C++ Streams

1. I/O Stream
2. File Stream
3. String Stream

Compare cout and printf

- cout is convenient. You don’t need to worry about those conversion specifiers:
  - %s for string
  - %d for decimal integer
  - %x for hexadecimal
  - %f for floating point
- For your defined data type (e.g., CRational), cout knows how to print it out, as long as you have defined `operator<<()` for CRational.
  - For each printf(), you must work out the details to print out its numerator and denominator.

Q: printf is good at formatting

- Sometimes I still want to specify the format:
  - %4d
    - width of an integer
    - padding '0' if the integer is shorter
  - %5.2f
    - number of digits after the decimal number
  - %-10s
    - left-justify (default is right-justify, including strings)

Right-Justify with Padding

```
#include <stdio.h>

int main() {
    int a[] = {1, 10, 100};
    for (int i=0; i<3; ++i)
        printf("%5d\n", a[i]);

    for (int i=0; i<3; ++i)
        printf("%05d\n", a[i]);
    return 0;
}
```

```
    1
   10
  100
00001
00010
00100
```

Right-Justify with Padding in C++

```
#include <iostream>
#include <iomanip>
using std::setw;
using std::setfill;
using std::cout;
using std::endl;

int main()
{
    int a[] = {1, 10, 100};
    for (int i=0; i<3; ++i)
        cout << setw(5) << a[i] << endl;

    for (int i=0; i<3; ++i)
        cout << setw(5) << setfill('0') << a[i] << endl;
    return 0;
}
```

```
    1
   10
  100
00001
00010
00100
```

Floating Point Number

```
#include <stdio.h>

int main() {
    float b[] = {3.14159, 27.625, 100.75};
    for (int i=0; i<3; ++i)
        printf("%6f\n", b[i]);

    for (int i=0; i<3; ++i)
        printf("%6.2f\n", b[i]);
    return 0;
}
```

Default: 6 digits after the decimal point.

```
3.141590
27.625000
100.750000
  3.14
 27.62
100.75
```

Floating Point Number in C++

```
#include <iostream>
#include <iomanip>
using std::setw;
using std::setprecision;
using std::cout;
using std::endl;

int main()
{
   float b[] = {3.14159, .5, 100.75};
   for (int i=0; i<3; ++i)
      cout << setw(3) << b[i] << endl;

   for (int i=0; i<3; ++i)
      cout << setprecision(3) << b[i] << endl;

   for (int i=0; i<3; ++i)
      cout << std::fixed << setprecision(3) << b[i] << endl;

   for (int i=0; i<3; ++i)
      cout << setw(7) << std::fixed << setprecision(3) << b[i] << endl;
   return 0;
}
```

```
3.14159
0.5
100.75
3.14
0.5
101
3.142
0.500
100.750
  3.142
  0.500
100.750
```

Left-Justify

```
#include <stdio.h>

int main() {
    char a[] = "Alfa";
    char b[] = "Bravo";
    char c[] = "Charlie";
    printf("|%10s|\n", a);
    printf("|%10s|\n", b);
    printf("|%10s|\n", c);
    printf("|%-10s|\n", a);
    printf("|%-10s|\n", b);
    printf("|%-10s|\n", c);
    return 0;
}
```

```
|      Alfa|
|     Bravo|
|   Charlie|
|Alfa      |
|Bravo     |
|Charlie   |
```

Left-Justify in C++

```
#include <iostream>
#include <iomanip>
using std::setw;
using std::cout;
using std::endl;

int main()
{
   char a[] = "Alfa";
   char b[] = "Bravo";
   char c[] = "Charlie";
   cout << setw(10) << a << endl
      << setw(10) << b << endl
      << setw(10) << c << endl;

   cout << std::left
      << setw(10) << a << endl
      << setw(10) << b << endl
      << setw(10) << c << endl;
   return 0;
}
```

```
      Alfa
     Bravo
   Charlie
Alfa
Bravo
Charlie
```

File Stream

ifstream
ofstream

Q: fprintf() can output to a file

```c
#include <stdio.h>

int main() {
    FILE* fp = fopen("a.txt", "w");
    for (int i=1; i<=5; ++i) {
        for (int j=0; j<i; ++j) fprintf(fp, "*");
        fprintf(fp, "\n");
    }
    fclose(fp);
    return 0;
}
```

```
*
**
***
****
*****
```

A: C++ File Stream Does the Same

- With the same syntax as I/O Stream.

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

int main() {
    std::ofstream fout("b.txt");
    for (int i=1; i<=5; ++i) {
        for (int j=0; j<i; ++j) fout << '*';
        fout << '\n';
    }
    fout.close();
    return 0;
}
```

Q: If I want to extend the file ...

- A: Open the file using the Append mode.

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

int main() {
    std::ofstream fout("b.txt", std::ios::app);
    for (int i=1; i<=5; ++i) {
        for (int j=0; j<i; ++j) fout << '*';
        fout << '\n';
    }
    fout.close();
    return 0;
}
```

Q: How do I read input from a file?

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

int main() {
    std::ifstream f("c.txt");
    int n;
    while (f >> n)
        std::cout << n << std::endl;
    f.close();
    return 0;
}
```

Q: If I did not close the file?

```c
#include <stdio.h>
#include <unistd.h>

int main()
{
    unlink("a.txt");     // Remove the file

    FILE* fp = fopen("a.txt", "a");
    char msg[] = "First\n";
    fwrite(msg, sizeof(msg), 1, fp);
    // fclose(fp);       // If you do not close the file properly ...

    fp = fopen("a.txt", "a");
    char msg2[] = "Second\n";
    fwrite(msg2, sizeof(msg2), 1, fp);
    fclose(fp);
    return 0;
}
```

Something bad may happened.

C++ String

An object-oriented class with default value and many convenient member functions.

Strings vs. Character Arrays

- Actually, C only has `char` arrays

```
char [3][23] = {
     "Arnold Schwarzenegger",
     "Bill Gates",
     "Catherine Zeta-Jones"
     };
```

- How many bytes should be allocated for a string?
  - Too long - wasteful; too short - insufficient
- Not easy to operate
  - `strcat(a, b)`   vs. `a = a + b`

String Concatenation

```
#include <iostream>
using std::cout;

int main()
{
    char a[] = "Good afternoon. ";
    char b[] = "Bob. ";
    char c[] = "Charlie. ";
    char newline[] = "\n";
    char str[80] = "";
   [redacted]
    cout << str;
    return 0;
}
```

```
Good afternoon. Bob.
Good afternoon. Charlie.
```

Powerful C++ Class

```
#include <iostream>
#include <string>
using std::cout;
using std::string;

int main()
{
    string a = "Good afternoon. ";
    string b = "Bob. ";
    string c = "Charlie. ";
    string newline = "\n";
    string str;
    str = a + b + newline +
        a + c + newline;

    cout << str;
    return 0;
}
```

Strings

- C language only has null-terminated strings which is stored as character arrays.
  - char name[5] = "Mary";
- C++ provides a `string` data type which is much easier to use. This class provides a bunch of powerful functions.
  - You don’t need to worry about how many bytes you should allocate when you declare a string variable. It will adjust dynamically.

Creating String Objects

- string sentence = "This sentence is false.";
- string sentence("This sentence is false.");
- string bees(7, 'b');
  - string bees("bbbbbbb");
- string letters(bees);
- string part(sentence, 5, 11);
  - string part("sentence is");
  - the first character is at index position 0
- string names[] = { "Alice", "Bob" };
  - string arrays

Many constructors

Input a String

- Read a character string into a string object:
  - `string sentence;`
  - `cin >> sentence;`
- However, `cin` will ignores leading whitespaces, and also terminates input when you enter a space, so for the input “This is a book”, only “This” is read into the object.
- Use the getline() function:
  - `getline(cin, sentence);`

```
ifstream fsIn("abc.txt");
getline(fsIn, sentence);
```

the source of the input

the destination for the input

Input A String (P.730)

```
ifstream fsIn("abc.txt");
getline(fsIn, sentence);
```

- string s1;
- string s2;
- char s3[20];

- cin >> s1;
- getline(cin, s2);
- cin.getline(s3, 20);

Input:

```
Good morning.
Good afternoon.
```

Output:

```
Good
 morning.
Good afternoon.
```

Concatenating Strings

- Use the + operator to concatenate two string objects or a string object and a string literal.

```
#include <iostream>
#include <string>
using std::cout;
using std::endl;
using std::string;

int main()
{
    string sentence1 = "This";
    string sentence2 = "That";
    string combined = sentence1 + "\n" + sentence2;
    cout << combined << endl;
    return 0;
}
```

```
This
That
```

Concatenating Strings (2)

- You can also use the + operator to join a character to a string object
  - sentence = sentence + '\n';
  - sentence += '\n';
  - sentence += "\n";
- Length of a string
  - sentence.length() // returns an integer
  - sentence.empty() // returns true or false

Accessing Strings

- Access a character in a string

```
string sentence("Too many cooks spil the broth.");
for (int i = 0; i < sentence.length(); i++)
{
    if (' '  == sentence[i])
        sentence[i] = '*';
}
```

Subscripting is faster, but the validity of the index is not checked.

- Use the `at()` member function

```
string sentence("Too many cooks spil the broth.");
for (int i = 0; i < sentence.length(); i++)
{
    if (' '  == sentence.at(i))
        sentence.at(i) = '*';
}
```

Access a substring in a string

- Extract a part of an existing string object as a new string object.
  - string sentence("Too many cooks spoil the broth.");
  - string w = sentence.substr(4,10);
    // Extracts "many cooks"

Search Strings

- Four versions of the `find()` function:
  - `string phrase("So near and yet so far");`
  - `string str("So near");`
  - `cout << phrase.find(str) << endl;`
    `// Outputs 0 (starting position)`
  - `cout << phrase.find("so far") << endl;`
    `// Outputs 16`
  - `cout << phrase.find("so near") << endl;`
    `// Outputs string::npos = 4294967295 on MS VC++`
- The function returns the value `string::npos` if the item was not found.
  - The value of `string::npos` may vary with different C++ compilers, so you should always use `string::npos` and not the explicit value.

Search Strings (2)

- Searching from a specified position:
  - `string phrase("ABCDEABCDEABCDE");`
  - `cout << phrase.find("A");`
    `// Outputs 0`
  - `cout << phrase.find("A", 3);`
    `// Outputs 5`
  - `cout << phrase.find("A", 11);`
    `// Outputs string::npos = 4294967295`

Conversion to C-Style Pointer-Based char* Strings (P.743)

- For functions which expect legacy C-style char* parameters, this is convenient.
  - string house("Stark");
  - char ptr1[6] = { 0 };
  - house.copy(ptr1, house.length());
  - const char* ptr2 = house.c_str();
  - house = "Lannister";   // Modify the contents
  - cout << house << endl;
  - cout << ptr1 << endl;   // Are ptr1 and ptr2
  - cout << ptr2 << endl;   // affected?

copy (char* s, size_t len, size_t pos = 0)

Exercise: Parsing Rational Numbers

- hw-rational-7.html

- CRational a(1, 4);
- CRational b("3/4");
- CRational c = a + b; cout << c;

String Stream

I/O stream
File stream
String stream

String Streams (P.746)

- I/O Stream
  - cin >> a >> b >> c;
  - cout << a + b << endl;
- File Stream
  - infile >> a >> b >> c;
  - outfile << a+b << endl;

- String Stream (in-memory I/O)
  - `#include <sstream>`
  - `using std::istringstream;`
  - `using std::ostringstream;`

```
istringstream iss("1 2 3 4 5");
while (iss >> n)
    sum += n;
cout << sum << '\n';
```

Extract data from a string.

```
ostringstream oss;
for (int i=1; i<=5; ++i)
{
    oss << "a" << i << ".cpp";
    cout << oss.str() << '\n';
    oss.str(""); // Clear
}
```

Pack data into a string.

This is useful to synthesize a string with a combination of int and string.

```
#include <iostream>
#include <string>
#include <sstream>
#include <stdio.h>
using std::cout;
using std::string;
using std::ostringstream;

int main() {
    int a = 10, b = 20, c = a + b;
    char buffer[80];
    sprintf(buffer, "%d + %d = %d", a, b, c);

    ostringstream oss;
    oss << a << " + " << b << " = " << c;
    string result = oss.str();

    cout << buffer << '\n'
        << result << '\n';

    return 0;
}
```