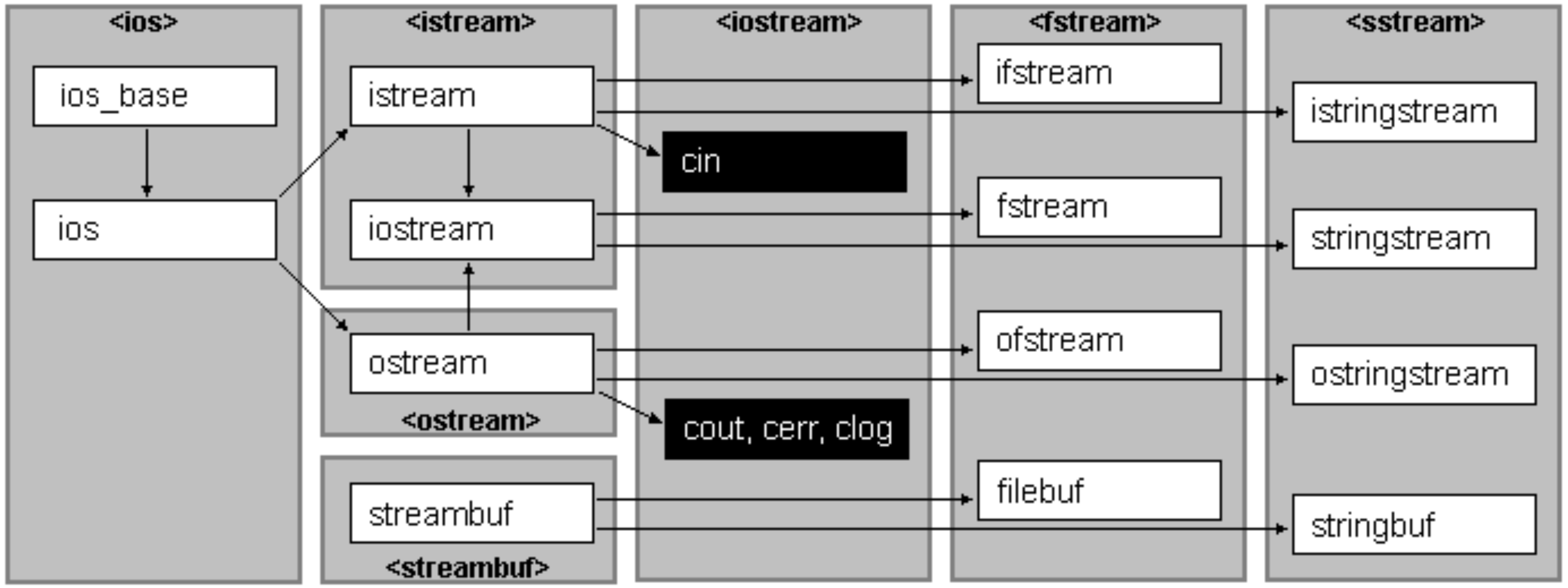
Stream

- C++ I/O occurs in streams
  - ☞ Stream is a sequence of bytes
  - For input, bytes flow from a device (e.g. keyboard or disk drive) to the main memory
  - For output, bytes flow from the main memory to a device (e.g. display or disk drive)
- Formatted vs. unformatted I/O
  - Formatted (high-level) I/O: group of bytes as a logic unit (e.g. integer and floating-point number) is preserved
  - Unformatted (low-level) I/O: individual byte is the item of interest
  - ☞ Both types of I/O are supported in C++ standard I/O streams

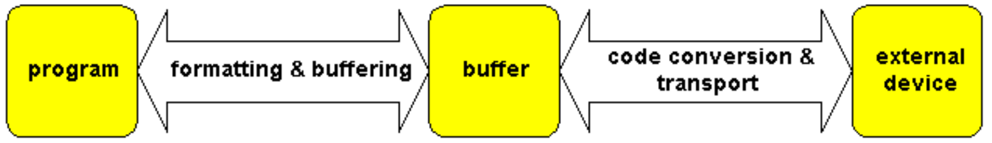
Class Hierarchy

The entire class hierarchy resides in the `std` namespace

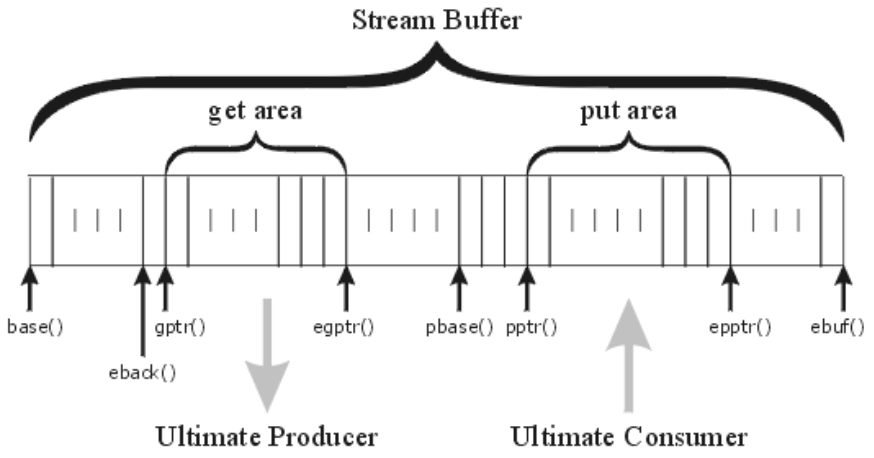
- The `ios` base class
  - Includes common character-dependent functionality and state variables required by all streams
    - Member functions
    - Stream state, formatting flags, and stream buffer



Stream Buffer



- Stream buffer
  - Acts as a buffer between the source (producer) / target (consumer) of data (e.g. input / output device) and the member functions of the classes derived from `ios` that format this raw data
  - Implemented as an array of bytes
    - get area (for input stream): the space available to accept bytes from the ultimate producer
    - put area (for output stream): the space available to store bytes that are on their way to the ultimate consumer



Stream Output

The predefined object `cout` is an `ostream` instance and is "connected to" the standard output device, which usually is the display screen (monitor)

- Class `ostream`
  - Stream insertion operator: `operator <<()`
  - Member function `put()` to output a character

```
ostream& put(char c);
```

```
#include <iostream>
using namespace std;

int main( )
{
    char *word = "again";

    cout << "Value of word is: " << word << endl
         << "address of word is: " << static_cast<void *>(word) << endl;

    cout.put('A');
    cout.put('B').put('\n');
    cout.put(67);
}
```

Can be used to show the pointer value (memory address) of the C-style character string

Stream Input

The predefined object `cin` is an `istream` instance and is "connected to" the standard input device, which usually is the keyboard

- Class `istream`
  - Stream extraction operator: `operator >>()`
  - Member function `get()` to input a character

```
int get();
istream& get(char& c);
```

```
#include <iostream>
using namespace std;

int main()
{
    char next;
    int blank_count = 0;
    do {
        cin.get(next);
        if (next == ' ') blank_count++;
        else cout.put(next) ;
    } while (next != '\n');

    cout << "Number of blanks = " << blank_count << endl;
}
```

Comparing `get` and `getline`

The delimiter character is not placed in the buffer for both `cin.get()` and `cin.getline()`

The delimiter character remains in the input stream for `cin.get()`, but it is extracted from the input stream in `cin.getline()`

```
#include <iostream>
#include <iomanip>
using namespace std;

int main( )
{
    const int SIZE = 80;
    char buffer1[SIZE], buffer2[SIZE], buffer3[SIZE];

    cout << "Enter a sentence for cin: " << endl;
    cin >> setw(SIZE) >> buffer1;

    cout << "Enter a sentence for cin.get: " << endl;
    cin.get(buffer2, SIZE, '\n');

    cout << "Enter a sentence for cin.getline: " << endl;
    cin.getline(buffer3, SIZE, '\n');
    cout << buffer1 << endl << buffer2 << endl << buffer3 << endl;
}
```

End-of-file (EOF) is `<Ctrl>-Z` on Windows, `<Ctrl>-D` on UNIX/Mac

More on Stream Input

- `cin::ignore()`
  - Read and discard a designated number of characters (default is one character) or terminate upon encountering a designated delimiter (default is EOF)

```
cin.ignore(1000, '\n');
```

The delimiter is also extracted from `cin`

- `cin::putback()`
  - Place the previous character obtained by a get from an input stream back into that stream (next to get)
- `cin::peek()`
  - Return the next character from an input stream but does not remove the character from the stream

Example on `ignore()`

A fail bit is set on `cin` after the third `cin.get()` since no character is extracted from the stream

The `'\n'` character is not extracted from `cin` in `cin.get()`

```
#include <iostream>
using namespace std;

int main( )
{
    const int SIZE = 80;
    char buffer1[SIZE], buffer2[SIZE], buffer3[SIZE];

    cout << "Enter a sentence for the first cin.get: " << endl;
    cin.get(buffer1, SIZE, '\n');

    cout << "Enter a sentence for the second cin.get: " << endl;
    cin.ignore(1000,'\n');
    cin.get(buffer2, SIZE, '\n');

    cout << "Enter a sentence for the third cin.get: " << endl;
    cin.get(buffer3, SIZE, '\n');

    cout << buffer1 << endl << buffer2 << endl << buffer3 << endl;
}
```

Example on `putback()`

```
#include <iostream>
using namespace std;

int main () {
  char c;
  int n;
  char str[256];
  cout << "Enter a number or a word: ";
  c = cin.get();
  if ( (c >= '0') && (c <= '9') ) {
    cin.putback(c);
    cin >> n;
    cout << "You have entered number " << n << endl;
  }
  else {
    cin.putback(c);
    cin >> str;
    cout << "You have entered word " << str << endl;
  }
}
```

`cin.unget()`

`cin.unget()`

How to rewrite the program using `cin.peek()`?

Unformatted I/O

- Processing raw bytes
  - Bytes are not formatted based on the data type
  - Member functions `read()` and `write()`

```
#include <iostream>
using namespace std;

int main( )
{
    const int SIZE = 80;
    char buffer[SIZE];

    cout << "Enter a sentence: " << endl;
    cin.read(buffer, 20);

    cout << endl << "The sentence entered was: " << endl;
    cout.write(buffer, cin.gcount());
}
```

If fewer than the designated number of characters are read in `cin.read()`, a fail bit is set

`cin.gcount()` returns the number of characters read by the last input operation

Stream Error States (1/2)

- Finding the state of a stream

```
#include <iostream>
using namespace std;

void show_state()
{
    cout << "\ncin.rdstate(): " << cin.rdstate()
         << "\n    cin.eof(): " << cin.eof()
         << "\n   cin.fail(): " << cin.fail()
         << "\n    cin.bad(): " << cin.bad()
         << "\n   cin.good(): " << cin.good() << endl;
}

int main( )
{
    int a;
    cout << "\nBefore a bad input operation:";
    show_state();
```

Stream Error States (2/2)

```
    cin >> a; // now enter a character to cause error
    cout << "\nAfter a bad input operation:";
    show_state();

    cin.clear();
    cout << "\nAfter cin.clear():";
    show_state();
}
```

- Stream state
  - Different states can be used to indicate different errors
  - If an error occurs during an I/O operation and the stream is set to anything other than the "good" state, further operations on that stream will be ignored
  - ☞ Use `clear()` to reset the stream to the "good" state

More on Stream Error States (1/2)

- Status bits
  - `ios::badbit` (1L << 0 → *that is, 1*)
    - Indicates a loss of integrity in an input or output sequence (such as disk full or an irrecoverable read error from a file)
  - `ios::eofbit` (1L << 1 → *that is, 2*)
    - Indicates that an input operation reached the end of an input sequence (end-of-file)
  - `ios::failbit` (1L << 2 → *that is, 4*)
    - Indicates that an input operation failed to read the expected characters (e.g. format), or that an output operation failed to generate the desired characters
  - `ios::goodbit` (0)
    - None of the above three

More on Stream Error States (2/2)

- Testing the stream states
  - Member function `rdstate()` returns the state of the stream
  - A state can have multiple bits set – use the bitwise operation to test whether a given bit is set

```
if (cin.rdstate() & ios::failbit)
{
   cerr << "The failbit is set.\n";
}
```

  - Member function `bad()`, `eof()`, `fail()`, and `good()` provide a handy way to test the stream state

```
if (cin.fail())
{
   cerr << "The failbit is set.\n";
}
```

Example

- Validation of numerical inputs

```
#include <iostream>
using namespace std;
```

```
int main()
{
    int nAge;
    while (1) {
        cout << "Enter your age: ";
        cin >> nAge;
        if (cin.fail()) {
            cin.clear();
            cin.ignore(1000, '\n'); // clear out the bad input
            continue;
        }
        if (nAge <= 0) continue;
        break;
    }

    cout << "You entered: " << nAge << endl;
}
```

reset the state bit back to `goodbit` so we can use `ignore()` later on the stream

Another Way to Test Stream Errors

- Testing of the stream object
  - The `operator!` member function returns true if the `badbit` and/or `failbit` is set
  - The `operator void *` member function returns a null pointer if the `badbit` and/or `failbit` is set
    - ☞ A common use of converting some class type to `void *` is to allow instances of the class to be tested
    - ☞ When a pointer value is used as a condition, c++ converts a null pointer to `false` and non-null pointer to `true`

```
int main()
{
    int a;
    cout << "Please enter integers:\n";
    while (cin >> a) cout << "You have entered: " << a << endl;
    cout << "Stream states: " << cin.rdstate() << endl;
}
```

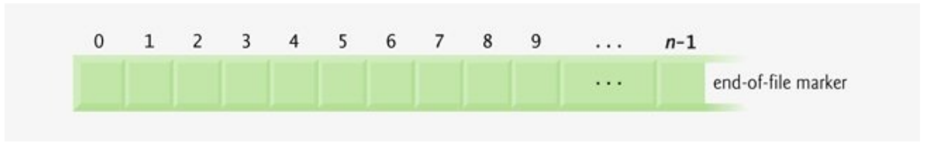
```
 if (!cin) …
  if (cin) …
```

Computer Programming

File Processing

File and Stream

- C++ views each file as a sequence of bytes
  - When a file is opened, an object is created, and a stream is associated with the object
  - The user only needs to deal with the object for file I/O
- Classes `ofstream` and `ifstream`
  - Defined in <fstream>
  - Derived from `ostream` and `istream`
  - ☞ Class `fstream` inherits from `iostream`
  - All member functions, operators and manipulators that belong to standard I/O streams can also be applied to file streams



Output File

Use `open("filename", ios::app)` to open the file in the "append" mode, where new data is written at the end of the file and old data is not overwritten

```
void open(filename, openmode);
```

- File for writing
  - Include the <fstream> header

```
ofstream object_name("filename");
```

The default open mode is `ios::out`

  - Alternatively

```
ofstream object_name;
object_name.open("filename");
```

If the file does not exist, a new file will be created; otherwise, all data in the file is overwritten

  - Write data to a file like writing to `cout`

```
object_name << "This is the message" << endl;
```

☞ Can use `setw()`, `setprecision()`, …

  - File close

```
object_name.close();
```

Once closed, `object_name` may be used to open another file

Output to a File

If the `outfile` stream cannot open the file, the `failbit` is set, and `operator!()` returns false

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

int main( )
{
    double income=123.45, expense=987.65;
    int week=7, year=2005;

    ofstream outfile("prog.txt");
    if (!outfile) {
        cerr << "File could not be open" << endl; exit(1);
    }
    outfile << "Week=" << week << endl << "Year=" << year << endl;
    outfile << "Income=" << income << endl
            << "Expenses=" << expense << endl;

    outfile.close();
}
```

Member function `is_open()` can be used to determine whether the object is associated with a file

Use "C:\\prog.txt" for a file with absolute path

Create a file with filename "prog.txt" in the current directory for writing

Similar to the use of cout

Appropriately close and disassociate the file

Input File

If `object_name` already has a file associated (open) with it, `open()` fails, and the `failbit` is set

- File for reading
  - Include the <fstream> header

    ```
    ifstream object_name("filename");
    ```

  - Alternatively

    ```
    ifstream object_name;
    object_name.open("filename");
    ```

  - Read data from a file like reading from `cin`

    ```
    object_name >> variable_name;
    ```

    ☞ Can use `get`(), `getline`(), …

  - File close

    ```
    object_name.close();
    ```

Member function `is_open()` returns if the stream is currently associated with a file

The default open mode is `ios::in`

Once closed, `object_name` may be used to open another file

Input from a File

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

int main( )
{
    double x;
    int i, j;
    ifstream infile("dat.txt");
    if (!infile) {
        cerr << "Error opening input file" << endl; exit(1);
    }
    infile >> i >> j >> x;
    infile.close();
    cout << "From file i=" << i << ", j=" << j << ", x=" << x << endl;
}
```

How to handle a file with unknown amount of data?

Reads three values from the file

```
dat.txt
36
123 456.78
```

```
dat.txt
12 18.3 89.01
```

Data File

```
#include <iostream>
#include <fstream>
using namespace std;

int main( )
{
    int i=0, row;
    double x[50], sum;
    ifstream infile("expr.dat");

    while (!infile.eof()) {
        infile>>x[i];
        i++;
    }
    row = i;
    for (i=0, sum=0;i<row;i++) sum += x[i];

    cout << "Total number of data points=" << row << endl;
    cout << "Their sum is=" << sum << endl;
}
```

How to handle a file with multiple columns of data?

Use dynamic memory management or `seekg()` to avoid the use of magic number "50" in the program

Using `while (infile)`… is more reliable since the `eofbit` may not be set if there is trailing whitespace after the digits

Data File Take Two (1/2)

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

int main( )
{
    double **a=NULL, max=0;
    int row, col, i, j;

    ifstream infile("expr.dat");

    if (!infile) {
        cerr << "Error opening input file" << endl;
        exit(1);
    }
    infile >> row >> col;

    a = new double*[row];
    for (i=0;i<row;i++) a[i] = new double[col];
```

expr.dat

```
3 4
12.3 33.1 59.2 -41.3
10.3 7.3  -3.9 112.3
5.8  -9.3 -33.1 15.6
```

Do proper error checking on the values of `row` & `col`

Do proper error checking on `a` and `a[]` (null pointer)

Data File Take Two (2/2)

```
    for (i=0;i<row;i++) {
        for (j=0;j<col;j++) {
            infile>>a[i][j];

            if (a[i][j] > max) max = a[i][j];
        }
    }

    cout << "The max value is=" << max << endl;

    for (i=0;i<row;i++) delete [] a[i];
    delete [] a;

    infile.close();

    return 0;
}
```

File Position Pointer

- Sequential file
  - The get / put pointer is updated as data is read / written from the file stream
  - The file may be read for several passes
    - To first determine the number of records in the file
  - It may be desired that a file is not processed sequentially from the first byte
    - Only a particular entry or record in the file needs to be updated
- File position pointer
  - Points to the next byte in the file to read or write
    - The get pointer in the `istream`
    - The put pointer in the `ostream`
  - Member functions `tellg()` and `tellp()`

Moving the File Position Pointer

```
istream& seekg(offset, direction);
ostream& seekp(offset, direction);
```

- Moving the pointer
  - Member function `seekg()` in `istream`
    - Move the get-pointer for input ("seek get")
  - Member function `seekp()` in `ostream`
    - Move the put-pointer for output ("seek put")
  - Specifying the direction of movement
    - `ios::beg`
    - `ios::end`
    - `ios::cur`

`seekg(n, ios::cur)` positions `n` bytes forward in the `istream` object

☞ Use `seekg(0)` or `seekp(0)` to reposition the pointer to the beginning (location 0) of the file

☞ Use `seekg(0,ios::end)` or `seekp(0,ios::end)` to reposition the pointer to the end of the file

Example

```
#include <iostream>
#include <fstream>
using namespace std;

int main ()
{
    long begin, end;
    ifstream myfile("expr.txt");

    begin = myfile.tellg();

    myfile.seekg(0, ios::end);
    end = myfile.tellg();

    myfile.close();
    cout << "File size is: " << (end-begin) << " bytes.\n";
}
```

Binary File

The difference between the binary and text (default) mode is that there is no implicit, system-specific conversion on the file (e.g. '\n' to '\r''\n')

- Data is stored in raw bytes (unformatted)
  - Use the `read()` and `write()` member functions
- File open mode revisited

	Mode	Description
1L<<3	ios::in	Open a file for input
1L<<4	ios::out	Open a file for output
1L<<2	ios::binary	Open a file in the binary mode (vs. text mode)
1L<<5	ios::trunc	Discard the file contents if they exist (default for ofstream)
1L<<1	ios::ate	Move to the end of the file upon opening (can move later)
1L<<0	ios::app	Seek to end before each write (implies ios::out)

Handling Raw Bytes

If the data is written to the file using
`file << num;`
then the size of the file is 5 bytes

The `reinterpret_cast` operator is used for cases where a pointer of one type must be cast to an unrelated pointer type

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

int main () {
    fstream file("example.bin", ios::out|ios::binary);
    if (!file.is_open()) { cout << "Unable to open file"; exit(1);}
    int num = 10000;
    file.write(reinterpret_cast<char *>(&num), sizeof(num));
    file.close();

    file.open("example.bin", ios::in|ios::binary);
    if (!file.is_open()) { cout << "Unable to open file"; exit(1);}

    int data;
    file.read(reinterpret_cast<char *>(&data), sizeof(data));
    file.close();
    cout << "Data is " << data << endl;
}
```

Example

```
#include <iostream>
#include <fstream>
using namespace std;

int main () {
    ifstream file ("example.bin", ios::in|ios::binary|ios::ate);
    if (file.is_open())
    {
      int size = file.tellg();
      char *memblock = new char [size];
      file.seekg(0, ios::beg);
      file.read(memblock, size);
      file.close();
      cout << "the complete file content is in memory";
      // do processing now
      delete[] memblock;
    }
    else cout << "Unable to open file" << endl;
}
```

Computer Programming

String

String as a Class

```
#include <iostream>
#include <string>
using namespace std;

int main( )
{
    string s1, s2, s3;

    s1 = "We can ";
    s2 = "use = + < and other operators with string objects";

    s3 = s1 + s2;
    s3 += '.';

    if (s1 < s2) cout << s3 << endl;
}
```

No need to specify the size
(cf. array→ size cannot be changed later)

String comparison – compare in terms of the ASCII code of the first character of individual strings

```
W=87
u=117
```

String

- The `string` class
  - C++ automatically keeps track of the size of the string and reallocates the space if needed

```
char a[] = "This is";
string b = "This is";

b = "This is a longer string."
strcpy(a, "This is a longer string.");
```

Okay!

Dangerous and wrong!

  - C++ standard library implements several member functions for the string class for manipulation of strings
  - ☞ Operator overloading in particular
    - = (initialization & assignment) (cf. character array)
    - + (concatenation)
    - >, <, == (comparison)
    - [] (subscript)

String Member Functions

Other functions:
`s1.at(i)`
`s1.substr(i, 4)`
`s1.c_str()`

```
#include <iostream>
#include <string>
using namespace std;

int main( )
{
    string s1("String of many words."), s2 = "many";
    int i;

    i = s1.find(s2);
    s1.replace(i, 4, "few");
    cout << s1 << endl;

    s1.erase(i, 4);
    cout << s1 << endl;

    s1.insert(10, "simple ");
    cout << s1 << endl;
    cout << "The length of s1 is " << s1.length() << endl;
}
```

Most of these member functions are overloaded with different number of arguments

Position in string is numbered from 0

# of characters to replace

Input to String

```
istream& getline(istream&, string&);
istream& getline(istream&, string&, char);
```

```
#include <iostream>
#include <string>
using namespace std;

int main( )
{
    string s1, s2;

    cout << "Enter a single word" << endl;
    cin >> s1;
    cin.ignore(1000, '\n');
    cout << s1 << endl << endl;

    cout << "Enter a few lines. Terminate with #" << endl;
    getline(cin, s2, '#');
    cout << s2 << endl << endl;
}
```

Good for reading a single word separated by "whitespace"

One can use
`getline(cin, s2);`
to read a line from `cin`

String and File (1/2)

```
This is a test file
Testing for the NTU C++ class
Replace all NTU words by the full NTU name
Output file does not have any NTU word
```

input file:
in.txt

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main( )
{
    string s1;
    int i,j, k=0;
    ifstream infile("in.txt");
    ofstream outfile("out.txt");
```

String and File (2/2)

```
    while (getline(infile, s1))
    {
        j=0;
        while ((i=s1.find("NTU",j))>=0)
        {
            s1.erase(i, 3);
            s1.insert(i, "National Taiwan University");
           k++;
            j = i;
        }
        outfile << s1 << endl;
    }
    cout << "Number of replacements:" << k << endl;

    infile.close();
    outfile.close();
}
```

Read line by line until error occurs

Find for all "NTU" words

`While ((i=s1.find(…))!= string::npos) …`

Write the modified string to the output file

Computer Programming

String Stream

String Stream

- C++ stream I/O
  - Standard stream I/O
  - File stream I/O
  - String stream I/O
    - ☞ Input from string or output to string (in-memory I/O)
- String stream processing
  - ☞ Input from a string: get arbitrary input then do validation
  - ☞ Output to a string: format the output nicely
  - Include the <sstream> header

```
ostringstream oss_name;
istringstream iss_name;
```

- ☞ Use the member function `str()` to get the string

Output String Stream

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main( )
{
    string s1("The first number is "), s2("The second number is ");
    int n1=39, n2=18;

    ostringstream sout;

    sout << s1 << n1 << ". " << s2 << hex << n2 << ".";

    cout << "Output string: " << sout.str() << endl;
    sout << " (The second number is in hex!)";
    cout << "Output string: " << sout.str() << endl;

}
```

Create an `ostringstream` object `sout` for string processing

Write as output (insert) to the stream object `sout`

Get the string stored in `sout`

Input String Stream

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main( )
{
    string s1("Input test 123 5.7 A"), s2, s3;
    int i;
    double x;
    char c;
    istringstream sin(s1);

    sin >> s2 >> s3 >> i >> x >> c;
    cout << "The following items are extracted:"
         << "\nstring: " << s2 << ", " << s3
         << "\ninteger: " << i
         << "\ndouble: " << x
         << "\ncharacter: " << c << endl;
}
```

Create an `istringstream` object `sin` from string `s1`

Read as input (extract) from the stream object `sin`

Number/String Conversion

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main( )
{
    string s;
    double x=123.59, y;
    stringstream sio;

    sio << x;
    sio >> s;

    sio.clear();
    sio << s;
    sio >> y;

    cout << "x=" << x << ", y=" << y << endl;
}
```

One can use `sio.str("")` to clear the content of the stream

A stream object both for input and output

Why `clear()` here?

To clear the `eofbit` that was set after the last stream extraction statement

Review

- C++ stream
  - The C++ stream class hierarchy
  - Standard I/O stream, file stream, and string stream
  - Output stream, stream insertion operator, and the member functions to put to the stream
  - Input stream, stream extraction operator, and the member functions to get from the stream
  - Stream error states
- C++ string
  - Encapsulation of the character array into a string class for a more reliable and flexible manipulation of string